CHDR Inventory System

Group 3

Fall 2021 - Spring 2022

The CHDR Inventory System is a collaborative effort between:

Mina Beshay, Jeffrey Cherisma, Cameron Cuff, Gavin Gilbert, and Matthew Waldrep

Table of Contents

Table of Contents **2**

1 Executive Summary **1**

2 Project Overview **2**

2.1 Project Description 2

2.2 Statements of Motivation 3

2.3 Goals and Objectives 6

2.3.1 Goals and Objectives 6

2.3.2 Constraints 6

2.4 Broader Impacts 7

2.5 Legal, Ethical, and Privacy Issues 7

2.5.1 Legal Issues 7

2.5.2 Ethical Issues 8

2.5.3 Privacy Issues 9

3 Project Requirements and Specifications **9**

3.1 General Requirements 9

3.2 Check-in/Check-out 10

3.3 Scheduling 10

3.4 Inventory 11

4 Diagrams **12**

4.1 Block diagrams 12

4.1.1 Backend Inventory System (Server) Diagram 12

4.1.2 Web Application (Client) Diagram 13

5 Project Members, Duties, and Sponsors 14

5.1 Project Members and Duties 14

5.1.1 Mina Beshay 14

5.1.2 Matthew Waldrep 15

5.1.3 Cameron Cuff 15

5.1.4 Jeffrey Cherisma 15

5.1.5 Gavin Gilbert 15

5.2 Sponsors 16

5.2.1 Amy Giroux 16

5.2.2 Connie Harper 16

5.2.3 Mike Shier 16

6 Project Ideas and Potential Directions 17

6.1 Cameron Cuff 17

6.2 Matthew Waldrep 17

6.3 Gavin Gilbert 17

6.4 Jeffrey Cherisma 18

6.5 Mina Beshay 18

7 Project Budgeting and Financing 18

8 Project Research **19**

8.1 Database research 19

8.1.1 Database Summary 19

8.1.2 Database Management System Overview 19

8.1.3 Characteristics of a Good Database Management System 20

8.1.3.1 Real World Entity 20

8.1.3.2 Self-Describing Nature 21

8.1.3.3 Supporting A.C.I.D. Compliant Transactions 21

8.1.3.3.1 Atomicity 23

8.1.3.3.2 Consistency 23

8.1.3.3.3 Isolation 23

8.1.3.3.4 Durability 24

8.1.3.4 Concurrent Use of a Database 24

8.1.3.5 Insulation Between Data and Program 24

8.1.3.6 Transactions 25

8.1.3.6.1 Create 26

8.1.3.6.2 Insert 26

8.1.3.6.3 Select 26

8.1.3.6.4 Update 26

8.1.3.6.5 Delete 27

8.1.3.7 Data Persistence 27

8.1.3.8 Backup and Recovery 28

8.1.3.9 Data Integrity 29

8.1.3.9.1 Physical Integrity 29

8.1.3.9.2 Entity Integrity 29

8.1.3.9.3 Referential Integrity 30

8.1.3.9.3.1 SET NULL 31

8.1.3.9.3.2 CASCADE 31

8.1.3.9.3.3 SET DEFAULT 31

8.1.3.9.3.4 NO ACTION 32

8.1.3.9.4 Domain Integrity 32

8.1.3.9.4.1 The Data Type and The Length 32

8.1.3.9.4.2 NULL Value Acceptance 33

8.1.3.9.4.3 Allowable Values 33

8.1.3.9.4.4 Default Values 34

8.1.3.9.5 User-Defined Data Integrity 34

8.1.3.10 Multiple Views 34

8.1.3.11 Stores Any Kind of Data 36

8.1.3.12 Security 37

8.1.3.13 Represents Complex Relationship Between Data 37

8.1.3.14 Query Language 38

8.1.3.15 Miscellaneous Properties 39

8.1.3.15.1 Sharding 40

8.1.3.15.2 Scalability 41

8.1.3.15.3 Indexing 42

8.1.4 Database Architecture Types 44

8.1.4.1 Relational Databases 44

8.1.4.2 Non-Relational Databases 45

8.2 Backend 45

8.2.1 Backend overview 45

8.2.2 Server Management 46

8.2.3 Flask 47

8.2.4 Flask Implementation 48

8.2.5 HTTP Methods 50

8.2.5.1 GET 50

8.2.5.2 POST 50

8.2.5.3 PUT 50

8.2.5.4 DELETE 51

8.2.6 HTTP Status Codes 51

8.2.7 Testing 53

8.2.7.1 Guides to testing 54

8.2.7.2 Integration Testing 55

8.2.7.3 Functionality Testing 55

8.2.7.4 Usability Testing 55

8.2.7.5 Interface Testing 56

8.2.7.6 Compatibility Testing 56

8.2.7.7 Performance Testing 57

8.2.7.8 Security Testing 57

8.2.7.9 Automated testing 57

8.2.8 GitHub and Git 59

8.2.8.1 Version Control 60

8.2.8.2 Types Version Control 62

8.2.8.2.1 Centralized Version Control 62

8.2.8.2.2 Distributed Version Control 63

8.2.9 GIT Implementation 64

8.2.10 Insomnia 65

8.3 Website Research 65

8.3.1 An Analysis of Layout, White Space, and Accessibility 65

8.3.2 Influencing Decisions with Visual Hierarchy 66

8.3.3 White Space and its Benefits 69

8.3.4 The Importance of Reachability 72

8.3.5 Considerations in Accessibility 74

8.3.6 A General Categorization of Various Typefaces 75

8.3.7 A Brief Comparison of Color Models 78

8.3.8 The Significance of Color Harmony 80

8.3.9 The Psychological Effects of Color 81

8.4 Mobile App Research 82

8.4.1 Expo 82

8.4.1.1 History 82

8.4.1.2 Installation 83

8.4.1.3 Camera 84

8.4.2 React Native 85

8.4.3 How to connect a React Native app to Apache Server? 85

8.4.4 How to make a barcode scanner in React Native? 86

8.4.5 How to test and export react native to work on IOS? 88

8.5 Image Processing 89

8.5.1 Why Image Recognition? 89

8.5.2 Machine Learning Summary 89

8.5.3 Methods of Machine Learning 91

8.5.4 Image Recognition Summary 91

8.5.5 Machine Learning Image Recognition Models 92

8.6 Mobile View 92

8.6.1 Mobile Web Design 93

8.6.1.1 Responsive Web Design 93

8.6.1.2 Adaptive Web Design 94

8.6.2 Key Practices for Mobile Web Design 95

8.6.2.1 Accessibility 95

8.6.2.2 Simplicity 96

8.7 Secure Password Storage 96

8.7.1 Hashing 96

8.7.1.1 Straight Hashing 96

8.7.1.2 Salting 97

8.7.1.3 Slow Hashing and Multi-Hashing 98

8.7.2 Secure User Passwords 98

8.7.2.1 Four Random Words 99

8.7.2.2 Phrases 99

8.7.2.3 Acronyms 100

8.7.2.4 Alternate Keyboard Layouts 100

8.7.2.5 Basic Ciphers 101

8.7.2.6 Miscellaneous Methods 101

8.7.2.7 Password Managers 101

9 Project Design **102**

9.1 Database 102

9.1.1 Database Design Summary 102

9.1.2 Database Design Requirements 102

9.1.2.1 Requirement 1 103

9.1.2.2 Requirement 2 103

9.1.2.3 Requirement 3 103

9.1.2.4 Requirement 4 103

9.1.2.5 Requirement 5 103

9.1.2.6 Requirement 6 103

9.1.2.7 Requirement 7 104

9.1.2.8 Requirement 8 104

9.1.2.9 Requirement 9 104

9.1.2.10 Requirement 10 104

9.1.2.11 Requirement 11 104

9.1.2.12 Requirement 12 105

9.1.3 Entity Relationship Diagram 105

9.1.3.1 User Table 105

9.1.3.2 Reservation Table 106

9.1.3.3 Item Table 106

9.1.3.3 Item Child Table 107

9.1.3.3 Item Image Table 107

9.2 Backend 107

9.2.1 Packages 107

9.2.2 Blueprints 108

9.2.2.1 Static Files 109

9.2.2.2 Templates 109
9.2.2 Python Testing Frameworks 110
9.2.2.1 Robot 111
9.2.2.2 PyTest 112
9.2.2.3 Unittest 113
9.2.2.4 DocTest 114
9.2.2.5 Nose2 115
9.2.2.6 Testify 116
9.2.2.7 Behave 118
9.3 Website Design 119
9.3.1 Choosing an Appropriate Color Palette 119
9.3.2 Considerations in Logos 122
9.3.3 A Comparison of UI Libraries 126
9.3.3.1 Bootstrap 126
9.3.3.2 Ant Design 127
9.3.3.4 Material UI 127
9.3.3.5 Materialize.css 128
9.3.3.6 Chakra UI 128
9.3.3.7 Fluent UI 128
9.3.3.8 Semantic UI React 129
9.3.4 A Comparison of Front End Hosting Services 129

9.3.4.1 Apache - Self Hosting 130

9.3.4.2 AWS Amplify Hosting 130

9.3.4.3 Netlify 130

9.3.4.4 Vercel 131

9.3.5 Testing the Front End 131

9.3.5.1 Test Driven Development - TDD 132

9.3.5.2 Types of Testing for Front End Applications 132

9.3.5.3 A Comparison of Testing Libraries and Frameworks for Front End 133

9.3.6 TypeScript vs. Javascript: Choosing an Appropriate Language 134

9.3.7 UI Mockups 135

9.4 Mobile App 141

10 Project Milestones 142

11 Project Summary, Conclusion, and Post-Thoughts 143

11.1 Project Summary 143

11.2 Lessons Learned and Insights Gained 144

11.2.1 Importance of Planning 144

11.2.2 Knowledge Gained on the Various Technologies Used in the Project 144

11.2.3 Interpreting Sponsor Requirements 144

11.3 Overview of Completeness 144

12 References 144

1 Executive Summary

At the start of the 2021 fall semester, our group formed around and accepted the Center for Humanities Digital Research Lab's project of designing an inventory management web and mobile application with application programming interface and database support. The objective of this project was to provide the lab with a system to distribute their recently acquired equipment amongst whatever students may require use of it. The application will allow a web portal for students to view and reserve equipment, and it will allow the administrators a web and mobile portal to check and inventory items.

The Center for Humanities Digital Research Lab received a $500,000 dollar grant for the aforementioned equipment. This equipment consists of a variety of objects, some moveable and others stationary, such as cameras, microfilm scanners, and virtual reality headsets. In accordance with the grantholders' requirement, this equipment is to be distributed to any and all students who require use of it in an orderly fashion, ensuring that records are maintained pertaining to the usage of each individual item.

As aforementioned, our group is tasked with creating an application with mobile and web components that is connected to a database. While the database is powered by MySQL, the other components of the application are ultimately up to us as a group. This gave us a unique opportunity to explore a variety of different tools and components for the assembly of the front end, back end, and mobile component. Those not involved with the database explored several different options and each arrived at a conclusion for their chosen tool that was debated upon and accepted by the group. As for the database, it was a unique opportunity to really delve into what makes a database management system work and the different options a database engineer has in constructing a database.

Overall, we chose to use a front-end powered by React for the web application and React Native for the mobile application. This is because React Native supports both iOS and Android, allowing us to only develop one application for both platforms rather than one for each. We also decided on Flask for use as our application programming interface. Flask is powered by Python and is also supported across multiple systems. Finally, even though MySQL was a requirement, we did explore other options in case there was a system that would suit our needs better. As expected though, we came to the conclusion that MySQL was, in fact, the best choice for our application.

2 Project Overview

2.1 Project Description

In recent years, the University of Central Florida College of Arts and Humanities received a substantial grant of $500,000 United States dollars from their endowment. As per the requests of those that originated the grant, the money is to be used in the creation and maintenance of an educational multipurpose gathering space for students affiliated with the College. This state-of-the-art facility will house a variety of amenities and equipment, all of which are readily available to students who wish to use them. The aforementioned items include, but are not limited to format scanners, microfilm scanners, and laser scanners. The facility also offers a variety of conference and study rooms which are equally available for use by the students. Unfortunately, there is presently no implemented method for students to actually reserve these items and rooms.

The objective of this project is to provide a basic scheduling and inventorying system for the equipment and items in this facility. We will create a web application which will enable students to reserve and check the equipment that has been made available to them. Students will have options to reserve anything from rooms to scanners for a future time period, and once that time period has arrived (assuming there are no conflicts with administrators or other students), the system will allow the student to either check that item out of inventory or if the item is immovable, use it during the allotted time period. For removable items, the student will be prompted to return the item to the staff once their time has expired, though it will certainly be possible for students to return the item before. Administrators will be able to view all incoming requests as well as the current inventory of items, and they can either approve or deny any of the incoming requests. Additionally, administrators will have access to a specialized mobile component that will enable them to interact with the identifying labels on the equipment to check them in or out, and also to inventory them. Administrators will also have access to the usage statistics on all of the equipment, which they can use to demonstrate to the grant providers that their money is being used appropriately.

Specifically, the overall project will have four components: database, application programming interface (API), a web front end, and a mobile app. The database is a relational database, powered by MySQL, which is accessed through phpMyAdmin on a specialized development server (in case any adjustments or updates must be made). The database will contain all of the information pertaining to inventory, users, and reservations on the items in said inventory. The database will also contain a log on all item usage (different from usage statistics) so that administrators will not only be able to see what items are currently in use, but also which items have been checked out by whom. The database will be connected to the front ends of both

the mobile app and the web app using a Flask API. As already stated, the website will offer the bulk of the features of this project, enabling users to reserve and check items, and it will be accessible to both administrators and students. The mobile app, on the other hand, will only be accessible to administrators, and it will function as a de facto barcode scanner, using the phone's camera to scan a text or barcode label to identify and inventory objects.

2.2 Statements of Motivation

I have chosen the CHDR Inventory Project for a couple of reasons. When it was presented to me in class, I knew this was the one. I've always been interested in learning to make a check-in/check-out/inventory program for hotels. In COP 4331: Procedures of Object-Oriented Software Development with Professor Leinicker, I suggested we make a guest mobile ordering system for hotels to my group. It would help hotel staff improve their communication and automate some front desk/management tasks. They agreed and we made a pretty cool project. I planned to work on the project after the course ended because I feel like it would be a helpful app for hotels to use. Most of the features of the hotel project are incorporated in this project. So I'd be killing two birds with one stone.

Another reason I am interested in this project is that I feel it would give me a broader understanding/ experience in app development. It has been one of my dreams to create an app and publish it in the google/app store.

I've always had a passion for helping people and just connecting with them. This has led me to my calling which is helping small businesses achieve their goals via software/apps. I will help them by making software that will help improve productivity while satisfying all their customers' needs. This project will help me understand the world of business and the requirements businesses need. It will also help me get my foot in the door of the business software development field.

These are just a couple of reasons why I'm motivated and excited to work on this project. I can't wait to get started and learn more about each one of my teammates and sponsors.

- Mina Beshay

I have spent over three years at this university learning about Computer Science and its related topics. I've learned everything from algorithms and data structures to hardware components and binary logic. The overwhelming majority of my coursework consists of smaller

assignments designed specifically to test my grasp and understanding of those aforementioned concepts. There is a problem, however: my coursework only tests those concepts in isolation, and as anyone who has spent time in the practical world of computer science knows, nothing is done in isolation. Every part of a computer and its related programs are interconnected with one another, and working with those various technologies to create a unique and functional product is very much different from writing an algorithm to implement a linked list or quicksort. I am motivated for Senior Design so that I might be able to understand how computer science is really conducted in the business world. Algorithms are tools, but one cannot build a house with just knowledge of tools alone.

My motivation for the Center for Humanities and Digital Research Lab project has several key points associated with it. One, the project is relatively straightforward and well within the scope of Senior Design. I feel that it is important for students to design and create something that enables them to use the skills they have assembled in college, and while many of the presented projects were interesting, I felt that they were way outside of what was expected for a senior in this major. Two, the project is affiliated with the university. I feel that a project affiliated with the university would have better resources for us (a development server and database already established) than one that wasn't. I feel more comfortable doing work for something that I am already familiar with, even though UCF is a massive institution with many branches. Finally, and this ties back to what I stated in my previous paragraph, I want experience on creating an actual software application that will be continuously used by people, as opposed to a small project that is purely for the point of a grade. It is important to me that I gain experience that is comparable to the real world, and I think with the straightforward scope of this project, I can fully appreciate the process of creating a real software application and not get too bogged down with stressing out over things that are well beyond my paygrade as a college student.

- Matthew Waldrep

I went through most of my time in the Computer Science program without having an idea of what area of concentration that I wanted to focus on . However, after taking Project Object Oriented Programming this past summer It finally opened my eyes to the possibilities that one can accomplish with a Computer Science degree. I learned that a software application has multiple components to it and this is when I knew that back-end and database would be the areas of my expertise. During my time in Project Object Oriented Programming, I worked on a couple projects and dedicated my time to the back-end position and was motivated to start working on personal projects that would broaden my knowledge in that area. So, when I saw the presentation for the new Center for Humanities & Digital Research Lab (CHDR), I knew that this was the project that I wanted to dedicate my time to for the duration of Senior design.

This project essentially piqued my interest because it is similar to the projects that we worked on in Project Object Oriented Programing. I knew this would give me a sense of comfortability while putting me in a position to challenge my abilities. The stakes for this project are higher now, as I am giving the opportunity to experience working with a real client, in order to complete a working product to their liking. My goals working on this project are for me to grow my knowledge as a back-end developer, while expanding my knowledge on the mobile development side also. Developing functions such as login and signing up for an account are work that I have done before, but having to possibly connect to the University of Central Florida API's in order to register using the students and staff information will be new for me. Working on API's that require for you to scan a barcode and amount of data that we will be working with, will be challenging but those are the exact reasons that I knew that this project would be a great project to work on.

- Jeffrey Cherisma

Ever since starting my schooling here at the University of Central Florida, I've felt almost adrift in a sea of choices. For three years I wondered what in the world I would focus on, what my specialty would be, how I would differentiate myself from the competition. I visited some clubs, attended one of our hackathons, but I didn't really find my way. Once I got to POOSD, I realized I learned a lot more about back-end development, like databases and APIs. Now that we're here in SD1, I think I've finally decided on focusing on the back-end. Especially considering this is an inventory management system, I'm looking forward to sharpening my skills as a developer.

I was motivated to take this project because it's not only a good introduction to a diverse set of skills, a good resume builder, and a way to practice my back-end fundamentals, but also because it helps my fellow Knights at UCF. This inventory system will streamline the process for managing the check-out and return of items from the Center for Humanities and Digital Research, saving time and effort of both students and administrators. I hope that the effort that my team and I put into this project will come to improve the lives and the schooling of everybody involved in CHDR.

- Gavin Gilbert

As someone who enjoys creating web applications and experimenting with various technology, APIs, and framework, this project instantly piqued my interest. I think what makes this project so unique is the idea that there are so many different moving parts that need to come together in order to achieve a final result. As such, this seemed like a great opportunity to learn new skills. One skill I've been trying to improve on is UI/UX design. Unfortunately, there aren't many classes in the computer science curriculum that teach design so it's a skill students usually

have to learn on their own. One of the best ways to learn anything in the tech world is with hands-on practice. Thankfully, a real-world project is the best way to immerse myself into the design. Because the project I chose is so detailed and has pretty much every requirement clearly mapped out, it should be a bit easier to create mockups and designs, especially when requirements can guide the designs.

Working in groups is another extremely important skill in the tech world as well. A lot of the most valuable information I've learned has been from those who have had more experience than I. Another benefit of working in a group is that because each member has a diverse set of ideas, everyone can learn from each other. This project involves mobile application development, database planning, API testing, UI/UX design, and so much more. All of these aspects motivated me to choose this project.

- Cameron Cuff

2.3 Goals and Objectives

2.3.1 Goals and Objectives

- Provide a service for the Center for Humanities and Digital Research to manage their rental equipment for the purposes of checking-in/checking-out.
- Allow for students to view available items on the website and check them out at CHDR.
- Allow for administrators to track which items are checked out, view reservations on room-size equipment, and keep track of inventory and usage.
- Promote the safe use of CHDR's equipment by students for research purposes.

2.3.2 Constraints

- Product must scale to thousands of (possibly more) users.
- Product must utilize UCF NID login services.
- Project must be completed exclusively by the five team members listed before the senior design showcase at the end of the Spring 2021 semester.

2.4 Broader Impacts

In a vacuum, this application will push no boundaries, nor will it promote increased participation in STEM-related topics amongst underrepresented groups. Unlike many of the projects presented to the class, this project is not some extremely sophisticated, cutting edge, innovative, or ambitious endeavor. It is reasonable to assume that there are many similar pieces of software available that can perform the functionality of our project with varying degrees of effectiveness. By itself, our software is simply a tool for staff members of the Center for Humanities and Digital Research to allow students access to their recently purchased equipment, and it is on that point which this software application will have some broader, albeit localized, impacts on engagement in STEM, or at the very least enhanced education, among a diverse variety of groups and individuals alike.

In practical use, this application will enable students to gain access to an extensive set of equipment ranging from scanning devices to study and conference rooms. Without our application, it would be exceedingly difficult for the staff of CHDR to allocate their equipment to a large number of students the facility will likely service over the coming years. By enabling access to the equipment, students will have the ability to enhance their education experience in a way that is active and hands-on. It is important to note that access to this equipment is complementary to the student's enrollment in the university. This unique experience for students to use specialized equipment to further their academic studies will allow a higher quality experience for all groups, especially those who previously have not had access to such equipment. It will hopefully motivate a greater number of students to continue their research in whatever their field of study is.

2.5 Legal, Ethical, and Privacy Issues

2.5.1 Legal Issues

Some exploratory research has identified a litany of various patents that either partially or entirely apply to the components of this project. For brevity's sake, we have listed several, all of which apply to the major functionality of our application:

- US7156311B2 - Claims a method of decoding a barcode using a mobile device. It discusses how one takes a picture of the barcode with their phone, and then they digitally enhance and decode the image into its respective bits. [1]

- US9033243B2 - Claims a method of bonding a barcode to a mobile communication device. It discusses how a mobile device can read and decode a barcode and then use that barcode to create an association between itself and the object the barcode is affixed to. [2]

- US10915857B2 - Claims an electronic supply management system in which barcodes are scanned and transmitted to a server over a network. The server then takes that data and processes it, later transmitting it to a mobile device on which it is displayed in a readable manner. [3]

Again, these represent only a few patents we could easily find. It is interesting to note that they are all related to the use of a barcode and barcode scanner. Despite the similarities, it is unlikely we will encounter any patent infringement issues with the aforementioned patents. US7156311B2, which is the most similar to our project's functionality, has recently expired in the United States, and our analysis has failed to find any information on if the patent has been renewed. US9033243B2 and US10915857B2 both entail barcode scanning, but they are not directly related to our project in the specifics of what ends to that functionality they are claiming. The former claims a method of binding a barcode to a mobile communication device, whereas the latter deals with supply management, whereas our project deals in equipment rentals. This is the extent of our patent research; anything further would best be done by someone who is well versed in patent law.

2.5.2 Ethical Issues

Due to the simple and straightforward nature of this particular project, it is highly unlikely there are any ethical concerns involved. This project is in no way ethically ambitious or questionable. The system by itself is so direct in its functionality that it could be considered ethically neutral. The only possible ethical issue that could arise would be if the equipment that was rented through the system was used for unethical purposes. This isn't a direct concern however as the business the student conducts with the rental equipment is unreachable to us. Therefore, we would consider it outside of the scope of this project.

2.5.3 Privacy Issues

There are some privacy-related concerns that must be addressed. The biggest concern is the fact that our database will contain NID information related to students and staff as well as their UCF affiliated email addresses. To protect users, we will encrypt their passwords using a one-way hash on our database. User emails will also be stored on the database for the sole purpose of informing them on topics related to their equipment reservations and rental due dates. We will also collect usage statistics relating to each of the registered equipment, and we will maintain a log of rentals with the users who rented that equipment out. All of this information

will be kept on the database, and sensitive information will only be accessible to database administrators. Information will only be shared in accordance with the university's privacy policy, as this application will be affiliated with the university's system.

3 Project Requirements and Specifications

3.1 General Requirements

- Database
    - Implemented using a relational database: MySQL
    - Hosted on an Apache server
    - Contains all rentable items along with their affiliated purchase and location information
    - Contains all users and their rankings in the system; rankings are superuser, admin, and user
    - Contains a log of all rented items and their affiliated renters
- API
    - Built using REACT
    - Will transmit any information to and from front end and database
- Web Application
    - Allows for all users to access the forward-mentioned requirements in sections 3.2-3.4
- Mobile Application
    - Allows administrators inventory management and barcode scanning functionality; see sections 3.2-3.4

3.2 Check-in/Check-out

- Administrators should be able to view an item and see who has the item checked out, along with other pertinent details such as due date, condition, etc.
- Students should be able to view an item's availability with the mobile app.
- The system should email students at predetermined intervals to notify them of upcoming due dates regarding specific items they have checked out.
- Administrators should utilize their mobile phone camera to scan item labels for easy lookup of information, and for the checking out of items.
- Students should be able to view items they have previously checked out.

3.3 Scheduling

The scheduling requirement will allow for users to check out items that are able to leave the premise and block out certain times of the day for items that are not allowed to leave the premise. Administrators will also have the option to block off the times for certain items and users will be able to view which items that are available to them to reserve. All this will be done with a system that allows for reservations to be made and where an administrator is also able to make that decision to accept or to reject, which both parties will be able to view. After which, both parties will be notified by an email that will confirm in detail the reservation to the user and one to the administrator, so they are able to accept it. It will then automatically allow the items to block off the scheduling report.

Front-end:

- Scheduling page that displays items and their availability status.
    - Place-based items have time slots available to be viewed.
    - Button that allows for users to reserve spaces.
- A page able to be viewed by administrators that displays users and the items they have checked out.

Back-end:

- Create a database that holds the users and administrators.
- Create a database that holds all the inventory items.
    - Placed-based items
    - Removable items
- Provide endpoints that connect to the database to retrieve the availability of certain items to depict the status of the items to the users and administrators.
    - Administrators can see which users reserved items and spots.
    - Administrators should be able to approve user requests.
    - Users can see which items are available to reserve.
- Endpoint that connects to our schedule database that shows the status of availability.
    - Administrators can block certain time slots.
- Endpoint that integrates with CHDR outlook calendar that sends an Icalendar file to the user and Administrator.
    - Sent to the user to confirm and remind them of the reservation.
    - Sent to the administrator to block off the time slot from the schedule.

3.4 Inventory

Inventory requirements mostly focus on the backend portion of the project specifically the database. From there I will make viable APIs necessary for it to function. Then I'll make the frontend portion requirements for the user to see.

- Database
    - Collection of items
        - Name: (Name of item)
        - Location: (Where is it located in the facility)
        - Quantity: (Item quantity)
        - Mark: (whether item can be taken out or not)
        - Image: (link to picture of item)
        - numCheckOut: (numbers of times the item is checked out)
        - scanDate: (date item was scanned into facility)
        - numScan: (number of times the item was scanned)
        - scanMonth: (Month the item was scanned into facility)
        - VendorName: (Name of vendor)
        - vendorPurchase: (how much the vendor paid for the item)
        - vendorPurchaseDate: (When vendor bought item)
        - Warranty: (Warranty info)
- APIs
    - API to keep track of scanDate and numScan: Every time the user scans the item it is recorded and the API uses it to make statistics for the month. Then it resets it to 0 after the month ends.
    - MarkAPI: Every time the admin scans the item into the facility, Mark becomes true, otherwise it's false.
    - usageStats: displays numCheckOut
    - Make an API that tracks whether the item is checked out or in
- Front-End
    - Have a usage stats button that when pressed runs usageStats.
    - Have a button for every API or make a report button that puts all the APIs in one document.

4 Diagrams

4.1 Block diagrams

4.1.1 Backend Inventory System (Server) Diagram

The following is a block diagram that specifies the details of the CHDR backend inventory system. The team members responsible for the portion of the project are Jeffrey Cherisma, Gavin Gilbert, and Matthew Waldrep.

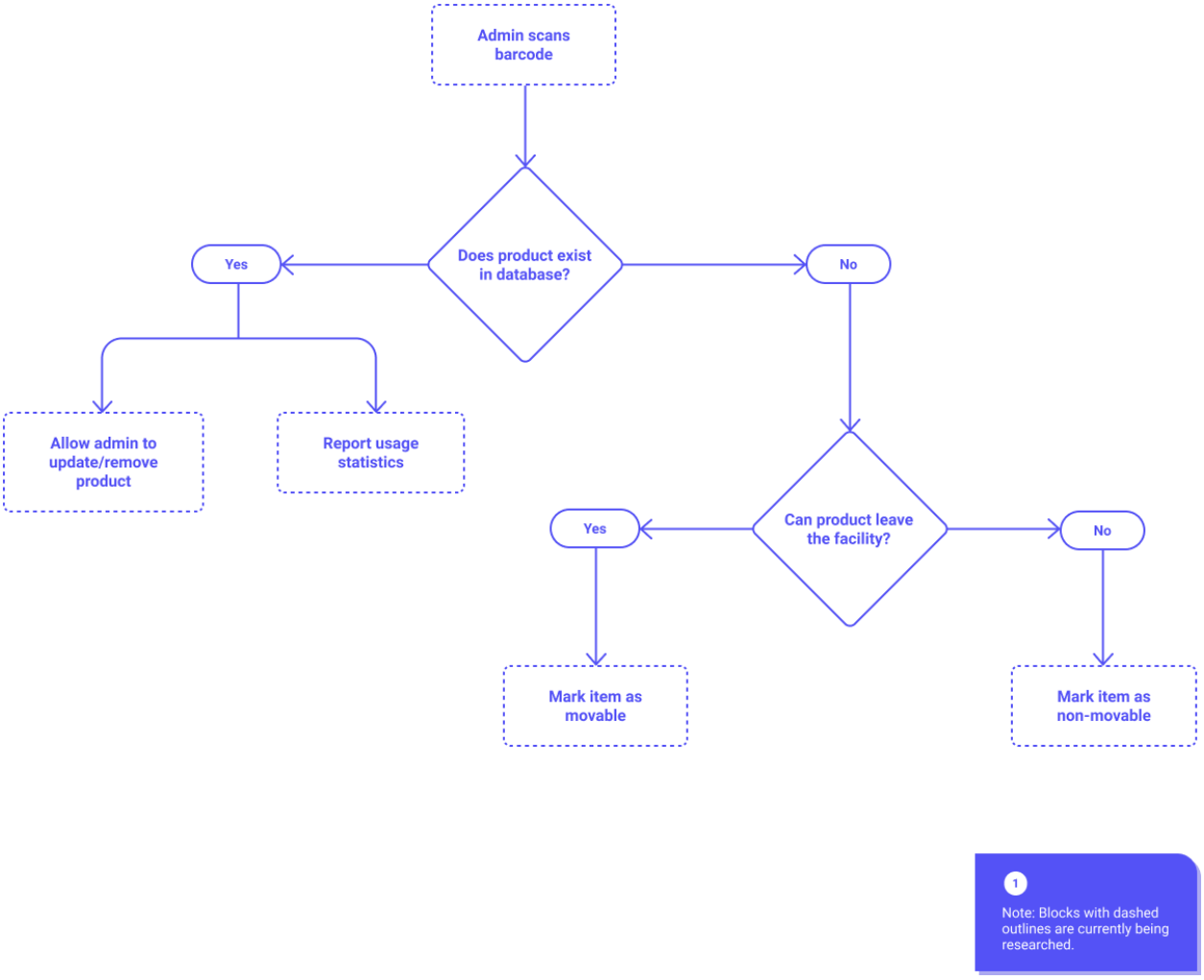


Figure 4.1.1.1 - A block diagram detailing barcode scanning log

It should be noted that in figure 5.1, a non-movable item is defined as any item that cannot leave the building, i.e., a room that is to be booked at a specific time or machinery that may be too heavy to leave the facility. In contrast, a movable item is defined as any item that may leave the building.

4.1.2 Web Application (Client) Diagram

The following is a block diagram that illustrates the flow of the CHDR web application. This is primarily used by those wanting to check out an item, reserve a room space, return an item, or check the availability of an item or room space. This portion of the project will be

developed by Mina Beshay and Cameron Cuff. It should be noted that, as in the previous diagram, blocks with a dashed outline are currently in research. Blocks colored purple are to be developed by members of the team responsible for the backend inventory system.

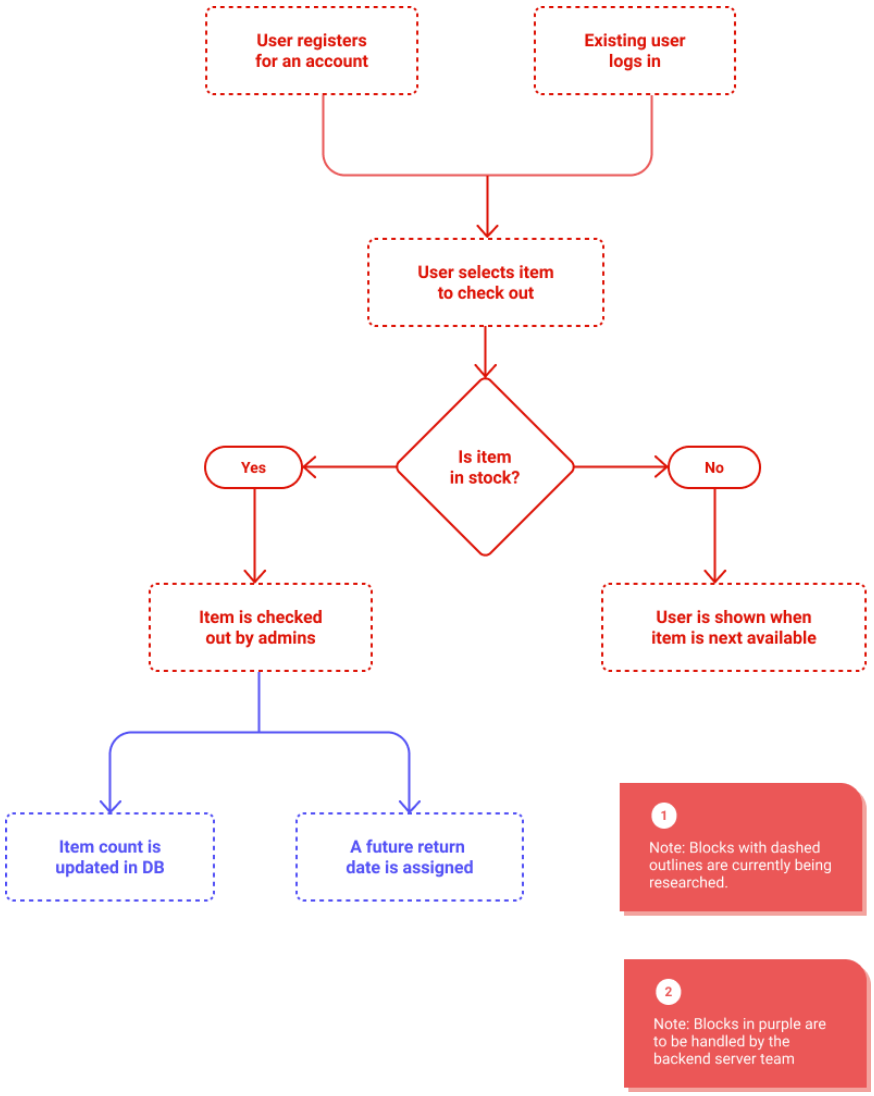


Figure 4.1.2.1 - A block diagram detailing the potential user authentication workflow

5 Project Members, Duties, and Sponsors

5.1 Project Members and Duties

5.1.1 Mina Beshay

- Communication: In charge of all communication with sponsors including emails and setting up zoom meetings bi-weekly.

- Organizer/Secretary: Keep the team on track. Making a Google calendar to see when we can meet, Making Github, Google Drive and other tools to keep data organized.
- Wrote Sections 3.4 Inventory requirements, Section 5
- Contributed to Sections 2.2: Statements of Motivation, Section 6.5: Project Ideas and potential directions and Section 10: Project Milestones
- Contributed to Mobile App Research and Design Sections

5.1.2 Matthew Waldrep

- Scribe: Taking notes during sponsor and group meetings. Making agendas before meetings and distributing them.
- Contributed to Sections 2.2: Statements of Motivation, Section 6: Project Ideas and potential directions
- Wrote Sections 2.1, 2.4, 2.5, 3.1, 7.
- Contributed to Database Research and Design Sections

5.1.3 Cameron Cuff

- Created and wrote Section 4 block diagrams for backend server and web application
- Contributed to Sections 2.2: Statements of Motivation, Section 6.1: Project Ideas and potential directions
- Contributed to Website Research, Website Design, and created UI mockups for website

5.1.4 Jeffrey Cherisma

- Wrote Section 3.3: Scheduling requirements
- Contributed to Sections 2.2: Statements of Motivation, Section 9: Project Milestones and Section 6.4: Project Ideas and potential directions

5.1.5 Gavin Gilbert

- Contributed to Sections 2.2: Statements of Motivation, Section 9: Project Milestones and Section 6.3: Project Ideas and potential directions
- Wrote Sections:
    - 2.3 - Goals and Objectives; Constraints
    - 3.2 - Check-in / Check-out Requirements
    - 8.6 - Image Processing
    - 8.7 - Mobile View
    - 8.8 - Secure Password Storage
    - 9.2.2 - Python Testing Frameworks

5.2 Sponsors

We have three sponsors for this project. Our Sponsors are very helpful and have set specific requirements for this project. Our three sponsors are:

5.2.1 Amy Giroux

- Amy Giroux: Associate Director of the Center for Humanities and Digital Research

  Amy Larner Giroux, Ph.D., is Associate Director of the Center for Humanities and Digital Research at the University of Central Florida (UCF) and a Digital Historian for the National Cemetery Administration, part of the Department of Veterans Affairs. Her research involves the contact zone between humans and technology within the intersections of history and learning. By leveraging technologies such as Augmented Reality and Virtual Reality, she brings learning to both classroom and field. Her specialty is cemetery research and she has been involved in both recording the physical aspects of the space through technology such as drone photogrammetry and 360º imagery and also researching the lives of those interred through genealogical research as she is a board-certified genealogist. Her research in St. Augustine National Cemetery has brought to light new information on Civil War soldiers, Native American prisoners of war, and Seminole War history. Giroux earned her doctorate in Texts and Technology from UCF and her dissertation included a digital project which mapped historical artifacts geo-spatially on Google Earth. She is currently the technical lead for UCF's Veterans Legacy Program. [4]

5.2.2 Connie Harper

- Connie Harper: Software Developer,

5.2.3 Mike Shier

- Mike Shier: Journals Manager for the College of Arts and Humanities

  Mike Shier is the Journals Manager for the College of Arts and Humanities at UCF and a Research Specialist in its Center for Humanities and Digital Research. He holds a Ph.D.

in English Studies from Illinois State University and an MFA in Creative Writing from Florida Atlantic University. He is also the Creative Nonfiction editor of *The Florida Review*. [5]

6 Project Ideas and Potential Directions

6.1 Cameron Cuff

- If our mobile application utilizes React Native, that will allow us to target both iOS and Android devices.
- Expo, an open-source platform that aids in React Native development, has various libraries that interface with mobile devices. As such, one of those libraries involves the ability to scan various types of barcodes.

6.2 Matthew Waldrep

- Use phpMyAdmin to reverse engineer a database schema to confirm that it matches our approved ERD
- Construct the ERD using Draw.io
- If a field in a table in the database must be present, we can use the SQL attribute of NOT NULL to ensure that the field is present
- SQLTest, a database unit testing application that can be used to test our SQL database in isolation; I may write my own scripts to test the database.

6.3 Gavin Gilbert

- System Design mockups using WondershareEDraw
- For the Large Screen Calendar showing reservation times for large items in CHDR's main office, we could use a system nearly identical to the one that the UCF Library uses for its study room reservations.
- Because CHDR wants us to use UCF NID login services, we should get in touch with UCF IT to see what the process for that would be, and perhaps get their assistance in implementing it.

6.4 Jeffrey Cherisma

- Audit check of all the inventory items that we will be working on.
- Might be best to use a part of a stack that makes it easier for the front end and back end to merge.
- Using flutter might be a good route to go for the mobile side, if we end up using react.

6.5 Mina Beshay

- Make a MERN stack but instead of using MongoDB we would use MYSQL
- Make a LAMP stack but instead of using HTML/CSS for the website we would use react.
- For NID login we can explore how the IT department or library or even the Recreational Wellness Center at UCF does it. We can import the API if it exists
- For NID login we can make an api that goes to UCF's webcourses login to login and directs the user back to our website after login
- Use React Native to make the mobile app then export it to an .apk file which admins of the Humanities facility will be able to download on github.
- Instead of using image processing to determine the journals we can have the same barcode for all of the same books with a quantity field in the database. When a user checks out the item, the quantity will decrease by one and so on.
- Keep group members focused and organized via Jira.

7 Project Budgeting and Financing

This project's development server, hosting, and database are all covered by the College of Arts and Humanities. Considering the fact that practically the entire project is already financed, we do not expect to have any major expenses throughout its development. We have not identified any software licenses that we need to buy, nor do we intend to purchase any graphical elements for the site. Should any additional complex graphical components be required, project sponsor Michael Shier has offered to acquire and provide any of those needed assets. If any unexpected expenses should come up during the duration of this project, we will follow the listed steps for handling those expenses. One, we will consult our sponsors on the expense. Two, together, we will determine if this expense is avoidable or if it is necessary. Three, we will

determine whether or not the expense is something for the sponsors to cover. Finally, if the sponsors choose to not cover the expense, it will be divided up evenly amongst ourselves.

8 Project Research

8.1 Database research

8.1.1 Database Summary

As per the requirements of our sponsors, this application requires the use of a database. Furthermore, our sponsors requested that our database be powered by MySQL, one of the most common database services available to the public. Despite the requirements, our group firmly believes in understanding why a particular decision is an ideal choice and what that decision's alternatives are, as it is entirely possible we might find and propose a solution that could fit our sponsor's needs better than their solution.

In order to accomplish this, we started by exploring the fundamentals of database management systems. We explored the characteristics of a database management system and the qualities that differentiate an excellent system from a poor one. We then explored the two differing architecture types of commercial database management systems, aiming to understand how each of those worked. The objective of all of this research and exploration was to understand how the underlying technology of a database management system works, and why MySQL is the best choice for this particular project. We also explored all of this to determine what the best practices are for us engineers so that we can deliver a safe and secure product to our project sponsors.

8.1.2 Database Management System Overview

First and foremost, it is absolutely imperative to understand what a database management system is; not knowing so would be the equivalent of trying to drive a car without knowing what a car is. In short, a database management system is a computerized data-keeping system. [6] It is a system designed to store, organize, and manipulate data, regardless of the quantity of data. The way in which data is stored, however, is relevant to the database management system, as different systems are optimized for different types of data (as will be explained in later sections).

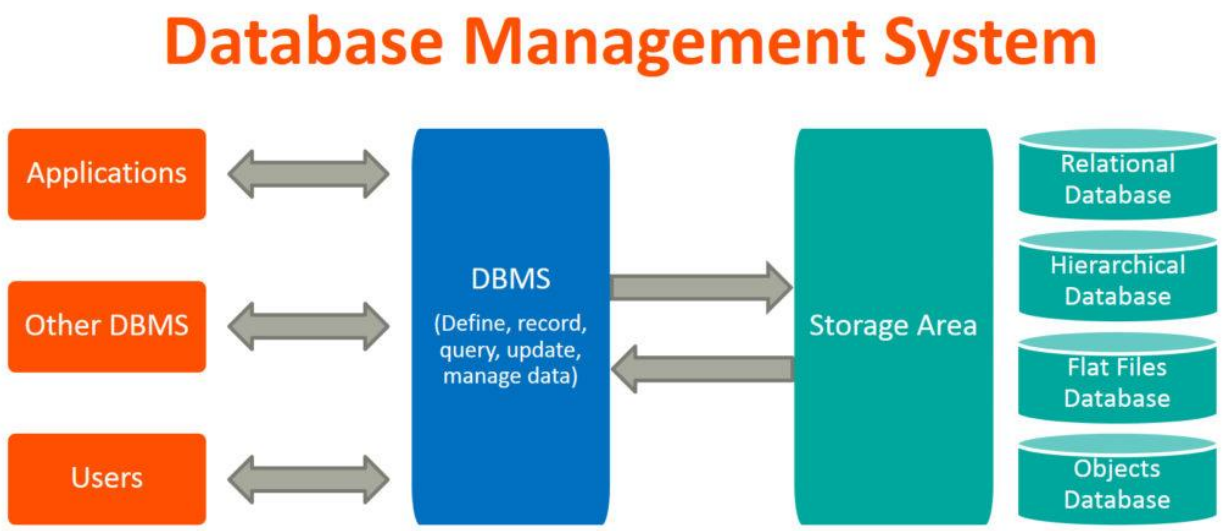


Figure 8.1.2.1: This is the basic workflow architecture of a database management system visualized in a high level diagram. As can be seen here, the concept is the same for any kind of database. [7]

Database management systems represent the highest level of abstraction when it comes to data management and storage. As such, all database management systems have three fundamental characteristics that define it: it contains interrelated data, it has a set of programs to access and manipulate that data, and the data is stored in such a way to where it can be easily retrieved and manipulated to obtain new information. [8] The primary goal of such a system is to provide a way to store and retrieve data in a way that is convenient and efficient.

8.1.3 Characteristics of a Good Database Management System

In addition to the previously mentioned required functions of a database management system, it must also provide other qualities so that it can not only execute those functions, but also maintain the security and integrity of the data. The greater software development community agrees that there are fourteen characteristics that a database management system must have in order to be considered excellent. The following sections will delve into what each of those characteristics are.

8.1.3.1 Real World Entity

Modern database management systems are required to be realistic. In order to be realistic, they must model real-world entities in their architecture. It would make gaining information from the data more difficult if the data was not modeled in a realistic way, as it would make the data more difficult to understand and visualize. Database management systems commonly incorporate

behaviors and attributes into their overall model in order to further assist in making the database "real." [9] For example, let us say that we have a database for an entire school. This database is comprised of entities that can represent the various departments, employees, or students of the school. These entities all have attributes, such as identification numbers. Depending on what real-world object the entity is representing (such as a person), the entity can have other attributes such a birth date or salary. In a relational model, behavior is incorporated to demonstrate how the different attributes interact with one another. In our school example, a particular teacher might work for a particular department. Furthermore, students are enrolled to a particular teacher.

8.1.3.2 Self-Describing Nature

Before database management systems came traditional file management systems, and before those came filing cabinets. File management systems and filing cabinets lack a major characteristic that is required for a database management system: definition. A filing system merely contains data, but that data is usually unstructured and undefined; it lacks something fundamental to the database management system: metadata. In general, metadata is considered to be the "data about data." [10] Let us say that we have a book; the book contains records of various people who had previously lived in a local subdivision nearby. The book also contains information such as the author, date of publication, and ISBN number. In this example, the data would be the formerly mentioned information, and the metadata would be the latterly mentioned information. In the case of a database management system, the metadata would not only describe the types and formatting of the data, but also the relationships between entities within the data. The metadata is the definition of and describes how the data is to be stored and related to one another. [9]

8.1.3.3 Supporting A.C.I.D. Compliant Transactions

ACID stands for Atomicity, Consistency, Isolation, and Durability, and it relates to the integrity of the data during altering transactions such as delete, insert, and update. These properties ensure that the real purpose of the data isn't lost. [9] Let's say that we have a database for a company that contains the information on employees that work for the company and departments that are a part of the company. Employees are assigned to particular departments within the company, so an employee's identification number is saved in the departments' table. Now, let's say that a particular employee was fired; in the database, we would delete that employee from the table (in reality, it would probably be better to just mark them as fired in a separate database). In order for this delete transaction to be ACID compliant, however, we would also have to remove their employee identification number from the department they worked at, as it wouldn't make sense to list them as working for a department when they're no longer

employed with the company. The goal of ACID properties is to preserve the “real world entities” property of the database.

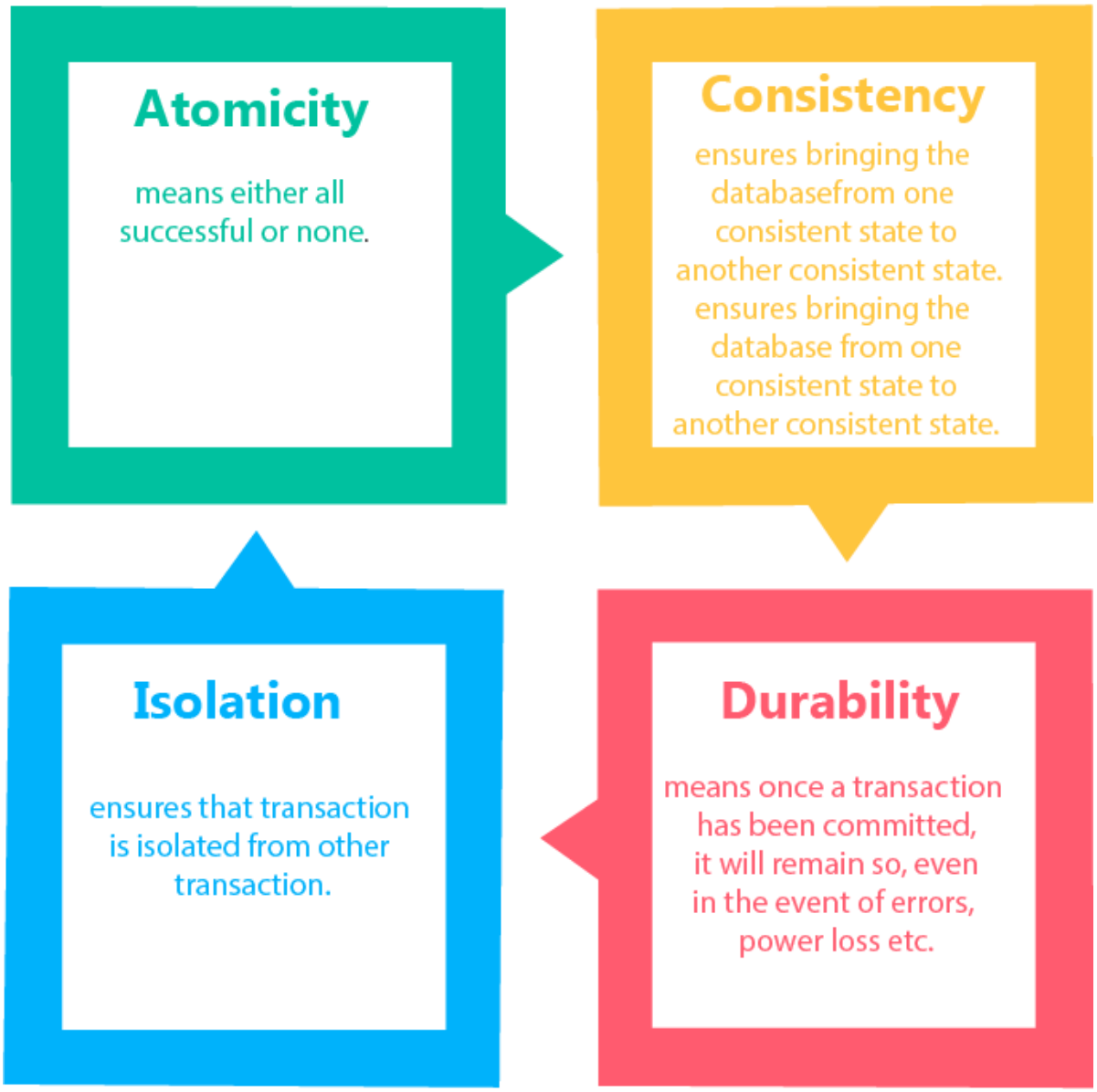


Figure 8.1.3.3.1: A friendly pictorial representation of the A.C.I.D. properties of proper database management system transactions. [11]

8.1.3.3.1 Atomicity

Atomicity refers to the qualities of being indivisible and irreducible. Something is considered atomic if it cannot be further subdivided into component parts. In the case of a

database management system, atomicity refers to transactions being processed as one such that the consistency of the database is not violated. It means that if one executes a series of transactions in one query, then the transactions are viewed as one unified transaction. This is important because it prevents the corruption of the resident data should a transaction fail in the process of executing queries. Suppose someone is performing a transaction where they first withdraw money from the bank, and then they deposit money into that same account. Atomicity ensures that the withdrawal query is successfully executed, and then it is followed by the deposit query. If either of the two queries fail, then neither of the transactions are committed to the database and both are read as a failure. [12]

8.1.3.3.2 Consistency

Consistency (otherwise known as correctness) ensures that the defined database constraints are enforced on database queries and the database itself. A database transaction may only change the target data in ways that are permissible by those constraints. This rule must guarantee that, in a sequence of transactions, the previous transaction must commit its changes before the next transaction can be allowed to make changes to the database. Should a subsequent transaction attempt to execute before the previous transaction has committed to the database, then this rule would be violated, and the data would likely be corrupted. [13]

8.1.3.3.3 Isolation

Isolation is defined by how visible the effects of a particular transaction are to other users in the database system. It can also stipulate when the effects of said transactions are accessible to the other users. Unlike the previous two, there are different levels of isolation that can be applied to a database, and both have their advantages and disadvantages. One may choose to implement a lower level of isolation if they wish to promote greater concurrent access to the database. This is important as the implementation of greater concurrency can result in savings on the use of system resources but at the cost of higher numbers of concurrency errors such as dirty reads and lost updates. At too low an isolation level, one risks violating the consistency requirements of the database, as previous transactions must complete before the next ones can begin. On the other hand, a higher isolation level can reduce or even eliminate concurrency errors, but at the cost of database blocking (not being able to access the database while a transaction is in progress, even if that transaction is unrelated) and increased use of system resources. One way to ensure isolation is to use a temporary table to perform the transactions on the relevant data, that way the original data remains safe and intact if something should go wrong. [14]

8.1.3.3.4 Durability

Durability is defined as a property which guarantees that committed transactions are permanent and immutable. Once a transaction has been processed and committed, it is permanent and will remain permanent regardless of what happens to the system. This is imperative in case of a blackout or some other event that might cause the resident database to crash. Without durability, it is entirely possible for already committed changes to roll back, which would not be ideal in any database, especially ones involving things such as money or reservations. Durability can be implemented by keeping a detailed log of all transactions including the dates and timestamps on when those transactions took place. In the event of a crash, the system can recreate all of the transactions that were lost to ensure durability. [15]

8.1.3.4 Concurrent Use of a Database

A database management system must safely support concurrent access from multiple users across multiple access points. [9] As mentioned in Section 1.3.3.3, it is imperative for a database to keep data transactions isolation as to avoid concurrency errors corrupting the data, but at the same time, failure to implement concurrency can result in database blocking which can greatly slow down the runtime of the database when multiple people are using it, consuming both time and monetary resources. It goes without saying that isolation and concurrency go hand and hand, and the database management system must be able to safely process concurrent transactions. On the user's side, one must only be allowed to view committed transactions and not ones that are in progress or are halfway executed (atomicity treats a series of related transactions as one). This is to ensure that the user is viewing accurate and non-volatile information, and this also ensures the same standard of quality for when this data is inevitably used in another transaction. One common method of handling concurrency is to store data that is being transacted upon in a temporary log file or table, that way the original data remains intact in case there's a problem. [16]

8.1.3.5 Insulation Between Data and Program

A good database management system ensures that the data along with its related metadata remains independent of the program that is managing it. Under this condition, the database management system is treated as an entirely different entity than its resident data. Originally, the structure of a datafile was defined by the application program that was using it. What this means is that any datafiles either produced or managed by that particular program could only be transacted upon by that specific program with the specific configuration that is required for those files. Now let's say that the company decided that they needed to change the overall structure of

one of those datafiles. In a modern database management system, the structure of the datafiles is stored on a catalogue that resides in the actual system, so one would only need to change the catalog, but before, one would need to change all of the programs that interacted with that datafile, wasting time and resources. [9]

8.1.3.6 Transactions

Transactions are defined as actions that are performed on data within a database to move it from one state to a new state. [9] Transactions are absolutely fundamental to the processing and handling of data, as without them, it would be impossible to manipulate anything that is stored inside a database. At its core, a database management system must support the operations of create, insert, select, update, and delete (expanded upon in subsequent sections). In addition to these fundamental operations, the transaction must also be ACID compliant (see Section 1.3.3) to ensure that the transactions perform their intended tasks without corrupting the data. Finally, transactions must be able to commit their changes to the database to reflect the updated data.

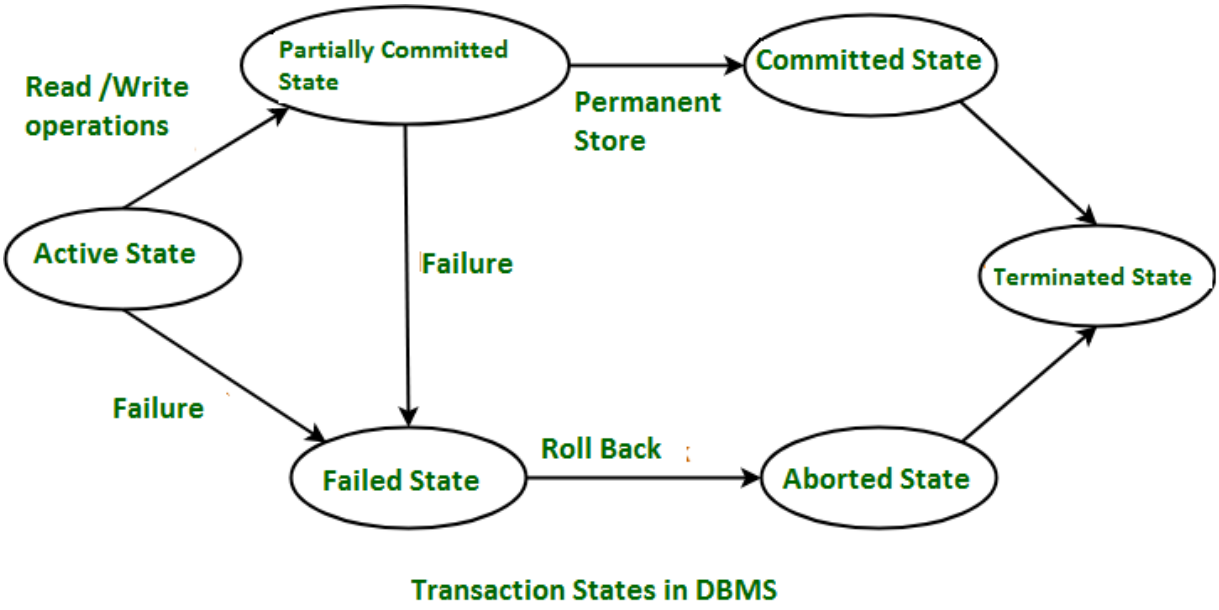


Figure 8.1.3.6.1: A diagram that succinctly summarizes the transaction process in a typical database management system. Transactions will stay in a partially committed state until they either fail or are completed. If they fail, it is the responsibility of the database management system to roll back the partial progress of the failed transaction. [17]

8.1.3.6.1 Create

The create transaction is commonly used to specify the structure of data. The create operation does not actually create any data, but rather, it creates the schema of a particular data

table. In MySQL, one uses the CREATE keyword to either create a new table (one can also specify the fields this table supports and any constraints the table may have) or to create a new database (again, constraints can be specified in the CREATE statement. [18]

8.1.3.6.2 Insert

The insert transaction inserts new data into a given table within a database. Inserting is fundamental to transactions as it is one of the CRUD operations (insert would be the C as it is used to create new records) required for any application with a user management system. Insert is commonly used by specifying the columns to populate and the tuples of data that will populate those columns, however, it can also be used to commit data into a table that has been transacted upon (transactions typically happen in a temporary table, so it is necessary to select the finished data from that table and insert it back into the database). [19]

8.1.3.6.3 Select

The select transaction selects data from a database for reading. Selection is one of the fundamental CRUD operations (equivalent to the R for reading) and is required for choosing data to process or manipulate. Without the ability to select, one cannot view what is inside a particular table within the database, much less use it. At minimum, one should be able to select specific rows from tables based on that row's identification, but generally speaking, one can select data based on a lengthy set of parameters and constraints that are specified by the user. In addition to merely reading data, MySQL allows for select statements to have conditions that allow the user to refine their selection from the database. [20]

8.1.3.6.4 Update

The update transaction allows for one to update an existing record in a table. Updating is one of the fundamental CRUD operations (the U for update) and is another required part of executing complex transactions (updating a record after an operation has been performed). Where selecting allows the user to read the database, updating allows the user to then write to the database, which is necessary for any database driven application. Additionally, one can specify conditions to restrict what columns get updated in MySQL. This allows the user to only update the data that is relevant to the transaction they are performing. [21]

8.1.3.6.5 Delete

The delete transaction allows for one to delete a record or series of records from a table. Deleting is the last of the fundamental CRUD operations (the D is for delete) and is the last required function for conducting transactions. Deleting allows a user to remove records that are no longer relevant to the current use of the database, thus enabling them to minimize wasted space on old records. It should be noted that the DELETE function in MySQL is not the same as the DROP function. The DROP function is used to change the actual structure and schema of the data. Its use is in removing columns, tables, and even the entire database. One should be cautious to differentiate between DELETE and DROP as use of the wrong term can have undesirable effects. [22, 23]

8.1.3.7 Data Persistence

Data persistence is the characteristic that data may only be removed from a database management system if the data removal is explicitly specified by the user. If the user has not given the system any input, then the data must remain in its present state for eternity. The underlying principle is that data stored in a database management system can never be lost. This is relevant for both failed transactions and system failures as the data's integrity must be preserved should any of the two things occur. In a failed transaction, it is the responsibility of the system to either rollback whatever parts of the transaction it had managed to complete or the transaction will be completed, and in the event of a system failure, it is the responsibility of the system to keep a backup of the data on a non-volatile storage medium so that it can be quickly reloaded into the system. If anything should occur, it is the responsibility of the database management system to preserve the data unless the database user should want to explicitly alter the data. [9]

8.1.3.8 Backup and Recovery

As mentioned in the previous section, a good database management system must provide a safety net to protect against data loss or corruption in the event of a system failure. While a system failure is rare, the result of a complete loss of data is so extreme that it would be foolish not to have a backup of the data, especially when the cost of protecting the data is negligible compared to the cost of losing everything. [9] In principle, a database should be backed up as often as possible to minimize the data loss in the event of a system failure, however, backing up a large database can often be costly and time-consuming. Microsoft recommends that databases be fully backed up once a week. Additionally, the database management system should perform a differential backup every twelve to twenty-four hours. A differential backup is when only data

that has recently been transacted is backed up, rather than the entire database. This is to help speed up the backup operation, as it is not feasible to backup a database with terabytes of information with the same frequency. Finally, the transaction backlog should be backed up every five to ten minutes, though the frequency can be adjusted based on how often transactions take place during that interval. It is the responsibility of the database management system to determine which backup to restore in the event of a system failure or data loss. [24, 25, 26]

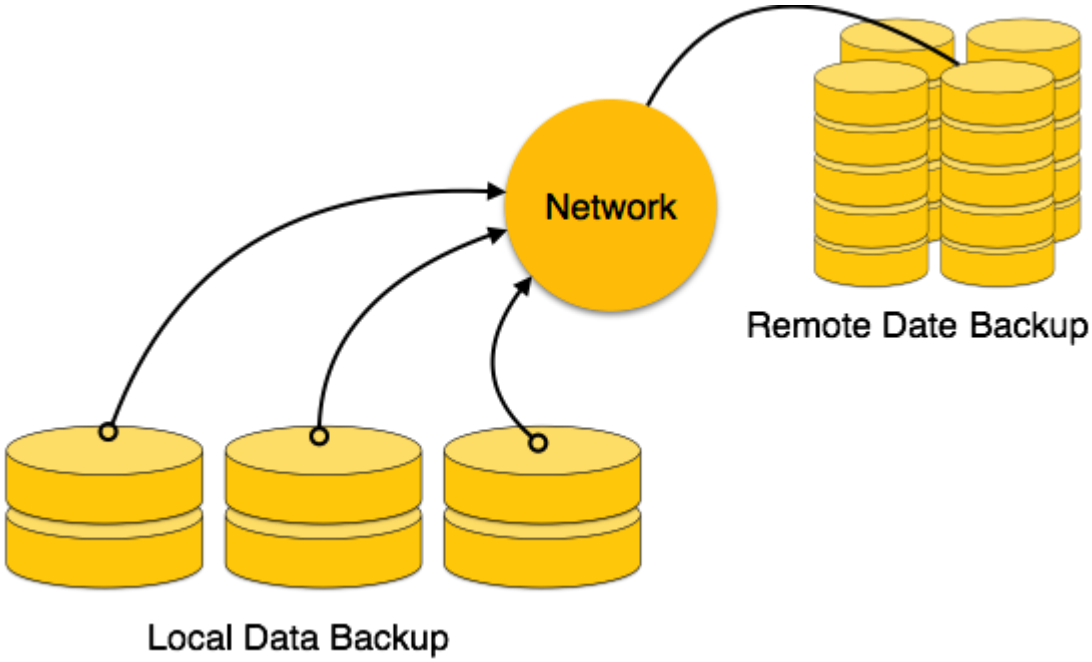


Figure 8.1.3.8.1: This is a simple diagram showing that the ideal data backup consists of both backup to a local storage system and backup to a remote storage system, in case something should occur at the place the local data backup is housed. [27]

8.1.3.9 Data Integrity

It is essential that a good database management system provides data integrity; this is considered to be one of the most important characteristics of a database management system. This quality guarantees the reliability and quality of both the data and the system the data resides in. There are a litany of ways that the integrity of a database can be compromised, such as a non-ACID compliant transaction or a security fault, and it is the responsibility of the database management system to guard against these faults. [9] Data integrity is a vast topic and can be subdivided into different categories of integrity.

8.1.3.9.1 Physical Integrity

Physical integrity is the outermost layer of integrity; it is the security of the physical database itself. This can apply to not only the physical servers the database resides on, but also the security of the applications within the database. The quality of physical integrity should guarantee that the data on a database remains accurate and consistent despite any external factors that might occur. For example, if a hurricane were to roll through the town in which the database is housed, then regardless of the effects of that hurricane, the data should be preserved (see the backup and recovery section). This also applies to the cyber security of the database; an ideal database should be impenetrable to hackers. Of course, there is no such thing as ideal security in the real world, so an actual database management system should strive to get as close as possible. This can be achieved by limiting access to the database using a philosophy of least privilege (an user should be given the smallest amount of access required to do their job). [28]

8.1.3.9.2 Entity Integrity

Entity integrity is the first type of a broader subsection of data integrity called "logical integrity." Entity integrity guarantees that each element in a database is unique. This is accomplished through use of primary keys, which are internal fields in tables within a database that are utilized to uniquely identify a record in the database. Entity integrity can be maintained by ensuring that every record is assigned a primary key and that every primary key in a table is unique. [28] In SQL, this is usually accomplished by declaring the first field of a table to be the record's identifier. A primary key must have several constraints to protect against transactions that might threaten entity integrity, such as that the key must not be null, and it must be unique.

This is accomplished, in SQL, by using the "NOT NULL" and "AUTO_INCREMENT" constraints. "NOT NULL," as the name implies, requires that the field must not be null. Inserting NULL data into a field that is marked as being "NOT NULL" will result in an error, and the transaction will be rejected. [29] "AUTO_INCREMENT" ensures that the value from the previous record will be incremented and inserted into the next record. This is particularly useful for numerical data types such as INTs, and most primary keys tend to be stored as some flavor of int. It is important to note that it is generally considered bad practice to insert values into an AUTO_INCREMENT column, as it could create problems for entity integrity should the inserted value be less than the highest value in that column. [30]

As another consideration for maintaining entity integrity, one can add the UNIQUE constraint to further ensure that the values stored in the table are unique. As far as use in primary keys, this constraint is actually redundant, as a mixture of good querying practices and the "AUTO_INCREMENT" keyword will guarantee that this column is always unique. As such, the use of the "UNIQUE" constraint will not be used for a simple integer primary key. Finally, one

must declare the primary key using the “PRIMARY KEY (name)” statement in the table declaration with “name” as the identification field in the table, generally named “id.” [31]

8.1.3.9.3 Referential Integrity

Referential integrity refers to the use of a series of procedures and protocols designed to guarantee the proper and consistent storage and usage of data. Referential integrity must guarantee that the rules and constraints implanted within the structure of the database are obeyed by all necessary alterations, additions, and removals. Referential integrity also guarantees that the aforementioned operations only occur if those operations are required. Some of these rules require procedures for removing or prohibiting duplicate data, ensuring that the data is precise and accurate, and prohibit the insertion of data that is considered unsuitable by those rules. Additionally, referential integrity must guarantee that the relationships between entities are respected through use of foreign key constraints. This is especially applicable when deleting data from a database, as without the proper constraints, one can end up with a situation where a record’s foreign key points to another record that no longer exists, which violates several of the required characteristics of a database management system. To prevent this situation from occurring, one can choose from several integrity philosophies. [28]

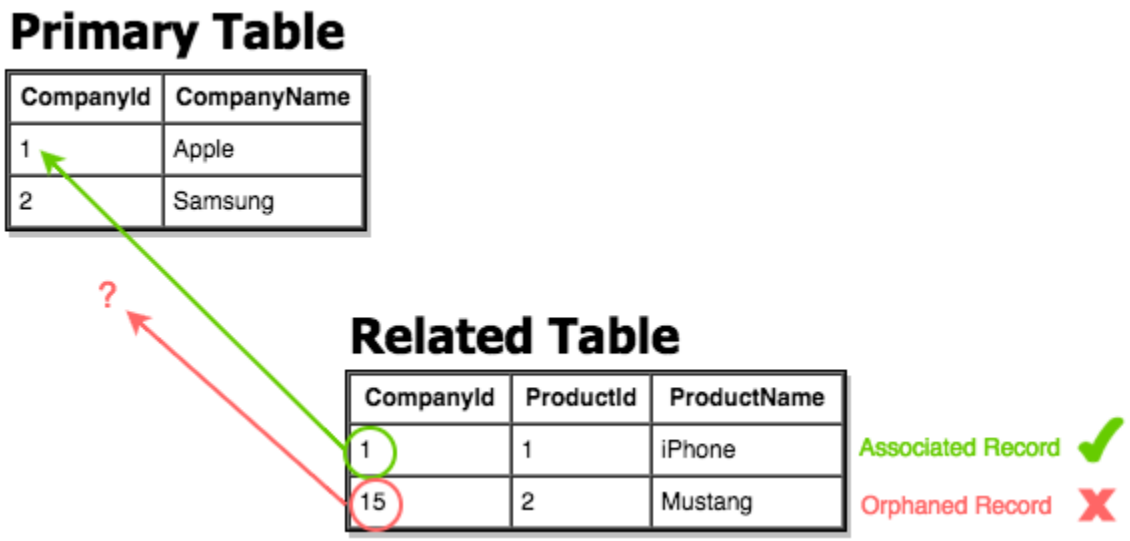


Figure 8.1.3.9.3.1: This is a figure demonstrating what can happen if nothing is done when a primary key record is deleted. As is shown, the deleted primary key record leaves the related table with an orphaned record, which is simply unacceptable. [32]

8.1.3.9.3.1 SET NULL

The “SET NULL” philosophy ensures that the deletion of a record from the primary key tables affects rows in the foreign key table, then those rows must be set to NULL. The same goes for updating values in the primary key table, as if the updating of a value in the primary key table results in changes to the rows in the foreign key tables, then those rows are set to NULL. In order to prevent a conflict of constraints, however, it is necessary to ensure that the table columns do not possess the constraint of “NOT NULL” as attempting to implement this philosophy with that constraint would result in an error. [33]

8.1.3.9.3.2 CASCADE

The “CASCADE” philosophy ensures that if a record from the primary key tables is deleted, then any rows in the foreign key table that are impacted by this deletion must also be deleted. The same can be said for updating values in the primary key table, as the row values in the foreign key table that are affected by the updating operations must be updated with the resultant values from the primary key table. [33]

8.1.3.9.3.3 SET DEFAULT

The “SET DEFAULT” philosophy ensures that if the deletion or updating of the primary key table affects any rows in the foreign key table, then those rows are set to their default values. This is another operation, like “SET NULL” whose values are independent of the primary key table, as regardless of what the primary key table’s rows were, the connected rows in the foreign key table are set to a particular value. In this case, rows in the foreign key table require use of the “DEFAULT” constraint. The “DEFAULT” constraint allows a database designer to specify a default value for a particular column in a table. This is especially useful in situations where one might not always get data on insert statements for that particular column, but they still wish to utilize the “NOT NULL” constraint. [33, 34]

8.1.3.9.3.4 NO ACTION

The “NO ACTION” philosophy completely prohibits the altering of foreign key table rows from updates to or deletions of the primary key table. This is also SQL’s default action, and it specifies that if an action on a foreign key table results in changes to rows in the primary key table, then the transaction is denied, rolled back, and an error message is displayed. [33]

8.1.3.9.4 Domain Integrity

Domain integrity, another component of the broader concept of logical integrity, relates less so to the tables and their relationships to one another and more so to the actual values and data types that are stored within these tables. Overall, it refers to the constraints and rules that govern the values housed within the tables. The domain integrity can be defined by several constraints that are specified in the creation query of the table. [35]

8.1.3.9.4.1 The Data Type and The Length

In a database, it is important to specify, on table creation, what type of values are allowable in a particular field of the table. MySQL offers several simple and complex data types that can be utilized to suit the needs of any developer. Among those are numeric, date and time, and string data types. These types are the most commonly utilized as they are all that is necessary to represent most forms of information, however, MySQL does offer other types such as spatial and JSON for more specific use cases. Data types specify the number of bytes to allocate to each field in the table when a new instance of the table is created, and it also gives a container to put each element of data when new records are inserted into the table. It is worth noting that the aforementioned data types are ones provided by MySQL, and other database engines may provide different implementations or additional complex types. Additionally, MySQL allows the engineer to specify the number of characters to be stored in that particular value. It should be worth nothing that the length of the data type is not equivalent to the number of bits in that type. For example, a data type of INT(8) would be an integer with 8 digits, allowing for a maximum value of 99999999. Generally speaking, length should be long enough to accommodate all entries that could possibly be inserted into the database, but not too long as to waste memory upon the insertion or creation of new records. [36]

8.1.3.9.4.2 NULL Value Acceptance

As mentioned several times in previous sections, one of the most important constraints is the NOT NULL constraint. NOT NULL ensures that the value of a column in a particular row is not null, and if null data should attempt to be inserted, an error is generated and the insertion is blocked. This does not necessarily mean that null values are never permitted in a table or that null values in a data table are particularly bad, although it is common practice to require that elements in a table are not null. There are use cases, however, in which null values can be useful in extracting and analyzing data stored within a database. For example, null values could be used to indicate that records in a table lack a certain feature or property, and although the BIT data type (or some kind of binary type) would be more appropriate, it is still a viable option. Null

values can also be useful in situations where a foreign key table might now always have a primary key table associated with it. In this situation, null values can be utilized to demonstrate that a parent table has no children associated with it.

8.1.3.9.4.3 Allowable Values

The concept of allowable values feeds into a broader category of the use of domain constraints. SQL offers several constraints that can be used to disallow values that are unrealistic or prohibited from a particular record. Additionally, SQL offers constraints that can associate particular values with keys or indices. Common constraints are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT, and CREATE INDEX. NOT NULL guarantees that the value is not null. UNIQUE guarantees that the value is unique in the table. PRIMARY KEY and FOREIGN KEY are used to establish a relationship between tables. CHECK ensures that inserted values satisfy some conditional statement that is specified after the keyword. DEFAULT is used to specify default values. CREATE INDEX is used to create and rapidly retrieve data stored within the database. In combination, these constraints give a database programmer great control over their database.

8.1.3.9.4.4 Default Values

Default values are the last of the components of domain integrity; they are another important feature of tables in MySQL, and they can serve as an alternative to NOT NULL should someone wish to not utilize that constraint. Default values allow the database engineer to specify a value that should be placed in the table in case that value is not supplied by a particular record during an insertion statement. In MySQL, one can invoke a default value by simply using the DEFAULT keyword in the constraints list following the declaration of the name of the feature and the datatype and length of the feature. Additionally, one can also use default values to enforce referential integrity, as mentioned in several sections above. [34]

8.1.3.9.5 User-Defined Data Integrity

Finally, user-defined data integrity is the last component of the broader category of logical integrity. In spite of all that entity, referential, and domain integrity offer, the implementation of those three alone is not enough to satisfy the logical integrity requirements of the database. It is important to remember that databases will be utilized by many customers, all with their own unique use cases, and as such, believing in some kind of one size fits all mentality would not be satisfactory. Thus, user-defined data integrity is a philosophy that further refines

the integrity constraints to the needs of the particular database engineer. This form of data integrity must be defined by the institution that runs the database, and the definition of this integrity can vary greatly between different customers. [28]

8.1.3.10 Multiple Views

In a database management system, it is important for the system to allow different users to have different views of the database and its resident tables. While this can be seen as a mere issue of convenience and ease of viewing, as limiting the view to only relevant tables can make extracting information from and analyzing the data within the database significantly easier and computationally faster, it is also an issue of convenience, as a particular user should only be allowed to view the data that is relevant to their use case, and not other data that might be sensitive and normally inaccessible to that user. The use of a view in a database should follow the philosophy of least privilege, only providing the user with just enough for them to complete their tasks. Depicted below are figures that show how the views can be utilized to slice portions of a database. [9, 37]

Figure 8.1.3.10.1: This is a diagram of the use of a view to show the information in this particular database pertaining to the month of January. As is demonstrated here, the user can only see the information pertaining to January.

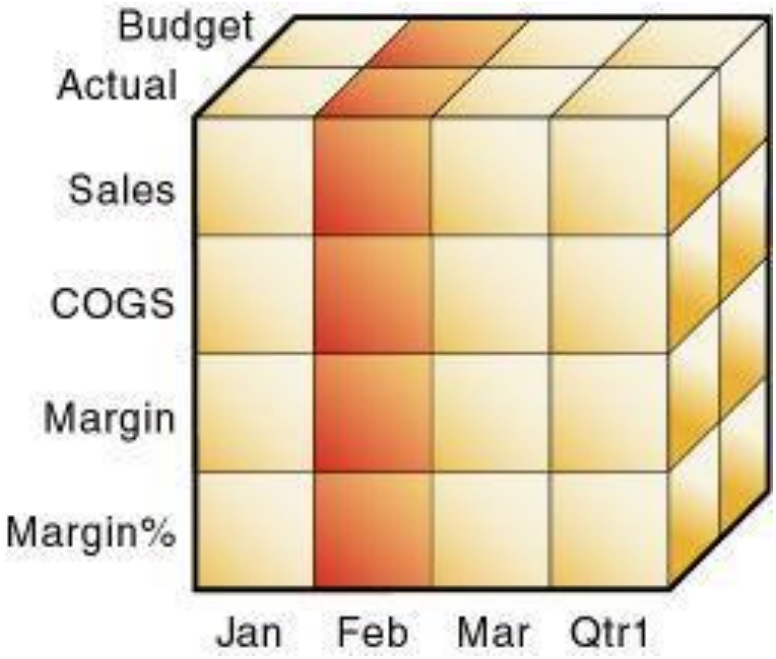


Figure 8.1.3.10.2: This is a diagram of the use of a view to show the information in this particular database pertaining to the month of February. As is demonstrated here, views can be used to show specific subsets of the database, and in this case, the user can only see the information pertaining to February and not to January.

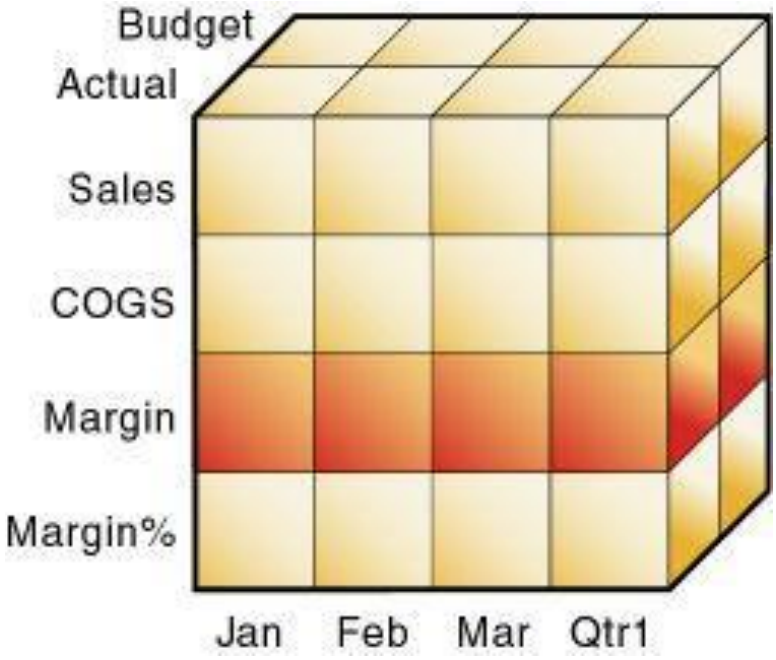


Figure 8.1.3.10.3: This is a diagram of the use of a view to show the information in this particular database pertaining to the field of Margin over the course of a quarter. As is demonstrated here, this time the view is sliced differently, revealing only one feature of the database over the entire length of time these measurements were collected.

As mentioned before, database management systems allow users to create specific views of the database. A view is represented as a virtual table created by using SELECT statements. Generally speaking, these SELECT statements are accompanied by conditionals that only filter the relevant data. One may create a view in MySQL by using the CREATE VIEW query. They may specify the rows of the view as though they are specifying the rows of a table, but again, the rows are specified using SELECT statements rather than by specifying the data types and their constraints. Views also allow users to concatenate multiple columns or adjust the representation of a column to be more readable or usable to the user, thus allowing for greater flexibility than simply working with raw tables. [38]

8.1.3.11 Stores Any Kind of Data

In relation to the first attribute, that a database management system should be realistic and that it should represent real world entities, it makes sense that a database management system should also be able to store any kind of data to represent any type of entity or attribute that an entity could possess. For example, suppose a company maintains a record of all of its employees in a database. Naturally, the company would have records that contain the employee's name, salary, and address. While these attributes can be used to describe an employee, if the database management system only allowed those three types of fields, then it would be a poor database management system. In practice, a database management system should be able to accommodate any field that a database engineer may wish to include in a particular table. There are many ways to describe a real world entity, and the database management system should be able to accommodate each and every one of those possible descriptions. [9]

8.1.3.12 Security

As mentioned a multitude of times in previous sections, security is a high priority in a database management system. Specifically, security pertains to the level of access a particular user has to the information and records stored within. While nearly half of all data breaches consist of human error, it is up to the database management system to mitigate technical vulnerabilities. These vulnerabilities can be exploited through injection attacks, buffer overflows, denial of service attacks. It is also possible for malware to infect the database management system, and it is even possible for an attacker to prey on the backups of the data if they are not properly secured. In order to mitigate these threats, it is essential that the database management system provides safeguards in case an attack manages to make it to the system. [9, 39]

In any database backed application, the database management system is the last line of defense. The system should encrypt all data and only allow users to view the minimal amount of

data required for them to complete their tasks in accordance with a least-privilege philosophy. The former must be accomplished by the system, but the latter can be accomplished by the database engineer through use of views. Any and all software accompanying the database management system should be secure and bug-free, and the database backups should be treated with equal levels of care as the actual database. Finally, a good database management system should offer auditing functionality which can monitor all user accesses and transactions as well as their results (whether or not the transaction failed and succeeded and the result of the transaction). [39]

8.1.3.13 Represents Complex Relationship Between Data

As covered in prior sections, a database should be able to fully and completely represent real world entities. Naturally, we can group entities together to form a system of entities. As in any system, entities are related, and so it is only natural to require that a database management system should be able to represent any amount of arbitrarily complex relationships. In many cases, one can abstract the relationships in a system down to simple key pairs. For example, two tables in a database can be related through use of a primary key and foreign key relationship. In MySQL, this can be accomplished through usage of the primary and foreign key constraints. However, there are use cases in which a simple relationship like that may not be adequate to fully describe how two entities are related. Moreover, there are many cases where a singular entity can relate to more than one entity or where many entities can relate to many other entities. It is required that the database management system provides the database engineer with a method to represent these complex relationships. [9]

8.1.3.14 Query Language

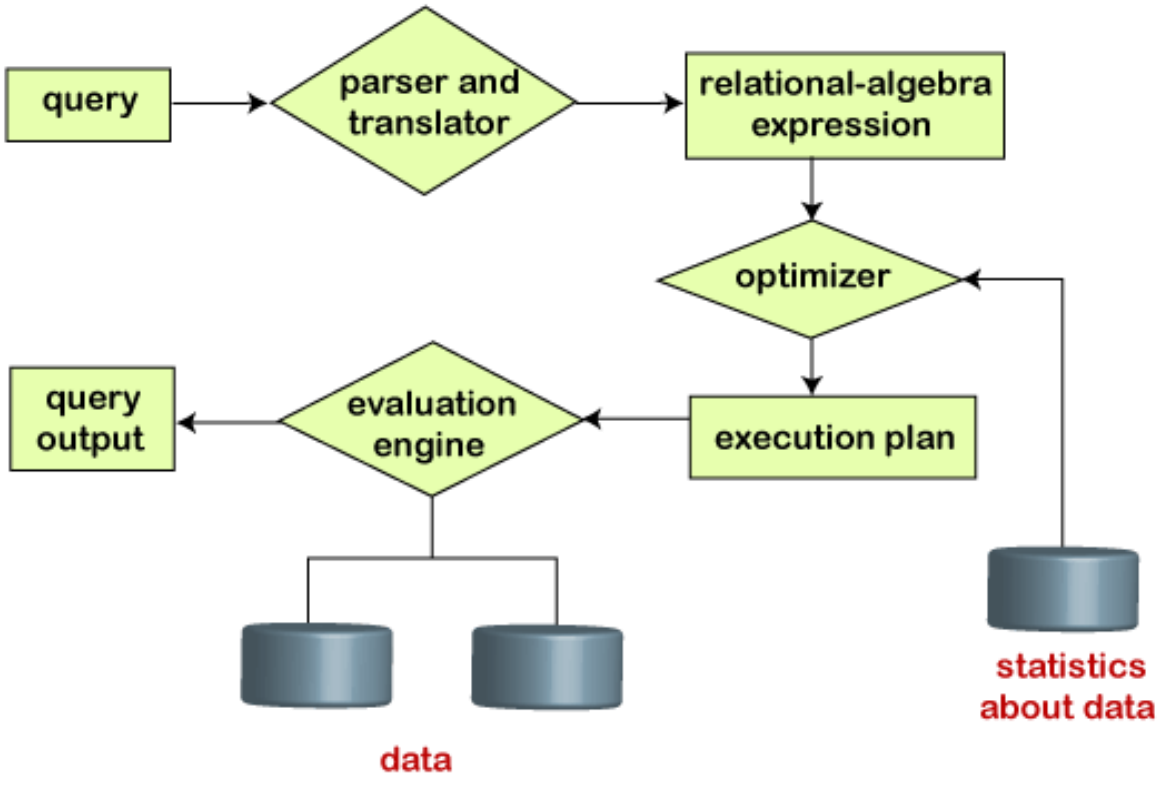
Perhaps the most important characteristic of a database management system, a database management system must offer a query language through which one can interact with the database. Without a query language, one cannot interact with the database management system in any regard whatsoever. In fact, it isn't even possible to let the server know to create a database without the use of a query language. Query languages allow users to retrieve and manipulate the data stored in a database management system. While it may seem sufficient to perform these functions at bare minimum, a good database management system should have a powerful and robust query language that allows for more effective and efficient use of the data within. [9]

In a database management system's query processing engine, queries go through a series of steps so that the high-level, human-readable language is converted into a format that the database management system can interpret and understand. When a user decides to compile and execute a query, that query enters the first step of execution, called "parsing and translation." In

this stage, the query is evaluated from its keywords to its logical conditions and table columns. The query is translated into something called a relational-algebra expression. Relational algebra is the equivalent of assembly code in a computer processing unit. This is the language of the database management system. [40]

Once the queries have been translated, they are passed through an optimizer which reduces the relational-algebra expressions down to their most optimal forms. This process is necessary to reduce any redundancies in the queries and to ensure that the most efficient instructions are actually being executed on the database. The optimizer can incorporate known metadata such as the size of the table, frequency of a certain observation, and other statistics that can help further optimize the expressions. [40]

Once optimized, the queries go through the evaluation stage, where the expressions are formatted into an execution plan, and then they are evaluated on the resident data. At this point, one of two things can happen. The query can either have an error, resulting in an output indicating that the query could not be completed, or the query can successfully execute, modifying the data and then committing those changes, outputting the changes to the user. Figure 8.1.3.14.1 details this process.



Steps in query processing

Figure 8.1.3.14.1: This is a diagram of a typical query processor. Queries are first parsed and translated to relational-algebra expressions. The expressions are then optimized and an execution plan is formulated. The expressions are then evaluated on the data, and the result of the query is output to the user. [40]

8.1.3.15 Miscellaneous Properties

While the aforementioned characteristics are required of a good database management system, there are several other characteristics that can further enhance the performance or efficiency of the system. The qualities are not necessarily required of a database management system, but many commercial systems possess them, and in cases where the size of the data stored may be enormously large, they are necessary to facilitate ease of access and modification to the stored data.

8.1.3.15.1 Sharding

In a database management system, one of the most important factors to consider is storage. Since a database is basically a managed storage system, it is important to ensure that the storage doesn't run out and also that the data within is still accessible in a timely manner, regardless of how much data might be housed within. To ensure this, many database management systems implement something called "sharding." Sharding is where a single logical table is split up amongst different areas of the database. A shard is a disjoint subset of a particular table, and the union of all these shards would equal the table itself. Different shards can be stored on different disks in the same server or even on different servers across a distributed network.What matters is that the main server keeps a record of references to each of these shards to avoid a loss of data. Sharding can be broken down into two categories: horizontal and vertical; support for each varies across different database management systems. [41]

Horizontal sharding is where a table is split across its horizontal axis. This is where all of the columns are kept together, but not all of the rows will occur in a single shard. In order to access all of the rows, one must go through each reference to each of the shards. Vertical sharding is where tables are split along the vertical axis, namely, the columns. In this situation, all of the rows are kept together, but the columns are disjoint and referenced by pointers. In a relational database, records can use primary and foreign key pairs to reference particular column shards. [41]

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Vertical Shards

VS1

CUSTOMER ID	FIRST NAME	LAST NAME
1	Alice	Anderson
2	Bob	Best
3	Carrie	Conway
4	David	Doe

VS2

CUSTOMER ID	CITY
1	Austin
2	Boston
3	Chicago
4	Denver

Horizontal Shards

HS1

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston

HS2

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Figure 8.1.3.15.1.1: This is a diagram of vertical versus horizontal shards. As can be seen here, vertical shards are split along the vertical axis of the table, splitting columns apart and keeping the primary key as a link between columns. Horizontal shards are split by row, maintaining all of the columns but partitioning the full records into different shards. [42]

8.1.3.15.2 Scalability

As mentioned in the previous section, a database management system needs to be able to handle an arbitrary quantity of data. Whereas sharding handles the logical partitioning of data, scalability deals with the actual physical machines. There are numerous cases where the size of the data actually exceeds the capacity of the machine or machines it is housed on, thus, it is necessary to either extend the machine it is on, or add additional machines to the cluster of machines on which the database is housed. Like sharding, scalability can be categorized into horizontal and vertical scalability. [43]

Horizontal scalability pertains to adding additional machines to the server. Rather than upgrading the current database server, one simply adds an additional server to the cluster, thus increasing the overall capacity of the database. Horizontal scalability is limited by the amount of room available for the machines, and even then, this can be remedied by using a distributed database. On the other hand, vertical scalability pertains to upgrading the machine the database is currently on. At this point, there is little to no concern as to the physical space the machine will

take up, but now there is a limit on how fast the latest upgrades for the server are. If the upgrades are not adequate for the database, then it may not be feasible to scale vertically. [43]

For context, SQL databases tend to be vertically scalable, whereas NoSQL databases tend to be horizontally scalable, and it should be up to the database engineer to factor that in when choosing a particular database management system.

8.1.3.15.3 Indexing

One of the key features of a database management system is its ability to access data. Accessing data, however, is not enough on its own; a good database management system should be able to access its data quickly and efficiently, especially when there is a large quantity of data housed on the server. In most computer programs, one can store data in an array where the key is a positive integer, and the value is whatever resides at that particular location in memory. This arrangement is not satisfactory for a database (as not all keys are integers or start at 0), and thus, all database management systems have various methods for quickly indexing and accessing data stored within. These methods consist of primary index, dense index, sparse index, and clustering index. [44]

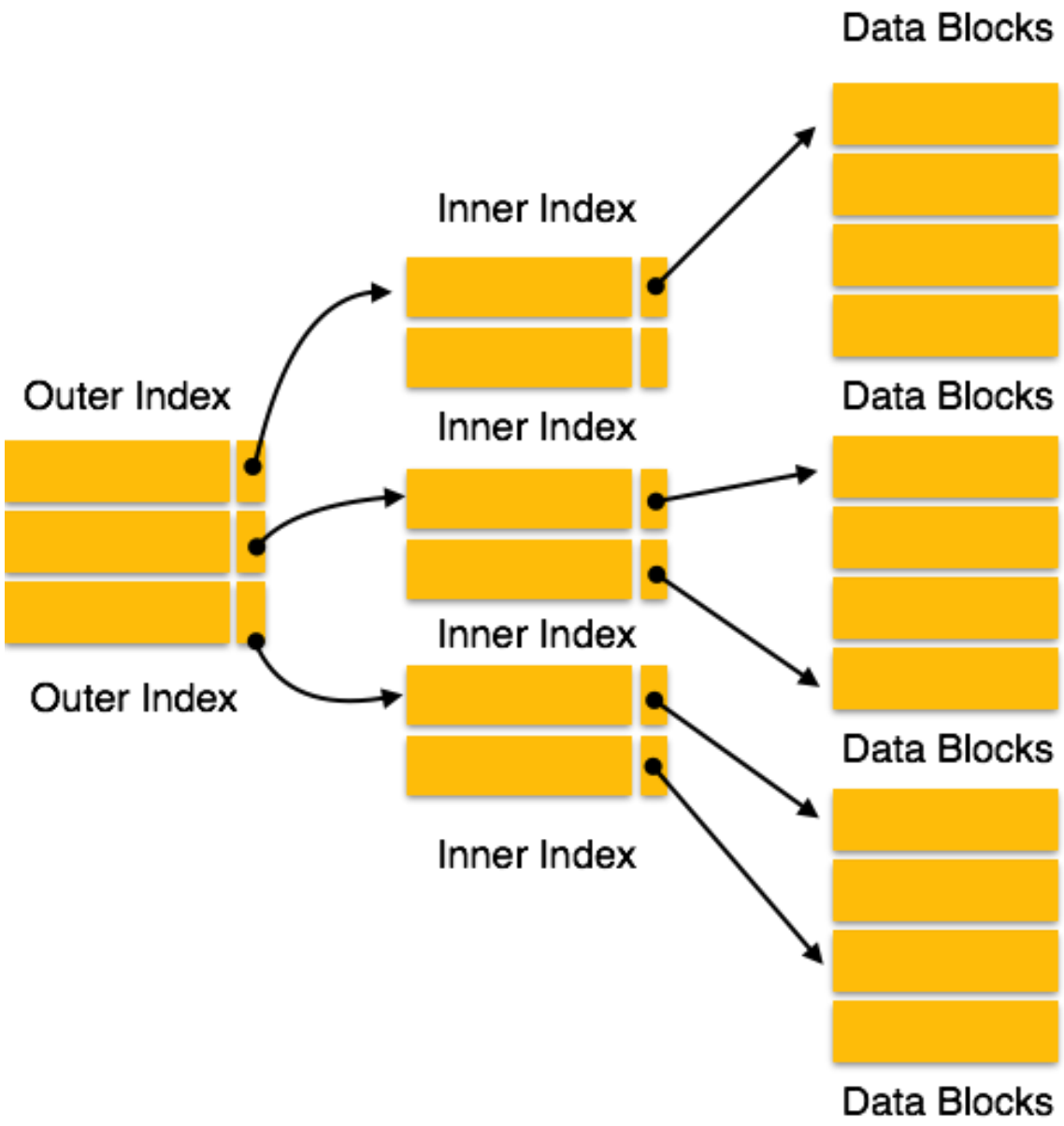


Figure 8.1.3.15.3.1: This is a diagram of an example of indexing. Indexes are used to search for specific data within data blocks. As can be seen here, the indexes for a tree, leading to a close to logarithmic runtime complexity when it comes to searching and accessing operations. [45]

Primary index is very similar to the method of accessing a value in an array. It is where the index of the data is determined based on the data's primary key information. Without any additional work, this method is fairly good at accessing data if the data is searched for based on index and the indices are sorted. In fact, most database management systems that use this flavor of indexing can get a logarithmic runtime complexity when it comes to searching for data based on this index. This means that if there are a billion elements in the database, then searching for a particular element based on its index will only take roughly thirty steps of work. [44]

Dense and sparse index are subcategories of primary indexing, and they are a prime example of runtime and space complexity tradeoff. Dense index contains an index record for every search key in the data file, greatly increasing the efficiency of search operations. Specifically, the number of entries in the index table is equal to the number of records in the main table, and while this decreases searching times, it increases the amount of memory required to store data, as the index of the data essentially needs to be stored twice. In contrast, sparse index is where only a few items have index records, and each index record points to a block in the table. This greatly reduces the size of the index table, but it can slightly increase the time it takes to search for data as it is reasonable to assume that the block would have to be searched either linearly or logarithmically. [44]

Finally, clustering index, something that MySQL uses, is a flavor of indexing where records with similar characteristics are grouped together, and indexes are created for those groups. Unlike primary indexing, clustering indexing can use non-primary key columns as part of the index, enabling search in the database based on features that are not the primary key. Groups of columns can be placed together to form an index, and this enables a close to logarithmic search and accessing of data. [44]

8.1.4 Database Architecture Types

Though there are many competitors in the database management system scene, there are only two major categories of architecture types that a particular database management system can fall into. These categories are relational and non-relational databases. Each of these two types has unique characteristics that make it ideal for some use-cases and detrimental for others.

8.1.4.1 Relational Databases

A relational database is exactly what it says; it is a database that stores and accesses data points that are related to one another. Relational databases are based on an entity-relationship model, and they work well in those particular use-cases. In most relational databases, each row of a table has a key that can be used to access that particular row from other tables in the database. Columns in a table can hold the attributes of a particular entity, and each record should have a value for each attribute. Relational databases are built around the idea that entities are related to one another and those relationships can be meaningfully modeled in the database. All SQL databases are considered relational databases. [46]

8.1.4.2 Non-Relational Databases

A non-relational database does away with the entity relationship model in favor of less tabular methods of storing data. Non-relational, or NoSQL, databases store their data through use of data structures and documents rather than through use of tables and relationships among those tables. While it is still possible to maintain relationships between particular elements in a NoSQL database, it isn't the ideal use-case. NoSQL databases can be particularly powerful in their respective use-cases (again, document styled data, JSON, etc.), especially since search queries do not have to search through index tables to locate a particular data entry. NoSQL databases are also great at horizontal scaling and can be a great choice for handling large volumes of data, but ultimately, it is up to the database engineer to determine if a NoSQL database is the ideal choice for their use-case. [47]

8.2 Backend

8.2.1 Backend overview

Creating an application requires a lot of components to it and can be at times a very extensive amount of work. Some applications require one person to complete it while some applications require many individuals to complete the task. Today we will be focusing on the back-end side of projects, often referred to as the server side. This allows an application to be able to send and receive from the client to the server side. The back-end component consists of the servers, different applications and databases which are used to retrieve the request of a client. Another crucial part is Application programming interface (API), which permits any application to communicate with one another and this could be among applications being built or to a pre-existing application.

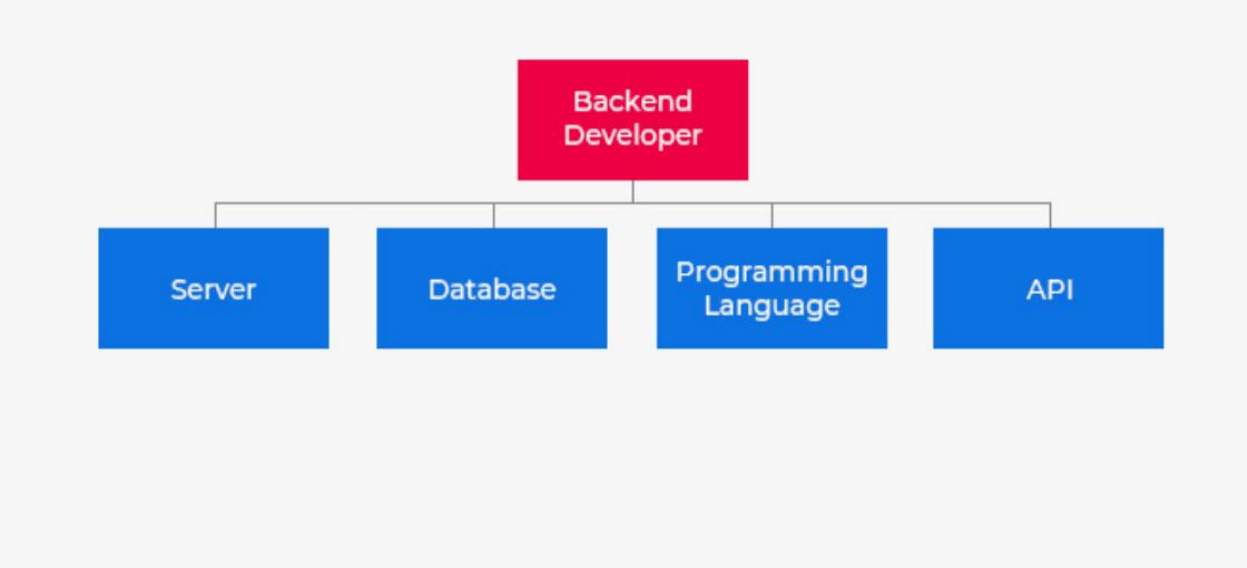


Figure 8.2.1.1: Flow diagrams that describe essentially the different skills required of a backend developer. [48]

8.2.2 Server Management

A server can be viewed as the kitchen of a restaurant, where clients come in, put in an order for their meal and then send it to the kitchen to be fulfilled. A server is software and at times hardware that is used as a pinpoint to use Hypertext Transfer Protocol (HTTP) to fulfill the response of a client made over the Web. [49] Without a server, communication on the Web would be a difficult task to accomplish, as servers are used to send and receive emails and build web pages. Furthermore, a server can be used to display different contents, such as static or dynamic. A static web server is one that sends its files that it hosts as is, meaning that no modification can be made to it but a dynamic server which is often used more and is one that can be updated or modified as clients visit the site.

Luckily our sponsor was nice enough to provide us with a ready to use server, which is the Apache HTTP Server, so we will not have to worry about setting one on our own. Created in the mid 1990's, Apache is a popular server used among developers and companies such as eBay, LinkedIn and Facebook. This is a server that is open-source and permits anyone to be able to use it for free, is maintained and upgraded by everyone over the world and is able to be run on any operating system. For our use we will be using the Linux version and at the same time use the MySQL database that comes with it and would be a few of the components that make up the LAMP stack. Some of the drawbacks of using the Apache server is the ability to modify the configuration that is offered, the performance of websites that have high traffic and getting rid of preexisting unwanted services and modules.

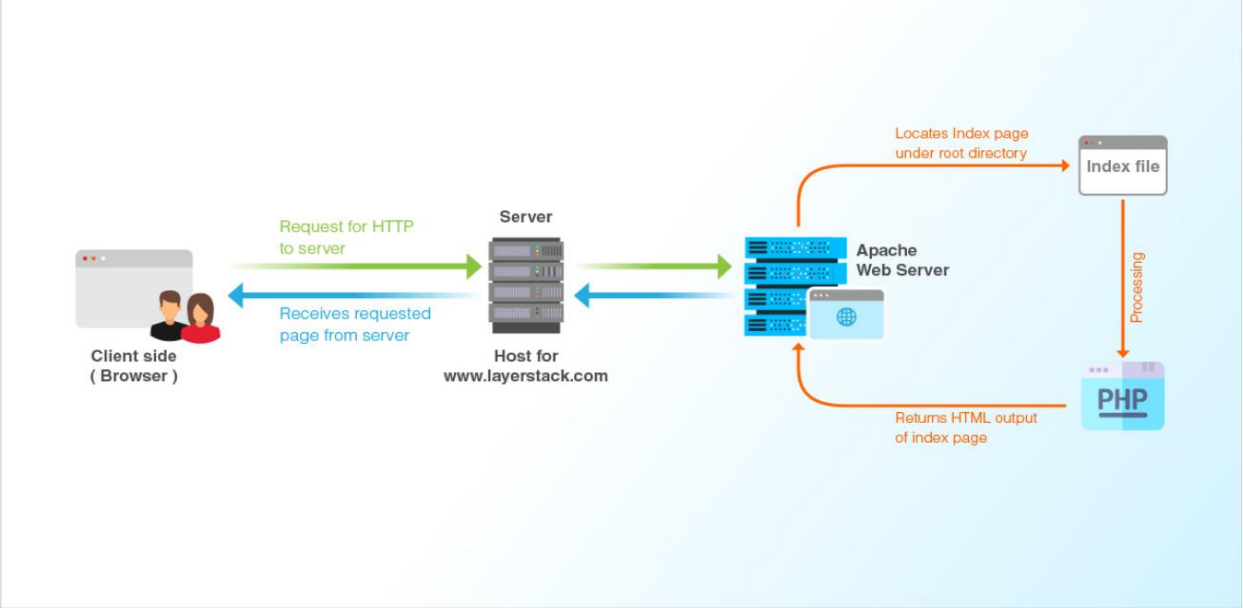


Figure 8.2.1.2: Depiction of how a server takes the request from the client side and returns it to the backend, while showing how all the needed files interact with each to return the needed request.[50]

8.2.3 Flask

As stated, before in section 4.2, the web server could be thought of as a kitchen in a restaurant that fulfills the orders of the clients that come. Then how does the order manage to make its way to the staff in the kitchen? If you answered by saying a waiter, then you are hundred percent correct and that is where the Flask framework from Python comes into play. A micro web application framework that is written in Python, which is often used to make web applications and HTTP request management. Thus, launched only in 2010, this framework has become popular and is used by many major companies such as LinkedIn, Uber and Dropbox. [51] Flask is run on engines such as Werkzeug that are used to implement requests, responses and many more other functions. Also, WSGI is used as a web server gateway to get your application to be able to run with the web. [51]

So how exactly is flask used to be viewed as a waiter in a restaurant? Well, we will use it to create an application programming interface (API) to communicate with our web server, database, and client requests. We must then use the representational state transfer architecture style that promotes performance, scalability, simplicity, and reliability in the giving system. [52] Following this architecture style you are faced with some constraints, that consist of the server not having any states, having cacheable data that is retrieved from the server and the client and having a uniform interface that allows for the server to access resources without defining their representation

8.2.4 Flask Implementation

Step 1:

Windows users must ensure that Python is installed on the laptop, either Python 2 or Python 3, must visit https://www.python.org/downloads/ in order to download the version you prefer. Keep in mind that the virtualenv module does not come with Python 2. As for users who are using MacOS and Linus those operating systems come with Python 2 preinstalled and must download Python 3 if they would like to use that version instead.

```
Jeffreys-Air:~ jeffreycherisma$ python --version
Python 2.7.10
```

Figure 8.2.4.1: Checking the version of Python that you are using.

Step 2:

Must create your environment and application in this step, by creating a directory of your project and making sure that you include your virtual environment to use for Flask.

Windows:

py -version # -m venv <name of the given environment>

MacOS and Linux

Python3 -m venv <name of the given environment>

```
Jeffreys-Air:Myapp2 jeffreycherisma$ python3 -m venv venv
Jeffreys-Air:Myapp2 jeffreycherisma$ ls
__pycache__     app.py          venv
```

Figure 8.2.4.2: Creating your controlled environment..

Step 3:

It is believed to work in a virtual environment when working with the Flask framework, this will allow you to be able to create multiple applications and the changes you make in one

environment will not affect the changes made in another and therefore you must activate it before you begin your work.

Windows:

<name of the given environment>\Scripts\activate

MacOS and Linux:

<name of the given environment>/bin/activate

```
Jeffreys-Air:Myapp2 jeffreycherisma$ source venv/bin/activate
(venv) Jeffreys-Air:Myapp2 jeffreycherisma$
```

Figure 8.2.4.3: Activating your controlled environment.

Step 4:

The last step is to install Flask using pip.

```
(venv) Jeffreys-Air:Myapp2 jeffreycherisma$ pip install Flask
Requirement already satisfied: Flask in ./venv/lib/python3.9/site-packages (2.0.
2)
Requirement already satisfied: itsdangerous>=2.0 in ./venv/lib/python3.9/site-pa
ckages (from Flask) (2.0.1)
Requirement already satisfied: click>=7.1.2 in ./venv/lib/python3.9/site-package
s (from Flask) (8.0.3)
Requirement already satisfied: Jinja2>=3.0 in ./venv/lib/python3.9/site-packages
 (from Flask) (3.0.3)
Requirement already satisfied: Werkzeug>=2.0 in ./venv/lib/python3.9/site-packag
es (from Flask) (2.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in ./venv/lib/python3.9/site-pack
ages (from Jinja2>=3.0->Flask) (2.0.1)
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the '/Users/jeffreycherisma/Desktop/Senior_des
ign2/Myapp2/venv/bin/python3.9 -m pip install --upgrade pip' command.
```

Figure 8.2.4.4: Install Flask.

8.2.5 HTTP Methods

REST APIs like stated before allows for you to communicate among different applications through the world wide Web. Also, another way that major companies allow for external developers to gain access to their data. A major use of this in this day is websites forgoing registrations for users and allowing the individuals to use a preexisting Google or Facebook account. In order to accomplish all this, REST APIs have implemented a set of HTTP methods that allow for APIs to know what sort of information or data that is being requested of them.

8.2.5.1 GET

This is the request that allows you to retrieve data from a server, whether it be your own personal website or using a URL to access another application's server. Does not do any modification of any sort and often time access must be given to accomplish this request. Must also remember that for sensitive data that you do not use this type of request.

8.2.5.2 POST

Using this request allows applications to populate their servers and database, as it creates a new resource. Without a Secure Socket Layer (SSL), this request is insecure just like the GET request and this is insufficient as sensitive data is often used with the POST request. Also, it is considered to be non-idempotent as identical requests can be made, therefore must be managed properly.

8.2.5.3 PUT

The PUT request is one of many requests that allows you to make modifications and is used to update existing resources. Can also be used as a POST request to create new resources to a server. However, this request is idempotent and therefore will only allow you to create the one resource.

8.2.5.4 DELETE

By far the most upfront request there is as the name signifies what it does. This request is called to delete a resource in the database and once such a task is completed it cannot be undone. Another request that is also considered to be idempotent as it will only delete that one resource that has been stated to be deleted.

HTTP Method	Request has body	Response has body	Safe	Idempotent	Cachable
GET	NO	YES	YES	YES	YES
POST	YES	YES	NO	NO	YES
PUT	YES	YES	NO	YES	NO
DELETE	YES	YES	NO	YES	NO
TRACE	NO	YES	YES	YES	NO
OPTIONS	NO	YES	YES	YES	NO
CONNECT	NO	YES	NO	NO	NO
PATCH	YES	YES	NO	NO	NO

Figure 8.2.1.3: HTTP Methods and their components

8.2.6 HTTP Status Codes

Since we must communicate between the server and the browser, there must be a way to know what the status of such requests is. This is where HTTP status codes come in handy, these codes consist of three-digit which to normal users might look unfamiliar. However, to programmers especially web and app developers these are codes they make themselves aware of. These status codes, also known as response codes, are categorized into five categories that have their own significance.

These three-digit codes will always begin with a one in the front, such as 1xx. The first category consists of four codes only and their sole job is to provide information back from the browser, an example is “100” which stands for “Continue”. The next category 2xx results in a successful status, which signifies that the browser successfully received your request and often the status as a developer that you would like to see. We then have the 3xx category which are redirection status codes, these are not that bad of codes to get as they are only meant to redirect a user. The request might have been successful but further action might be required to complete the request. Now, this category 4xx, is a developer or user's nightmare as it means that a client

error has occurred and the website or part of the site will not be able to be reached or be used by the user, thus resulting in a bad user experience. The last category 5xx is the server error, letting us know that the request could be fine, but the server is not able to complete the request. All these response codes are crucial to understand what is going on and important developers implement them in their codes to make it easier for the next developer.

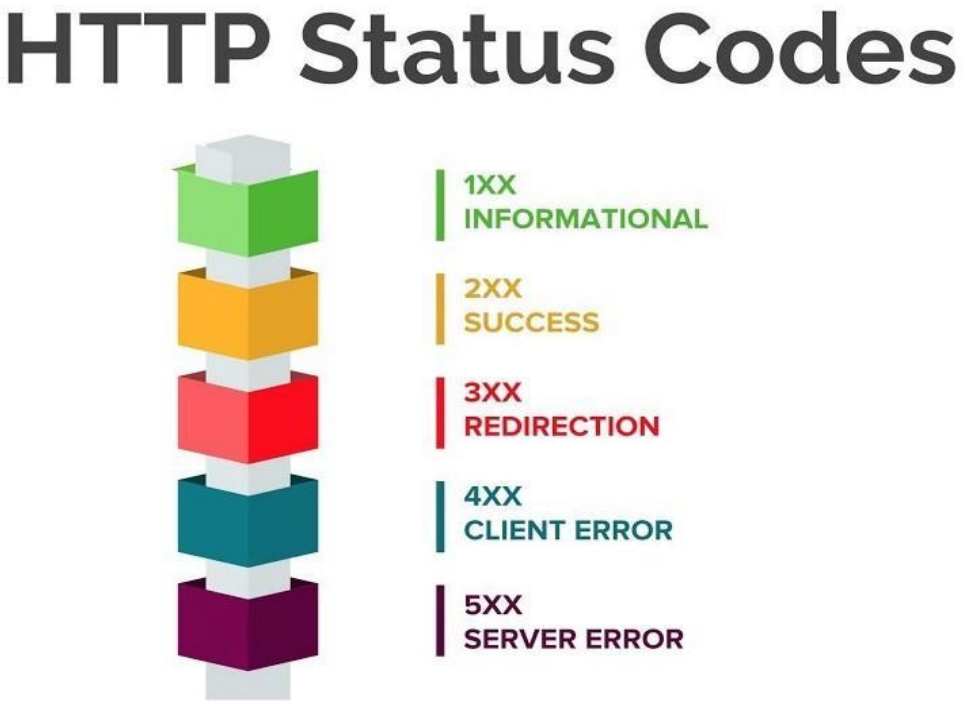


Figure 8.2.1.4: The five layers of HTTP Status Codes [53]

8.2.7 Testing

Our task as a team is to create and present a web application for the new Center for Humanities & Digital Research Lab, that will serve students as a means to navigate what the lab has to offer to them and all while making the workers in the lab lives easier. To accomplish such a task though, we have made sure that we produce bug free software and that all the necessary functions do what they were meant to accomplish. By far the most crucial part of our process of

producing this web application will be testing our software, because fixing bugs can be expensive and the longer a bug lasts the more expensive it'll be to fix [54].

Testing is an integral part of good software design. The ability to quickly identify failures in your code and go back and fix them is vital to ensure the quality of your product. Lack of testing is the reason teams are swamped with bug, after bug, after bug in the end stages of development. As such, our team will be taking every step necessary to implement good testing, in order to provide the best product possible to our client. But what exactly should we be testing for?

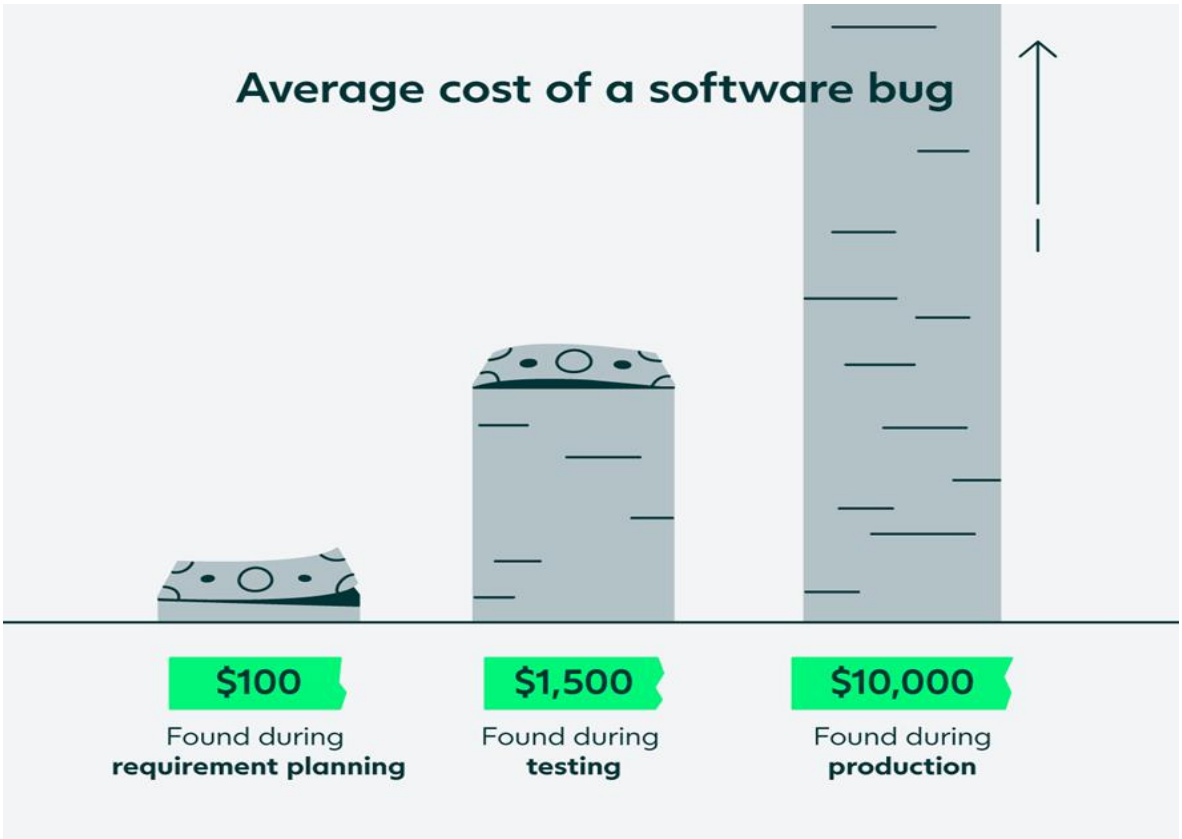


Figure 8.2.1.5: Depiction of the cost of fixing errors in your software in the different stages of development [54]

8.2.7.1 Guides to testing

As stated, prior testing is crucial in the development process of an application, and careful attention should be given to every aspect. A software tends to have many integral parts that make it whole, from the front end that allows the user to interact with the software and the backend that handles the grit of the work. So, if one part has a bug then this could lead to a domino effect, where other functions could develop a bug also. Therefore, it is beneficial that there are a set of tests that developers can follow to test their application.

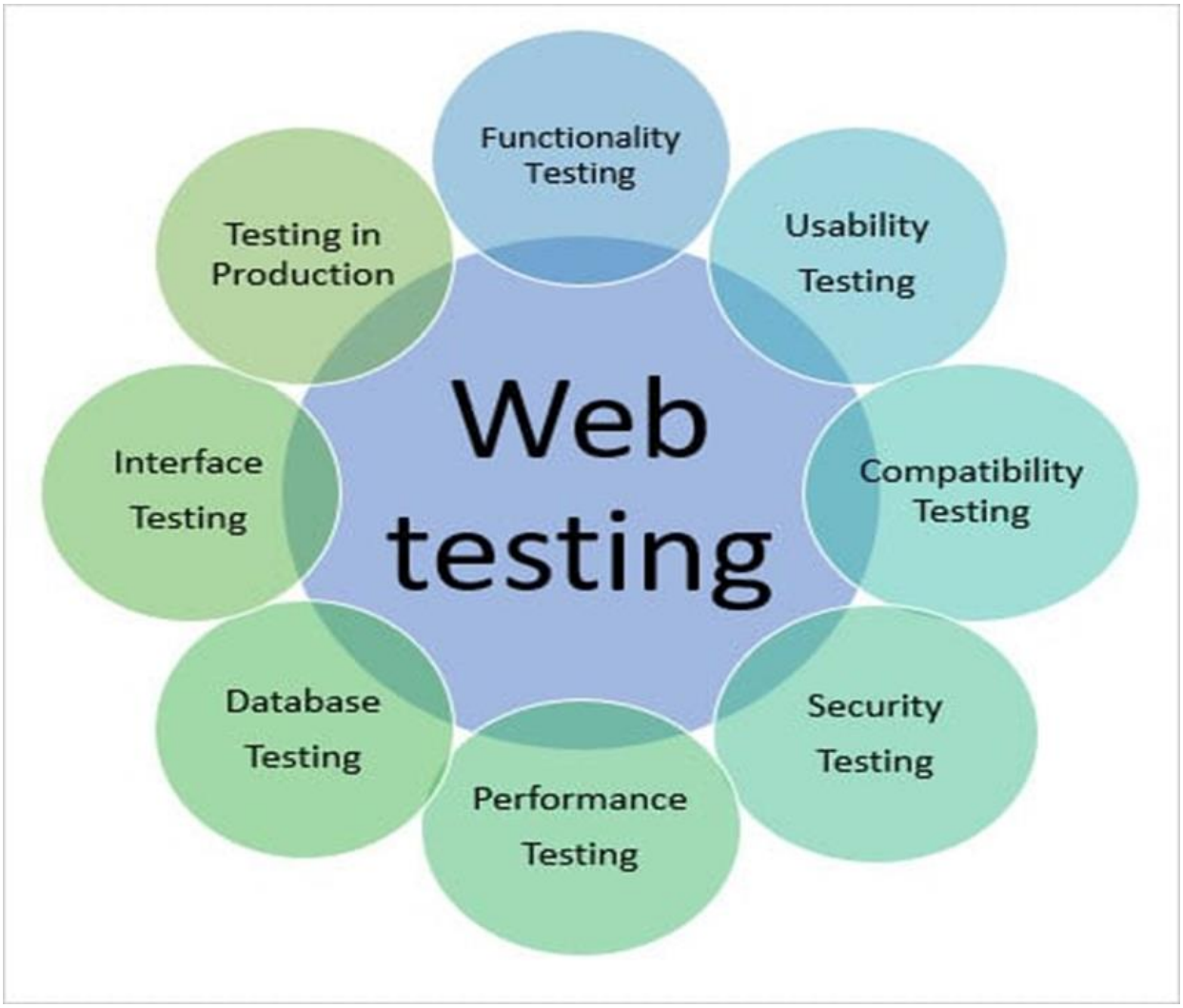


Figure 8.2.1.6: Different variation that should be completed on a software before deployment

8.2.7.2 Integration Testing

Integration testing is the next step up in our testing protocols. Put simply, integration testing broadens our scope of testing to examine the connections between groups of code. This includes the use of libraries, communication between database and server, or ensuring that the proper session information is communicated from page to page. Because these tests examine a larger system of code, it is important that you have unit tests completed for all components you are creating an integration test for. If you test a group for integration and it errors out, being able to identify which specific set of code is vital in order to quickly address the problem.

8.2.7.3 Functionality Testing

Testing the functionality of a software is checking to see if that specific function accomplishes what it was created to do. This is the time to check if all the kinks work, does the reference to a link work, is the route that connects the front-end with the back-end doing what it needs to and if data flows properly. You must check the outgoing links, internal links and properly see if they take you to the correct domain and whether they are broken. Another crucial part to look out for is to test the user interaction functionalities, making sure that all the buttons and forms are proper, that they are set with the right input value. Also this is when you should be checking that your cookies are encrypted correctly, as they will often play a direct role in maintaining a users session on the site, as they are usually the gateway to any security leak to a site.

8.2.7.4 Usability Testing

The process of which the human and computer interaction characteristic of a system are measured, weaknesses are then identified for correction. [soft] An application or website should be good to the eye of a user, while being easy to use and there should not be many complications on how to interact with a software once the instructions have been given. Contents are crucial on how they are placed on the pages; developers must be careful on the colors, fonts, and the placements of images on the page. The goal is to not have a user stress over using your application, so any method that makes it easy for the user then it should be taken into consideration to be used, for instance the main menu should be included on every page of the site, as it allows for easy access.

8.2.7.5 Interface Testing

A process that verifies whether the communication between two different software is done correctly. Interface could be defined as a connection that integrates two components, those can be anything such as API's and web services [55]. The crucial part of this test are the interfaces such as the application server to database interface and web server to application server. The purpose of this test is to make sure that the queries being made by applications are handled properly when errors occur, and it notifies the errors..

Essentially there are four types of Interface testing:

- **Workflow:** Make sure that your engine handles the necessary workflow that is expected out of it.

- **Edge cases-unexpected values:** Occurs when what is being tested for includes date, month, and day in a reversed manner.
- **Performance, load, and network testing:** Checks to see what sort of volume interface that we are dealing with and then decide on the amount of the load test.
- **Individual systems:** Every system is tested individually.

8.2.7.6 Compatibility Testing

A crucial test that needs to be completed, as we must check on the compatibility of our application which include:

- **Browser:** Checks to see if the application is compatible with the different browsers such as Internet Explorer, Firefox Safari and Opera. Not only must we check to see if the application pops up properly on the browser, but it is also a necessity to make sure that all the functions work properly as they should on every browser. That is why as developers you must ensure that your application is cross-web compatible.
- **Operating system:** Just as you must check for browser compatibility, you must check on the operating system compatibility with your applications. The various operating systems include Windows, Linux, MAC and many more but those are the three most popular. All those operating systems are different on their own and therefore certain features you might find in one might not be available in the next.
- **Mobile:** In the dawn of the twenty first century, technology has broadly advanced in its own way and cell phones have been at the forefront of this movement. Mobile compatibility will be important to our application as most administrative work will be done on a mobile phone and therefore having our website not be compatible will be an issue.

8.2.7.7 Performance Testing

Tests the performance of the web application, to see what exactly the application can handle. By using load testing and stress testing we can see exactly how the application performs in real life. Load testing consists of allowing for multiple users to access the application at the same time, all while passing a great amount of data to the database and seeing if the website will break or leak. The application must then be put through a stress test, which is used to break the site by giving stress and check as to how the system reacts to stress and how it recovers. [56] While testing for all these, we must also keep in mind the connection speed and check for the application speed, to see how it is impacted through all the wear and tear.

8.2.7.8 Security Testing

Another crucial test developers need to ensure that they do is a security test, most applications will contain important user data and therefore must be secured. We must be checking for vulnerabilities such as network scanning, password cracking, integrity checkers and virus detection. Password cracking is essential, as we must be sure that passwords are properly protected and hashed properly to prevent unauthorized access.

8.2.7.9 Automated testing

As you can conclude, manual testing for all the previous parts we have spoken about can be strenuous and take a lot of time since it requires everything to be done by a developer by hand. Not only must you check on every function by hand, you must also ensure that you also check log files, external services, and the database for errors. [57] Using automating testing, we can fix many of the issues that we face when manually testing an application, by introducing software tools we are able to cut the time spent on testing down drastically. We must make it be known that automated testing is not automatic, and that is a series of codes that will be written to be automated and tested on the specific application, meaning that your automation is as effective as the individual who wrote it. As for our application we will most likely end up using a combination of both manual testing and automated testing.

Why use automated testing:

- Executing your test becomes a lot simpler
- The execution of your test is faster
- Have a wider frame of test coverage
- Tests become more reliable
- Saves money and the time of the group
- Development cycles are shortening
- Reduces the maintenance cost of testing

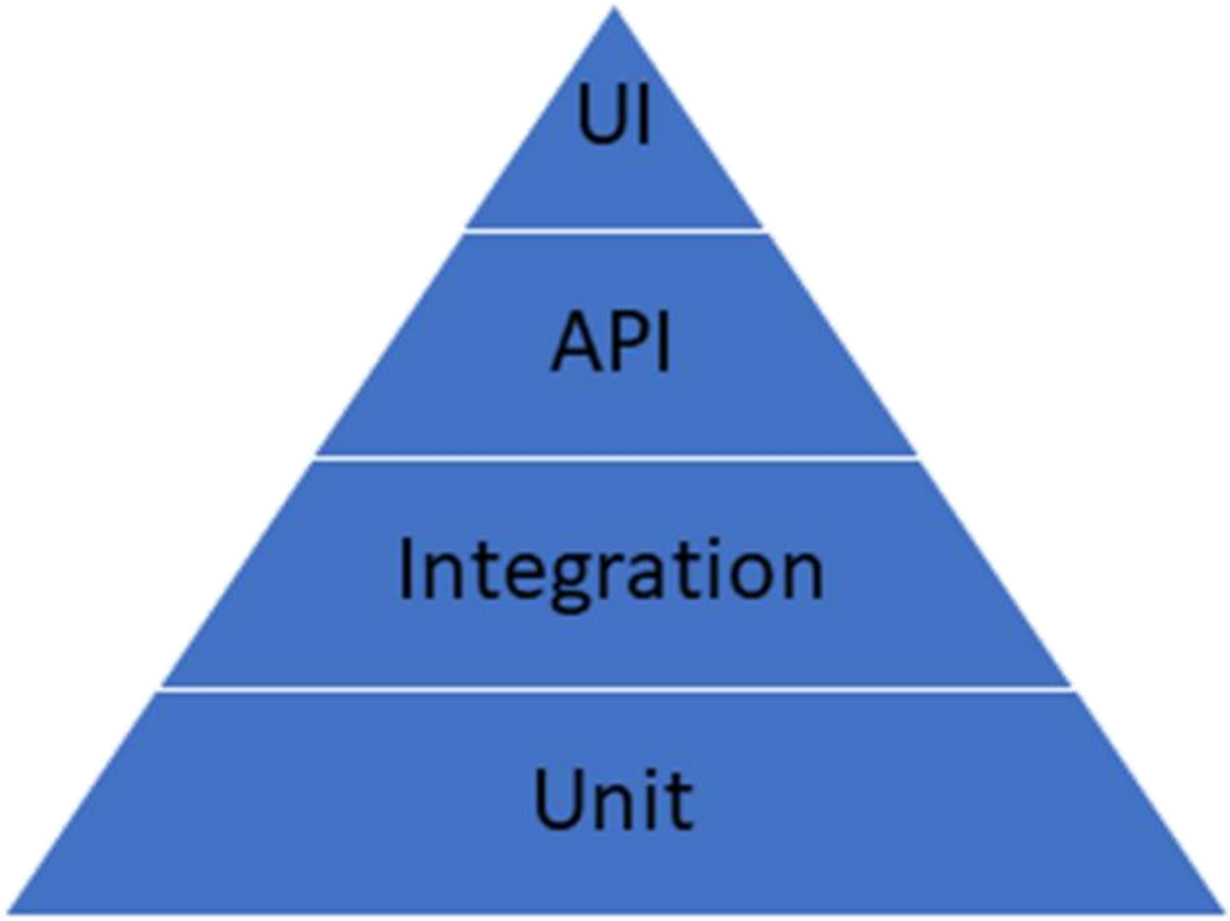


Figure 8.2.1.7: Automation test pyramid [57]

Setting up a strategy to follow when testing is the best thing to do, as it makes it easy to have a plan laid out to follow. As shown in Figure 8.2.1.7, by following such flow in our strategy it allows us to see how much time and the importance level of the spec that will be tested for. We have unit testing at the base of the pyramid, then the service layer or API testing and GUI testing at the top that makes up the pyramid.[57] Furthermore, those who take part in test automation are developers who are very skilled and are very experienced in writing test scripts. But with the advancement of technology, developers made it possible to provide software that makes it easy for beginners to get started on automated testing.

Unit testing is a small code fragment, or a sort of miniature program, that takes a small portion of our product code and runs sets of input through it to evaluate expected output. Input can cover standard use cases, and will often extend into various edge cases that are unlikely to appear in regular use. This is to ensure that even under non-standard use-cases, the product (or rather this particular fragment of product code) functions as expected, even under conditions that it is not meant to function under.

Put simply, unit testing is the process by which we test the smallest possible fragments of our code so that we know that every command we put in the product functions without issue in isolation. The keyword here is "isolation". This testing is not performed on larger chunks of code that interact with one another. That is reserved for integration testing and functional testing, testing that ensures that the product as a whole is functioning well together. Once you know that each individual part of your code functions well on its own, you can identify that the problems are not with individual pieces, but rather with the components that connect those individual pieces.

This form of testing is going to be the simplest to write, but also the most common form of testing you will find. Because it's testing smaller units of code, you need to write a lot to cover everything. As such, it is recommended that you write these tests as you write your code. This ensures that no sections of code go untested and cannot be identified as bugs.

8.2.8 GitHub and Git

As a software developer Git and GitHub are two must-have software that one must have on their working computer. As one might infer those two go hand to hand though, even though they can be deemed independent of each other. Git, which was first created in 2005 is a version control system, which basically allows a developer to be able to keep a record of their development throughout the whole process and is installed and maintained on a local system. [58] Meanwhile GitHub was created to be a hosting service for Git, this allows the ability to create your own profile with built-in control and task management to expand on the work that has been done on Git. Whereas Git is only permitted to be on your local system, GitHub is served on the cloud which allows a user to share their work to the public or just authorize certain users to only have access from anywhere in the world they are located.

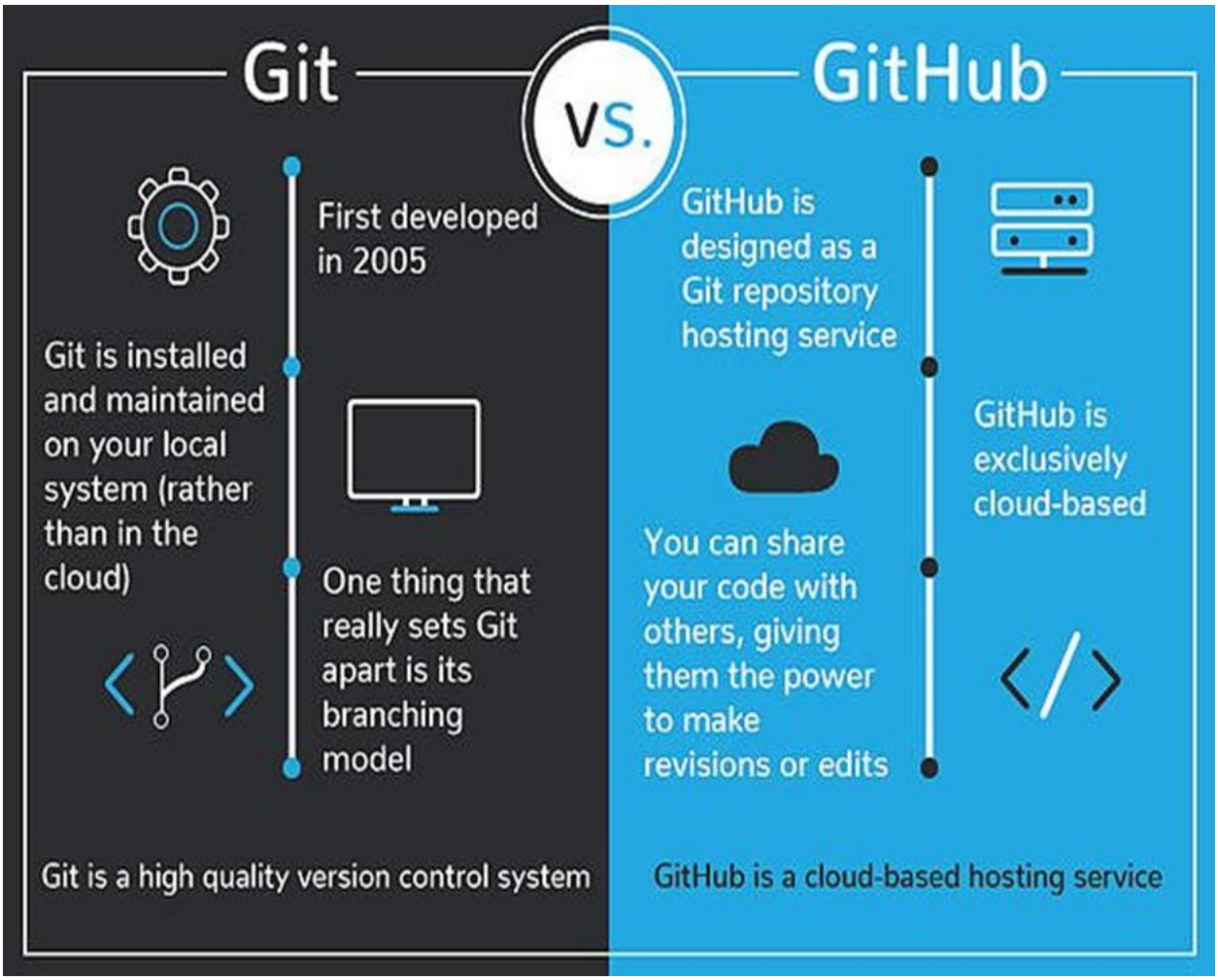


Figure 8.2.1.8: Git versus GitHub, what are the differences between the two and how do they correlate with each other [58]

8.2.8.1 Version Control

The process of creating a large piece of software as we are tasked to do can be tedious for an individual to do on their own and that is why this is set as a group-based project. However, with many hands working on a project, many roadblocks can arrive, especially when multiple individuals are working one file. With a team of five certain aspects required for multiple heads to work together and to eliminate some of the issues that might arise, we decided to use Git's version control capability..

- Issues that might arise developing software:
    - Miscommunication amongst your team

- Constant work on the production server
- Incapability of being able to backtrack your work
- Lack of Backup
- Difficulty of integrating and delivery

We stated all the issues that developers might face when it comes time to produce software but with the use of version control, you are able to correct and fix many of these issues. Therefore, as a team we decided as a team to use Git and GitHub to manage the workflow of our files. This will allow us to work in peace knowing that every member will maintain their own space to work and having the confidence of making changes to each other's files, with the capability of reverting to the file at any point of time while knowing who made what changes. The use of version control will come in tremendously handy, when it comes to deploying our web application, as we are given the capability of being able to push different versions of our files to our production server to test on.

Version control as you've read so far is a powerful tool that makes it easy to work in a team setting. However, there are some concepts that you must know to use version control, and everything is centered around Git. You first must create a directory with the files you would like to work on and clone a repository which are the files you'd like to have under version control. A great feature is having all the modification work done and tracked and saved automatically, but nothing is final until the changes are submitted and committed. [59] As stated before revision is a great feature and once you have committed your work, all the modifications that will occur will be known as a changeset and associated with it will be a random integer. This random integer is what will allow us to be able to retrace our work at any previous version, not only so the changeset associated with it the user has made the changes.

Furthermore, when it comes time to continue the work that has been committed to a repository, you need to do a pull request and must get the latest version of the file you would like to work on. Pulling the latest version of a file minimizes room for error and conflict that might occur, since it defeats the purpose of working on an older version that does not deem correct.[59] That is where diffing comes in handy, which allows you to view the difference between files as you change them and see where the errors have occurred. Diving deeper to being able to work on different versions of your work, we have the concepts of branching and merging which push version controls further. Branching is creating a copy of your main repository and being able to modify that copy, while also working on the original one. Which makes sense for the name Branching as you can make multiple copies and it would resemble a tree with several branches.

Once all parties feel satisfied with their work, we're given the opportunity to merge all the branches of files back together. The system makes it easy to merge, since it keeps track of all the changes that have taken place in the branches, it will merge each file and line of code automatically.

8.2.8.2 Types Version Control

8.2.8.2.1 Centralized Version Control

Often considered the more popular of the version control systems and can also be referred to as subversion. [59] The thought process behind this system is that we have a repository that is located in one place and individuals at any given location can gain access to it. Some of the benefits of this system is its ease of use, not too complex to understand and it is simpler to control which user has access in which to what capacity of access that they have. Using a repository that is located in one location does have its downfalls, such as managing servers and backups or being able to use branching and merging tools.

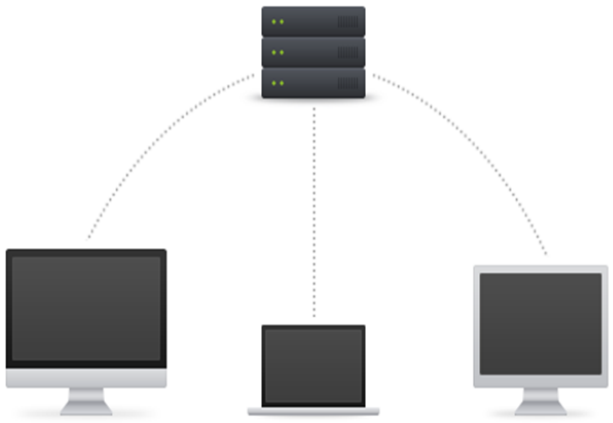


Figure 8.2.8.2.1.1: A centralized version control unit

8.2.8.2.2 Distributed Version Control

However, for our team the better version for us to use would have to be the distributed version control. Since we will be working from the comfort of our own home, having our own repository, and the history of all our files will be a big contribution to our workflow. Some of the upside of using this system version is how fast it is and the fact that a server isn't necessary, meaning that actions or any work is capable of being done on your local system. However, some cons of this system is that the distributed model is harder to understand but most importantly is the fact that the revisions are not incremented using numbers, therefore it becomes harder for users to revise them.

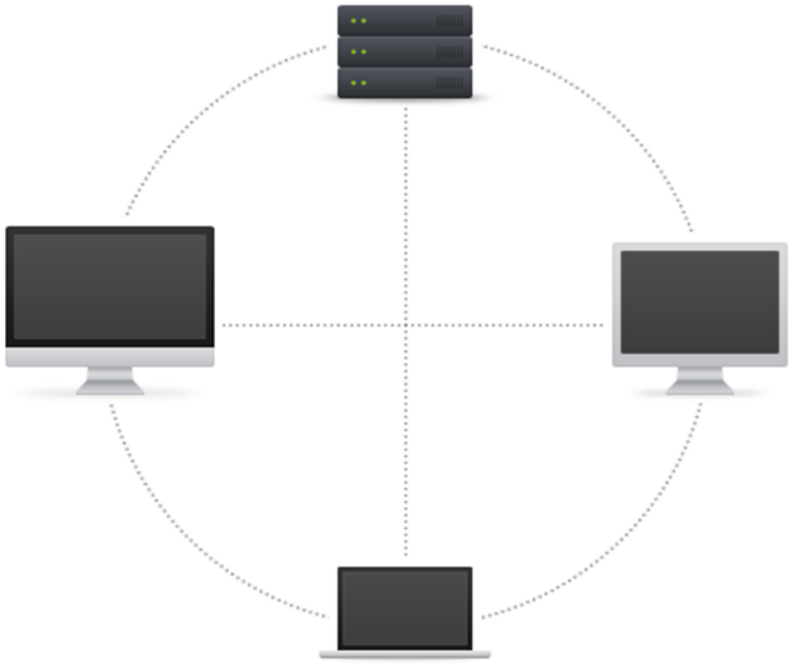


Figure 8.2.8.2.2.1: A Distributed version control unit

8.2.9 GIT Implementation

MacOS and Linux

Both operating systems already have GIT installed already and it is as easy as just checking the version on the terminal by inserting:

Windows

1. Visit https://git-scm.com/download/win and figure out which bit is compatible for your use and download

2. Once fully downloaded and installed, go to your terminal to ensure that the proper version has been installed.

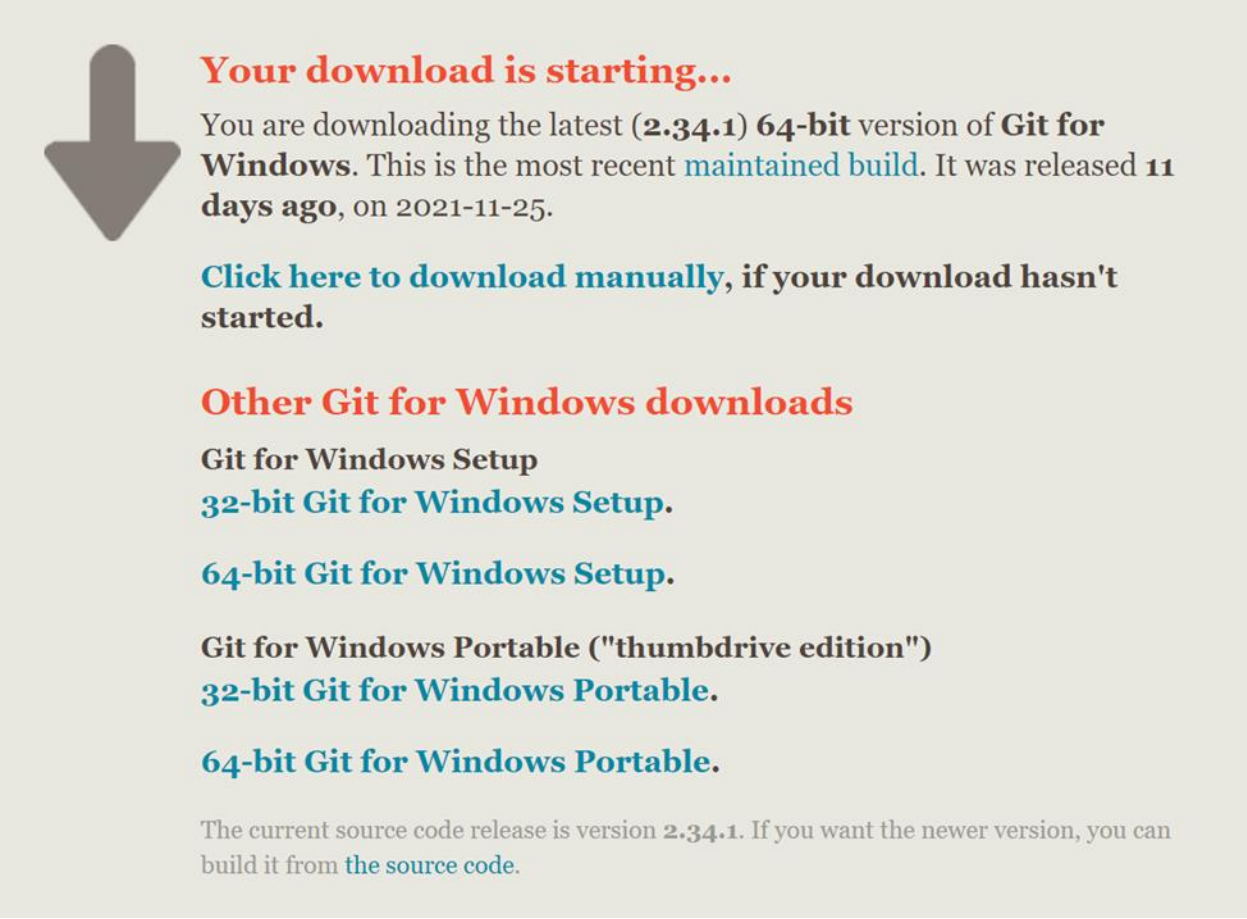


Figure 8.2.1.11: Git download page

8.2.10 Insomnia

There are various applications from ARC, Postman, that we could have used to test our endpoints that we will be creating. However, we decided to go with Insomnia with its friendly user interface, while also allowing you to organize, run and debug your APIs. One of the key features that Insomnia offers that most other HTTP request software do not offer is being able to

resolve a variable from another environment variable. Another is the ability to be able to save your requests and being able to know when your response time came whether it was successful or not. Not only this feature allows you to easily collaborate between team members and check on each other's work. [60]

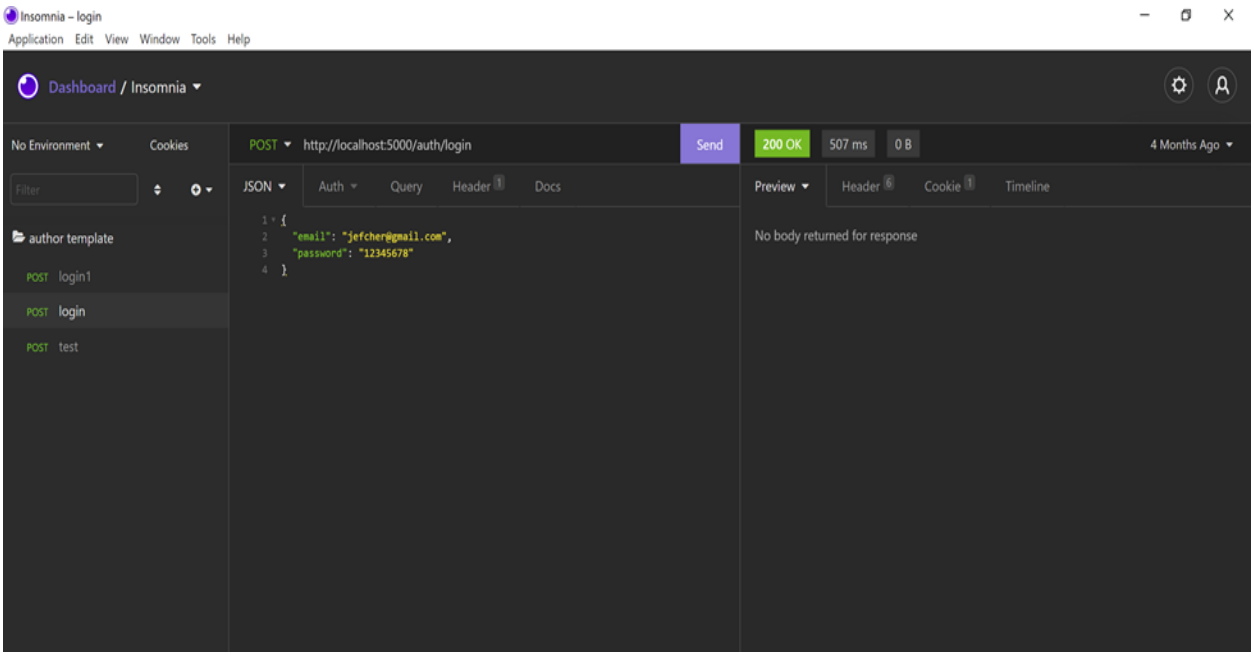


Figure 8.2.1.12: Insomnia test page and user interface

8.3 Website Research

8.3.1 An Analysis of Layout, White Space, and Accessibility

Visual hierarchy is a visual design principle used in UI/UX design, as well as front-end development. It's a technique used to determine the importance of certain content based on how the content is arranged. This technique isn't only about layout specifically, however. There are a number of aspects that go into this, such as size, color, contrast, proximity, white space, alignment, positioning, etc. Visual hierarchy can be used to persuade the viewer to take certain actions. Using principles of visual hierarchy, companies are able to increase the screen time of their user base, therefore equating to a larger profit gain for the company.

Let's look at this in the perspective of a real-world application. Facebook has its main source of navigation located towards the bottom, with additional navigation options located towards the top. This is a prime example of using visual hierarchy to focus the user's attention on what seems to be the most important navigation option. In a sense, we're somewhat inclined to

gravitate towards content if its reachability is within a comfortable zone. These are just a few techniques that a UI/UX designer can use to stand out from other corporate designs.

8.3.2 Influencing Decisions with Visual Hierarchy

Laying out elements logistically and strategically, designers can have an indirect influence on a user's perception of the importance towards certain content. Paired with contrast in size, a designer can make it apparent to the user what actions may take precedence over others. This technique is known as visual hierarchy.

Sizing is a basic but important design principle. It alone can be used to rank the importance of an element and influence the order in which you want the user to view content. By increasing the size of an element, you immediately attract the user's attention to that specific element. We can see this by looking at the landing page for various businesses. In fact, UCF in particular does this when displaying statistics:

Figure 8.3.2.1 – The statistics section of ucf.edu

Here, we can see that this design decision is clearly an attempt to draw the user's attention to the statistics on the right-hand side of the page. Although these elements are larger than the rest, they're larger in size without decreasing the importance of the description on the left. In an age where it's difficult to stand out, designers need to be able to grab the user's attention in a matter of seconds. As such, scale is one of the simplest ways to do so.

Let's analyze a more discrete example. This paper is organized into sections. There exists a main heading, which describes the overall topic of all of those particular sections, there are

sub-headings, which describe the overall topic of that particular sub-section, and there is body text. Each of these three elements have decreasing font sizes that in turn represent the importance of that text to the viewer. When any document follows this style, it's easier for the reader to digest that document as a whole. When scrolling or skimming through such a document, your attention is drawn to first the main heading, then the subsequent sub-headings, and finally, the body text. This allows a reader to get a grasp on what they're reading without actually having to read the entirety of the document. This is far from a new technique.

Johann Carolus, a 16th century German author [61] is often credited with publishing the very first newspaper [62, 63]. What's interesting about this newspaper is that the headings follow the style referenced above. A title that stands out from the rest of the document, and subtitles, that stand out from the rest of the text. In fact, this design choice happened to be so effective, that newspapers today all around the world can be seen with this style. It even bled into typographic design for the web (a topic which is discussed in a later section). A website like the New York Times has a lot of content on a single page, but by incorporating scale and different sizing with a mixture of typography, they're able to better organize their content and grab the user's attention with only a single headline.

Another example of using visual hierarchy to influence a user's decision or action can be instantly seen on the home page of various companies. When visiting the landing page of a streaming site like Netflix, Hulu, HBO Max, Disney Plus, etc., we're greeted with large bold centered text with the first immediate action being to sign up for the service. This design isn't limited to streaming services either. GitHub, Amazon's AWS, Heroku, and Netlify, just to name a few, all have their primary action, the sign-up button, either in the center of the page, or positioned in such a way that it stands out from the other content yet without abruptly stealing the focus from other actions.

Proximity plays an important role in design as well. Proximity, often interchangeably referred to as grouping, is the principle that concerns the effect generated when a set of items are placed close to each other. A set of collective elements can become more meaningful when in close proximity to one another. This principle can commonly be seen in modern day UI/UX design when we study the navigation bar. Most websites put their navigation bar at the top of the page and include links to services they offer. These links are almost always grouped together and within close proximity of one another. Leveraging proximity is even reflected when creating basic text content. Large blocks of text are grouped together in paragraphs to convey meaning. When we read text that's grouped together, we automatically, perhaps even subconsciously assume that the text we read is related to the text that preceded it. Books group content into chapters, textbooks group their content into sections and subsections, even restaurants place related menu items next to each other and group them into categories like drinks, appetizers, entrées, etc. This observation has even been studied in Gestalt psychology, a school of

psychology that emerged in the early twentieth century in Austria and Germany as a theory that humans often perceive patterns when introduced to groups of similar objects. These observations are categorized into a set of 6 principles called the Gestalt Principles of Grouping [64]. These rules include proximity, similarity, closure, good continuation, common fate, and good form.

Related to proximity, similarity is one the Gestalt principles of grouping that can be seen in UI/UX design. The principle of similarity states that human perception tends to lend itself to seeing stimuli that closely resemble each other as part of the same object. Companies play on this law of similarity by mimicking each other through their designs. Many news sites follow the same typical format under this principle. A large image on the landing page with a bold text overlay that directs attention to the most important article, and two to three columns of other related news. Companies use this technique because they realize that if we're used to seeing a particular design, then introducing some new product that follows that same style of design will usually be met with good reception. In a real-world example, it's the same reason my most airports in America have the same layout, or why most fast-food chains have the same layout for their drive through, or why most gas stations have the same layout, or even why almost all menus in restaurants follow the same categorization rules: it's all familiar to us. However, companies may even use this principle somewhat mischievously to deceive someone into consuming their product. The movie industry has even coined the term "mockbuster" [65], which is a film created solely to exploit the publicity of another major motion picture film. Because similarity is so ingrained into our everyday perception of life, designs are able to stand out when they break away from what we're used to. This is the reason why Apple's infamous iOS 7 redesign was so notable. They broke away from the typical skeuomorphic design in favor of a flat minimalistic design. The same can be said for Microsoft's update from Windows 7 to Windows 8.

So how do these topics relate to the project at hand? UCF's Center for Humanities and Digital Research department (CHDR) requires an inventory system that allows users to check out some item they possess. As such, it's given that there has the potential to be hundreds of items that need to be tracked. When designing for a system like this, we need to consider the fact that not all items may be related to one another. If a user is looking for a particular item, one might expect to be able to find that item by either searching for some related item, or by checking some category that may include that item. This is a concept that could benefit from the rule of proximity. Grouping similar items could make it easier for the user to find what they're looking for.

8.3.3 White Space and its Benefits

Whether it be consciously, or subconsciously, there are two things a user notices in a design. The presence of content, and the absence of content. White space acts as a wall

separating these two. In UI design, white space, or sometimes called negative space, refers to the empty space between or around elements of a design or on a page. This is most notably seen in the spacing between paragraphs of text, or the spacing between pictures, buttons, icons, or other items on a page. White space has a purpose, but its purpose differs based on the use case. It's common to see white space between paragraphs and characters of text because that spacing makes the document easier to read, but one might also use white space to add emphasis to a specific object. To put this into perspective, we can look at a website like Apple.

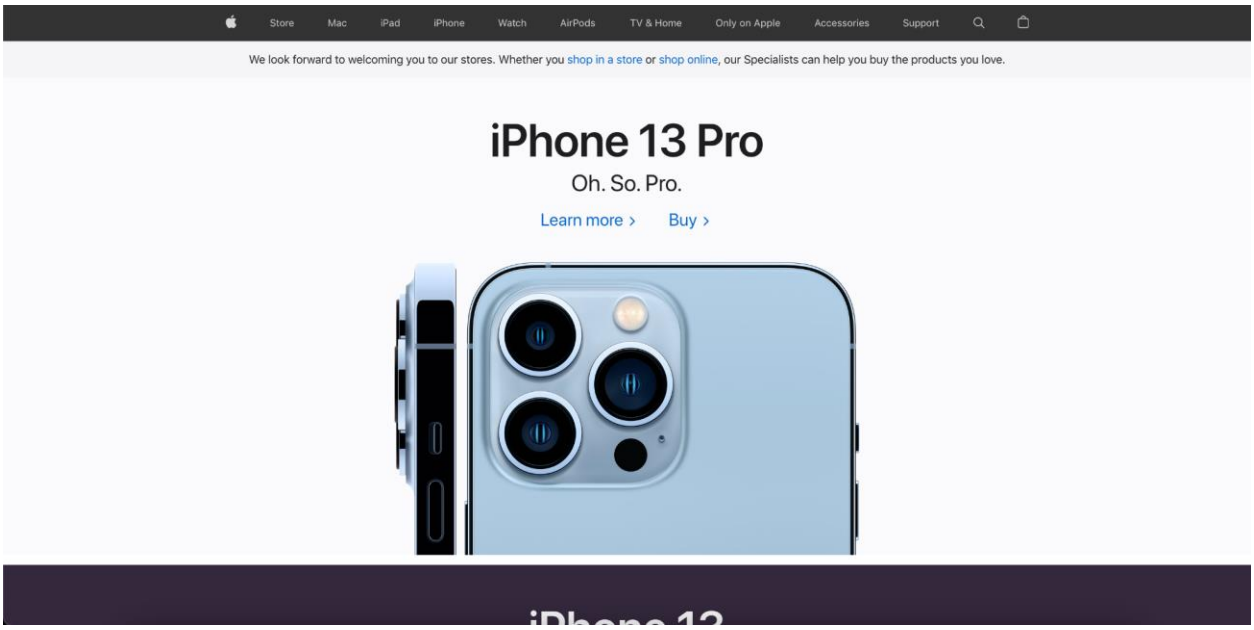


Figure 8.3.3.1 – Apple's landing page

A common misconception is that white space is wasted space. It's easy to look at a page with a lot of blank space and think something is missing. Often, designers may want to crowd a blank page with an abundance of images or content with the hopes that more images or more content will keep the user on the site longer. Usually, this isn't the case. We use white space as a form of contrast. In fact, the real meaning of white space becomes apparent when you look at other examples and their opposites. For example, we notice that lighter elements stand out when paired with darker elements. A large button grabs your attention when the content around it is smaller. Similarly, you can emphasize an element when you incorporate space around that particular element. Imagine a shopping site cluttered with pictures of clothing. A user may spend a few minutes deciding on what to buy only to not buy anything at all. Perhaps they glance at an item for only a few seconds before looking at the next item. Overall, they may spend on average only a few seconds looking at each item. If we incorporate white space and let our designs breathe, we can draw the user's attention to each item on the page. We shouldn't let the user waste time on searching for something, and this is a problem that can be solved by white space. We can manipulate white space with the use of margins, padding, line spacing, and line height.

White space can help us better process what we see. As an example, social security numbers are grouped into three sections of three, two, and four, in the form: XXX-XX-XXXX. For us, it's easier to memorize. Credit and debit cards do the same (although we don't normally memorize these numbers), they're split into four groups of four numbers in the form: XXXX-XXXX-XXXX-XXXX. This is because humans process and memorize data in a form known as chunking [66]. Generally, a chunk is a part of something larger. In terms of UI/UX design, chunking usually refers to breaking up content into smaller and more distinct units of information as opposed to presenting content all at once. According to the Nielsen Norman Group, some of the most commonly used methods of chunking involve splitting text. For example, short paragraphs that incorporate white space around them, short lines of text (approximately 50 to 75 characters), clear visual hierarchies with related items grouped together, or as discussed above, distinct groupings of strings, letters, and numbers, like a phone number for example. Splitting up content into chunks usually isn't enough on its own. There should be content to support these chunks as well. This could be by using headings and subheadings that clearly contrast with the text it describes (we can do this by changing the color of the heading or using a different font and font size), using captions that go along with images, or adding short summary sections to articles.

We can pair chunking and white space to make a design that's easy for a user to digest. Let's analyze an example as old as web development itself: the form. Say for example we have a form that allows the user to sign up for a roommate matching service. This form would have quite a lot of fields since we'd have to include a text field for the user's name, a field for their location, including subfields for city, state, zip code, etc. We could have all of these fields run from top to bottom in one long web page, but that would make it hard for the user to follow. They'd become fatigued trying to figure how many steps they have left before they complete the form. What if they need to go back and modify a section they entered incorrectly? They'd have to go scroll back and forth looking for just a single section to correct. This is where we can incorporate white space, chunking, and the Gestalt principles of grouping we mentioned earlier in section 1.2. If we take this roommate matching form and split it into clearly labeled sections, it'd be easier for the user to progress through the form without them becoming fatigued. The idea would be to take sections similar to each other, group them into blocks, and add a fair amount of white space in between each section. We may have a section specifically for location, or even a section specifically for interests.

The combination of chunking and white space may sound promising, but it's important to remember that there's a limit to the amount of content a user can take in at once. A study conducted by psychologist George A. Miller in 1959 found that most individuals have a short-term memory capable of retaining on average 7 (plus or minus 2) chunks of information at any given time [67]. What's more interesting is that the study even found that most individuals can

memorize up to 28 individual letters if they were grouped into chunks of 4 words each. We can see examples of this when we venture into real world designs.

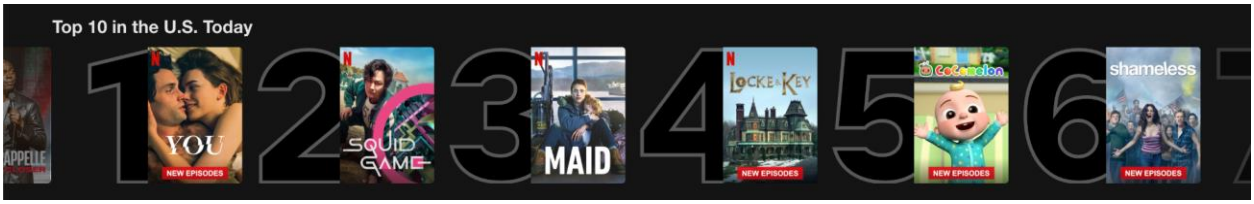


Figure 8.3.3.2 Netflix's Top 10 Section

Netflix is a site that has hundreds upon thousands of shows, movies, and documentaries to watch. When you visit Netflix's home page, you aren't bombarded with every single option at once, but a subset of items based on what Netflix thinks you're interested in. In their "Top 10 in the U.S." section, we can see how they incorporate the idea of chunking discussed above. They don't display all 10 choices at once, rather a list of only 7, Miller's magic number. HBO Max and Hulu have similar designs. They group their shows into categories and only display a small amount of shows at a time so as to not overwhelm the user with content. They all incorporate Gestalt's principles of grouping and proximity, where content is given an appropriate amount of white space to emphasize the idea that each section of content belongs to its own group. As we can see, white space plays a huge role in UI/UX design, and paired with chunking, not only can it help content stand out, but it can also help users digest a large amount of content.

One unique take on incorporating white space in a design is a design pattern called center stage. Center stage involves putting the most important part of the UI into the largest section of the page with supporting elements and secondary content being spread around the center. A popular example of this is Microsoft Word. The UI is built in such a way that the editor, which is positioned in the center of the screen, stands out as the most important part of the application. Controls, such as options to change the font size or color, are the supporting elements. The same can be said for Microsoft Paint. In fact, this design pattern seems to be a popular choice among applications meant for designers and developers. Adobe's Photoshop and Illustrator, Procreate for iPad, Visual Studio Code, and even Figma use this design pattern. The primary use of center stage should be to guide the user's eyes immediately to the start of the most important element at the center of the page. This central entity should grab the user's attention. Figure C8.2 uses a combination of center stage and white space to do this. Once the user's attention has been stolen by the central element, the user will assess secondary content in terms of how it relates to the main content. For example, the editor in Microsoft Word is the first item to grab your attention. Because you know the purpose of this central element, the content around it gains its context from the understanding of how you interpret the purpose of the editor.

8.3.4 The Importance of Reachability

According to Google statistics [68], more than 50% of web traffic comes from mobile devices and about 50% of mobile users are likely to browse a company's website on their phone because they don't want to download an app. Thus, when creating mockups, designers often need to account for mobile traffic. In fact, one of the major philosophies of UI/UX design is " mobile first". As the name implies, mobile first suggests that designers start with the product design from the mobile screen and work their way upwards to the desktop version. When we consider that mobile traffic now makes up more web traffic than that coming from the desktop [69], we can see why a mobile first approach would be important.

Because mobile traffic accounts for such a large percentage of web traffic, designs need to account for this. Designing for mobile devices presents a different set of problems than web design. For starters, it's obvious that there's less screen real estate to work with. One can't simply shrink a desktop design to fit on a mobile device and call it a day. Content needs to not only be readable, but it also needs to be reachable. Figure C8.4 contains a head map of accessible areas on a mobile device.

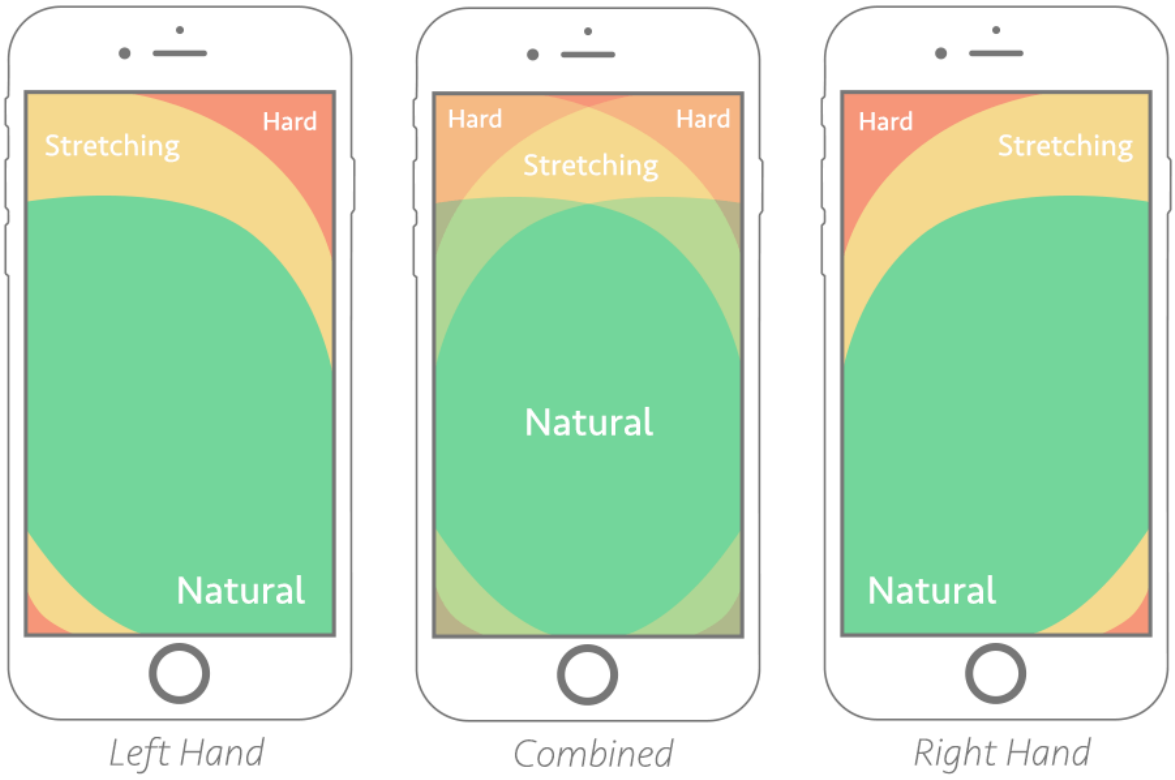


Figure 8.4.4.1 - Thumb zone mapping for mobile devices

When we analyze this, we notice that when using one's left or right hand, a large portion of the bottom of the screen is much more accessible than the top. As such, it makes sense that applications, such as YouTube, Facebook, and Instagram all have their navigation bars located at the bottom of the screen. It should be noted however that the figure above only accounts for a mobile device with an average screen size. There are phones with screens much larger and much smaller than this. We also need to consider that not everyone has the exact same hand size.

Different actions require a different amount of the screen to be taken up by content, which in turn may also require users to switch hand positions frequently. This could potentially lead to hand fatigue. Popular apps like Instagram and TikTok are designed in such a way that users don't need to reposition their hand to use the main part of the app. Instagram keeps actions such as liking, commenting, and sharing at the bottom of each picture because that's where the user's hand naturally rests regardless of their hand dominance. The same can be said for TikTok. Users don't need to reach an uncomfortable amount to watch a video or share a video with their friends because the controls lay in an accessible area. The problem of navigating to more content is solved by swiping up and down, an action that can easily be done with only the thumb. Design choices like this are what keep users engaged in the application for such a long amount of time.

Users shouldn't have to think about how they need to hold their phone in order to access certain content. As phones get bigger, designers need to constantly think about how reachable their designs are.

8.3.5 Considerations in Accessibility

Accessibility is so important in web design that there's an entire standard dedicated just to document it. Because most in the population don't have first-hand experience with a disability, it's easy to ignore design choices that would help those that do have a disability.

JAWS (Job Access with Speech) is a popular screen reader with speech-to-text capabilities that can help the visually impaired navigate the web. It works by announcing content to a user based on specific elements of the screen, allowing the user to create a mental image of what they're navigating. This topic gets a little complicated when we introduce websites with complex HTML layouts. It gets even more complex when we add JavaScript. Say for example we have a modal that asks the user if they'd like to consent to cookies. Screen readers will have no problem reading the text and headings on the modal, and they won't have any problem announcing the text inside buttons either. But what happens when one wants to close the modal? For us, we've been conditioned into believing that a button that says "X" means close or cancel, but when a screen reader announces "X" to someone who's visually impaired, that close button no longer has meaning. We know that action will probably close the dialog because we can physically see that we're in a dialog. That close button gains its context from its surrounding

content. When you take away the surrounding context, you lose all meaning. This is why aria labels are so important. Developers can label that “X” button with text that instead says “close”. This way, users with a visual impairment will hear “close” announced to them instead of “X”.

There’s another type of visual impairment known as color blindness. The two main types are red-green color blindness and blue-yellow colorblindness. Each category has different subcategories. Red-green color blindness has 4 types: deuteranomaly (makes greens appear redder), protanomaly (makes reds look greener), protanopia, and deuteranopia (both make you unable to tell the difference between red and green). Blue-yellow color blindness has 2 types: tritanomaly (makes it difficult to tell the difference between blue and green, and between yellow and red), and tritanopia (makes you unable to tell the difference between blue and green, purple and red, and yellow and pink). Red-green color blindness is the most common type of color blindness and ironically, red and green are often the most popular choice in design when used to indicate actions that are correct and incorrect (in American designs at least). Imagine you’re a student taking a quiz and after submitting, you’re able to access information on what questions you got correct and incorrect. If incorrect answers are outlined with a red border and correct answers are outlined with a green border, someone with protanopia wouldn’t be able to tell the difference between the correct and incorrect answers. Just like the example above, we give meaning to the colors red and green based on the context of the surrounding content. To someone without a visual impairment, red represents erroneous input while green represents correctness. This is why it’s important to consider various visual impairments when designing. Some designs may lose a large amount of context when we take away specific visual elements.

8.3.6 A General Categorization of Various Typefaces

Considering most of what we do on the web is read, typography is an important part of any design. A good font can be the difference between a website that was designed to appeal to children, and a website that was designed to showcase news articles. Designers aren’t limited to just one font however, there are entire studies dedicated to finding the right font pairings. Because there are thousands of fonts to choose from, it can be difficult to determine which font one should use in a particular design. Luckily, fonts can be broken down into categories to help determine when one should be used over the other.

There are five basic types of fonts: serif, sans-serif, script, monospaced, and display. Serif fonts are fonts that have extra strokes on the end of their letterforms called serifs. These fonts invoke a sense of formality, history, tradition, and integrity and are generally used for either headlines, titles, logos, or body text. They’re a great fit for content that needs to be physically printed because the stroke of each character helps the letters stand out clearly. This is why newspapers use serif typefaces. Unfortunately, that same serif that makes printed media more

legible can also make small text on smaller screens less legible. There are, however, serif fonts that are good to use in print as well as digital media, such as Times New Roman. Serif fonts themselves can be further divided into 4 categories: old style, transitional, slab serif, and didone (also known as modern serif).

Old style serifs can be recognized by their wedge shape ascenders, that is, the part of the font that rises above the main body of the font is shaped like a wedge. Old style fonts also have smaller x-heights (the distance between the middle of the font and the baseline, where the characters sit). Old style can be broken down even further into renaissance style typefaces and baroque style typefaces, although, it should be noted that the difference between the two mostly lies within how certain characters are written and displayed. Garamond is considered one of the most popular old style fonts because of how legible and formal it looks in print, but it's not recommended for use in the web because most screens have difficulty displaying the font as clearly as other serif fonts. There are variations of the Garamond font like EB Garamond, designed by Georg Duffner [70] to be a modern take on the classic font that's suitable for use in websites.

Transitional style serifs are a mix between old style and didone, hence the name "transitional". These typefaces were common around the mid 18th century until around the start of the 19th century. Transitional serif fonts have less contrast in the stroke of each character than that of old style fonts. The ends of the characters are typically marked not by angled wedges, but by smaller round ball terminals. This can be seen with the "y" character of the Times New Roman font. Because this style of typeface falls in between the popularity of two different genres, it's difficult to define where the genre starts and ends.

Slab serif fonts date back to the early 19th century. These fonts were meant to grab your attention when used with designs for posters. These fonts typically have thick serifs that match the width of the vertical parts of the font. Because of how clear these fonts were, they were often used in small print media. A few examples of slab serif fonts include Courier, Rockwell, Zilla Slab and Excelsior just to name a few.

Finally, we have didone, or modern serif. This first emerged in the late 18th century and is characterized by the extreme contrast in thin and thick strokes. Because of this contrast, these fonts often mimic the typographic designs made popular by calligraphers. Out of all serif fonts, these typefaces tend to be the least legible of all. This is usually why didone fonts were used for the large attention-grabbing headlines of newspapers and other print media such as book titles because the font needed to be quite large to increase legibility. Examples of didone fonts include Bodoni, Gallient, and Walbaum.

Sans-serif fonts are fonts without serifs on the ends of each character. These fonts have become one of the most popular choices for use in display on digital screens. On lower resolution screens, the small details of serif fonts may start to disappear because some screens may have trouble rasterizing the serifs of each character. This is why the lack of flourishes and detail in sans-serif fonts make them such a great choice for digital media and modern minimalism. Just like serif fonts, sans-serif fonts can be broken down into four categories: Grotesque, neo-grotesque, geometric, and humanist.

Grotesque fonts don't have much variation in their stroke widths leading to a relatively uniform appearance that gives a lot of visual character. These fonts are also usually geometric in design with simple letterforms that are great for display purposes when used in more bold weights. One of the earliest grotesque fonts, Ideal Grotesk, was designed by the Berthold type foundry in Berlin to be used in headlines for print media.

Neo-grotesque fonts put their emphasis on legibility, neutrality, and minimalism and aim to have as few strokes as possible. This category of fonts was also one of the first typefaces to give birth to fonts that included variations in character width and weight. These typefaces feature less contrast and more regularity and consistency between characters. Popular examples of neo-grotesque fonts include Helvetica and Arial.

In geometric fonts, basic shapes like circles, triangles, and squares play a huge role in the shape and design of the letterforms. Shapes help give the typeface a more modern look. Geometric fonts gained popularity in the 1920s during the Bauhaus movement [71], which sought to reunite creativity with manufacturing. As a result, geometric fonts have characteristics that are influenced by architectural and machine design. Futura, Avant Garde, and Pump Triline are some of the most popular examples. Pump Triline is a font that popped up around 1970 and started to gain traction as it increased in popularity during the 80s. As such, when it's seen in modern designs, it reminds users of the 80s era because of its heavy usage in disco posters and nightclubs.

Humanist sans-serif fonts are characterized by their stark contrast in the alternation of thin and thick strokes. Humanist fonts have almost the same characteristics as slab-serif fonts, however, the serifs in humanist fonts are more so elongated strokes that dangle at the baseline of the character. The slight calligraphic influence of humanist fonts gives them a somewhat fancy feel, almost as if to mimic that of cursive writing. This makes it a great typeface for formal writing and for display on the web with the benefit being that it's much more legible than serif fonts. Popular humanist fonts include FS Siena, Optia, and Gill Sans.

Script, monospaced, and display fonts are the final categories of fonts and are less common than serif and sans-serif fonts. Script fonts are based on man-made handwriting styles

and can often be differentiated based on the different tools they mimic, such as brushes, markers, pointed nibs. The letters are often connected to each other giving off a very formal atmosphere based on classical penmanship with a calligraphy brush. On the opposite end of the spectrum, script fonts can also be quite playful. One of the most controversial fonts, Comic Sans MS, is a script typeface designed by Microsoft in 1994 for use in informal documents and children's materials [72]. It may be hated in the typographic design community, but studies found that it is a great font for dyslexic students. Monospaced fonts are fonts whose glyphs each occupy the same amount of horizontal space. Monospaced fonts were born from the mechanical limitations of typewriters. When a key on the typewriter was pressed, the carriage moved the paper the exact same distance each time, so to keep characters consistent, a font with equal width characters was created. Monospaced fonts are also very popular in computer terminals. Due to the extreme graphical limitations of early computational devices, hardware was only capable of displaying a fixed number of pixels on the screen. As such, monospaced fonts were a great fit because they didn't need complex curves or paths in order to be displayed. Most text editors, IDEs, and terminals keep this font as the default font because of how legible it is. With monospaced fonts, the focus isn't on what the font looks like, but rather the message the text tries to convey. Display fonts are a somewhat broad category that can include all types of serif and sans-serif fonts. Display fonts are designed for short-form large-format applications, like headlines, text on billboards, posters, and magazine/book covers. Display fonts don't necessarily have a specific style attributed to them. Instead, they are made to be blown up to large proportions to exaggerate embellishments in the typeface. Popular restaurant chains are a good example of display fonts because they're visible and legible from such a large distance. Gotham, a font used by UCF's Center for Humanities and Digital Research department, was initially commissioned by magazine editors who wanted a display style sans-serif font with a geometric structure [73]. Gotham can even be seen today with its usage in Taco Bell's logo.

8.3.7 A Brief Comparison of Color Models

The Process of choosing a fitting color scheme for a design is more complicated than just choosing a color that fits the designer's taste. A study conducted by Colorcom showed that users take only about 90 seconds to make a subconscious judgment about a product and between 62% to 90% of that swift assessment is based on color [74]. In a digital age where screen time is king, the right color scheme can practically make or break your product. In order to use colors effectively, it's important to understand how colors can work together.

Color has two standard models: additive and subtractive. Additive color is what most digital consumers are used to, RGB (red, green, and blue). Additive colors produce different colors by summing the numeric representation of the component colors. By adding together red, green, and blue, we get white. For digital products, the additive color model is a fantastic choice

because it offers such a large amount of choice due to its wide color spectrum. Subtractive color is mostly used for print media. CMYK (cyan, magenta, yellow, and black) are the primary colors that represent subtractive color. In the subtractive color model, colors are produced when light passes through successive layers of partially absorbing material. This is in essence how dyes and inks work. By mixing cyan, magenta, and yellow, we can obtain black. Because of how the colors are produced, the subtractive color model isn't particularly useful in web design.

When represented in CSS, color can be written in a number of ways, most commonly with a 6- or 8-digit hexadecimal value (3 digits can be used as well to represent values for red, green, and blue respectively). When written with 8 digits, the last 2 digits represent that color's alpha channel, in other words, how transparent that color appears with 00 being completely transparent and ff being completely opaque. The benefit of the hexadecimal representation of color is that it's easy to transfer from one application to the next. When designing in Figma or Sketch, you can copy that hexadecimal value straight into your CSS file to achieve the same color you were designing with. The problem comes when you need to modify this color. One of the downsides of hexadecimal color representation is that it's difficult to change the shade of a color. If a client wants a blue that's a little lighter to mimic the color of the sky, you'll need to open some external application to modify that color. This is somewhat solved with the RGB/RGBA color representation which lets you modify the three color channels separately, along with the opacity channel. In the RGB color model, different amounts of red, blue, and green are specified with values ranging from 0 to 255, with opacity ranging from 0 (completely transparent) to 1 (completely opaque). Because RGB is additive, using 255 for all values produces white while using 0 for all values produces black. RGB somewhat solves the issue brought up by hexadecimal color values because it now becomes possible to change a color by adding or removing a certain amount of red, green, or blue, although it's still somewhat of a guessing game. HSL is the final way to represent color in CSS. HSL represents hue, saturation, and lightness and like RGB, also comes with an alpha channel when HSLA is used. HSL is unique because it kind of mimics a color wheel. The first value, hue, is a value from 0 to 360 with a unit of degrees. You can think of this as the position in a color wheel starting from the top and going clockwise. Saturation, a value from 0% to 100%, controls how much color should be present. The closer to 0%, the greyer the color, the closer to 100%, the brighter or more vibrant the color. Lightness, a value from 0% to 100%, represents how much black or white exists within the color. A lightness value of 0% means the color is completely black while a lightness value of 100% means the color is completely white. The thing that makes HSL so powerful is that it's capable of generating different shades of a color by only changing one value. This makes it a great tool for creating color palettes.

8.3.8 The Significance of Color Harmony

Color harmony refers to the arrangement of colors in a way that's aesthetically pleasing and attractive to the viewer. When certain colors are paired together, viewers can feel a sense of peace. On the opposite end, inharmonious colors clash and contrast with each other and when used incorrectly, can cause viewers to be disgusted. A proper balance of color is vital in UI design because it's one of the aspects of a design that has a large influence towards the viewer.

There are five main types of color schemes that work effectively: monochromatic, analogous, complementary, split-complementary, and triadic. By far the easiest color scheme to implement is monochromatism. A monochromatic color scheme is based on one single color with various shades applied to it. One of the reasons to use HSL in CSS is because generating a monochromatic color scheme would involve picking a hue but only changing the saturation of the color. Monochromatic color schemes are great because they're difficult to mess up and the usage of a single color bodes well with the idea of digital minimalism, especially when paired with a neo-grotesque font and a design with lots of white-space. Analogous colors are colors that are located right next to each other on the color wheel. An analogous color palette is commonly used when a design requires little to no contrast with respect to other content on the page, like images or banners for example. A complementary color scheme uses a mix of colors that oppose one another in the color wheel. This is a color scheme that was popularized in Google's Material Design philosophy. The idea was that one color is chosen to be the main primary color, while a secondary color is chosen to complement and support the primary color. If the primary color of your site was blue, you'd use yellow to emphasize actions that should stand out to the user, like a yellow button to send an email after a user has finished composing it. The split-complementary color scheme is similar to a complementary color scheme except it uses two additional colors. For example, if we chose red as our primary color, we'd need to choose two colors opposite red but adjacent to each other, like green and teal. A triadic color scheme uses three colors equidistant from one another on the color wheel. To properly balance these colors, it's recommended to choose one dominant color and use the other color as supporting colors or accents. The triadic scheme is vibrant, colorful, and tends to be enthusiastic, even if paler or more pastel colors are used.

Section 1.5 talked about the importance of designing with accessibility in mind and color harmony can play an important role in design. We've been conditioned to associate certain colors with different feelings, but what about the visually impaired? Red and green may complement each other but only for those of us who don't suffer from deuteranopia. A good color scheme should be able to convey meaning with use of contrasting colors, while also catering to the visually impaired. A common example is dark mode for most devices. Some users find it easier to consume content when there's a larger contrast between the color of the background and the color of text. Dark mode is a simple feature that can help with this.

8.3.9 The Psychological Effects of Color

Color in UI and UX design isn't just about choosing colors that look nice together, there's also an effect that certain colors have on a viewer. For instance, in medicine, red and orange pills are often used as stimulants. In marketing, popular fast-food chains often use red and yellow because studies have shown that these colors subconsciously stimulate hunger. Color even has such a large influence on the food that we consume that a lot of corporations dye food to add more appeal to it. This same effect can be used in design.

We often associate red and green with erroneousness and correctness. When you fill out a form, a red label underneath a field usually means that there was an error that needs to be corrected. When students take a quiz, an answer highlighted green usually means that was the correct answer. Designers may try to give meaning to color themselves purely based on hunches, but this isn't always the correct approach. An A/B test conducted by HubSpot [75] presented users with an action button colored either green or red. It may have been predicted that green would outperform red because of what we often associate green with. However, the study showed that the red button outperformed the green button by 21%. HubSpot theorized that users associated red as an urgent action, something that required their immediate attention.

In terms of accessibility, blue is usually the safest color. Types of colorblindness that involve trouble seeing shades of blue happen to be the rarest. In fact, nearly everyone can distinguish blue from any other shades because of how blue contrasts with colors around it. Purple, a color close to blue, is often disliked by men, but favored by women [76]. Purple is often associated with royalty, luxury, and arrogance because of how difficult it was to obtain purple pigments in the mid 16th to early 17th century. Color can be a bit of a tricky subject because a lot of the study is subjective. There's never one right answer so it can sometimes be difficult to justify why one single color palette should be used over the other.

8.4 Mobile App Research

The goal for this section is to do as much research as we can so we'll be able to implement the application with ease when we start the development phase of the project. I am going to try to answer/learn how to connect react native apps to Apache Server and Database? How to make a barcode scanner on an app? How to make API calls on react native? How to test and export React Native apps to make it work on IOS (Apple) devices?

8.4.1 Expo

Expo is a very useful platform and framework for universal React applications. It is a set of tools and services built around React Native. It helps users develop, build, deploy and quickly iterate on Android, IOS and web apps from the same JavaScript/TypeScript codebase.

8.4.1.1 History

Exponent, later known as Expo, was founded by James Ide and Charles Cheever in October 2013. Their goal was to make developing mobile software easier. The company started to do research and prototypes using JavaScript to develop react mobile apps. This was uncommon in 2013.

After two years of hard work (2015) they released the Expo development client for the IPhone. It was a wild success since react native IPhone was released later that year. In December of that year, Expo released its development client for Android right after React Native released React Native Android.

In the Summer of 2016 Exponent participated in the Y Combinator. During that year the source code to the Expo client apps was released as open source. This led to the client getting popular and more features including WebGL compatible graphics and more.

Figure 8.4.1.1 - Exponent logo before 2017 to Expo

Expo was getting bigger and brighter due to their recent achievements that they changed their name from Exponent to Expo in 2017. During the same time, Expo released Snack. Snack is an online code editor that lets users create and edit small demos that run in the Expo client app. In March of 2017 Expo and Facebook collaborated to make the first version of Create React Native App. The goal of this collaboration was to make it easier for everyday (users like me and you) to get started creating React Native apps.

Two years later, in March 2019, Web support for Expo was released . This lets web developers create web applications from their Expo projects. So in 2019, web developers can

create web applications, iPhone and Mobile apps all in the same place. From there Expo grew into what it is today.

The Expo team is headquartered in Palo Alto, California with developers worldwide and develops the Expo platform. The team provides developer tools and services, contributes to related open-source projects such as React Native, and is active in front-end technology communities and conferences.

8.4.1.2 Installation

We are going to use Expo for our app. We will need to install some dependencies including Node.js and GIT. We can get Node.js from https://nodejs.org/en/ and GIT from https://git-scm.com/ . If there's a mac user he will have to download Watchman as well from https://facebook.github.io/watchman/docs/install#buildinstall.

After this we need to download some tools. First, we need an editor to edit in. We can use VScode, you can download it from https://code.visualstudio.com/download and download the Expo extensions from https://marketplace.visualstudio.com/items?itemName=byCedric.vscode-expo. We can use either Powershell, VScode terminal or WSL to run programs.

Step 1: Open Node.js and install npm

Step 2: Download Expo CLI using "npm install --global expo-cli" Use "expo whoami" to verify you downloaded.

Step 3: Make an expo account

Step 4: Open the App store or google play store and download the Expo app.

Step 5: Login and you'll be set to go.

These are the general steps to get started with expo. This will be useful when we start developing our app.

8.4.1.3 Camera

Camera is an Expo tool that is very helpful in image capturing, and barcode scanning. It is a component that renders the back and front camera. Some of the Camera's parameters include: zoom, autofocus, white balance and flash mode. With a camera we can take pictures and record video. The camera component can also detect faces, and barcodes. This is helpful for our barcode scanner and image processing requirements. The Camera component is compatible with Android, IOS devices as well as Web as shown below.

Android Device	Android Emulator	iOS Device	iOS Simulator	Web
✓	✗	✓	✗	✓

Figure 8.4.1.3 - Camera Compatibility

In order to use this tool we have to open Node.js and download it by typing “expo install expo-camera” and import it as a component in the beginning of the file.

Camera has some great static methods that are going to be very helpful in our project. This includes: requestPermissionAsync, flashmode, autofocus, zoom, onMountError, onBarCodeScanned, takePictureAsync, and more.

Camera.requestPermissionAsync(): Asks the user for camera permission

Camera.getCameraPermissionsAsync(): Checks user’s permission for accessing the Camera.

flashMode: This feature will help people who are visually impaired/ have visual difficulty. We can set the flash mode by doing Camera.Constants.FlashMode, the default is Off.

Zoom: We will allow people to be able to take pictures while the camera is zoomed in. This way it’ll be compatible with users who have vision loss. It will return a float value between 0 and 1. The default is 0.

AutoFocus: This feature will help users take clearer pictures. We will use Camera.Constants.autoFocus.

onMountError: Most users hate when they don’t know what the problem is when an app is not working. This method will let the user know if the camera’s preview is not started by sending an error message.

onBarCodeScanned: When a barcode is scanned successfully, an object is returned that has the type and data of the barcode. This is helpful so I can send the data to API/Database to check if the type is correct.

8.4.2 React Native

React Native is a JavaScript framework for writing natively rendering mobile apps for IOS and Android. It’s based on Facebook’s JavaScript library for building user interfaces, but instead of aiming at the web, it targets mobile platforms. In other words: software developers can

now write mobile apps that look and feel truly “native,” all from the comfort of a JavaScript library that we already know and love. Plus, because most of the code you write can be shared between platforms, React Native makes it simple to simultaneously develop for both Android and IOS.

Like React for the Web, React Native applications are written using a combination of JavaScript and XML-esque markup, known as JSX. Then, under the hood, React Native initiates the native rendering APIs in Objective-C (for iOS) or Java (for Android). Thus, your application will render using real mobile UI components, not webviews, and will look and feel like any other mobile application. React Native also exposes JavaScript interfaces for platform APIs, so your React Native apps can access platform features like the user’s location or a mobile camera [78].

8.4.3 How to connect a React Native app to Apache Server?

In order for us to answer this question, we first have to make a React Native app. We have to download node.js, and npm. Then we build the app by following the steps below.

Step 1: Open node.js and type npm install -g create -react-app then create-react-app (name of app).

Optional step: Set a homepage for the app. Go to package.json and add “homepage":"https://antonellozanini.com/test" under the name field. The website could be whatever you want

Step 2: Build app by typing cd (app name) then npm run build.

Step 3: Copy and paste the build folder and add it to the Apache server.

Step 4: Make a “.htaccess” file in the html directory that contains Options -MultiViews

RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f

RewriteRule ^ index.html [QSA,L][79] in it.

8.4.4 How to make a barcode scanner in React Native?

After doing extensive research, I found that React has an already made library for barcode scanners. It has one for npm and expo depending on whichever one I want to use. This will help us later during the development phase.

We will first talk about npm barcode scanner implementation. Npm barcode scanner support these types of barcode scanners including: UPC E, CODE 39, CODE 39 MOD 43, EAN 13, EAN 8, CODE 93, CODE 138, PDF 417, QR, and AZTEC [77]. After creating the react native app we have to follow these steps to make it work.

Step 1: Open node.js and install the camera by doing npm install react-native-camera –save.

Step 2: Link the camera to our react native app by typing react-native link react-native-camera.

Step 3: We will make a barcode.js file and add barcode functionality

Step 4: In our app.js we will import camera and export barcode by doing

"import React, { Component } from 'react';

import {Text, View, StyleSheet, Alert, TouchableOpacity, Image} from 'react-native';

import Camera from 'react-native-camera';
export default class BarcodeScan extends Component {…}"

The expo version of the barcode scanner is very similar but less code. Expo supports both IOS and Android barcode scanning. Expo supports IOS barcode scanner types including: Aztec, Code 39, Code 93, Code 128, Code 39 Mod 43, datamatrix, Ean 13, Ean 8, Interleaved 2of5, Itf14, PDF417, UPC_E, and QR. Expo also supports Android barcode types including but not limited to: Aztec, Codabar, Code39, Code93, code128, Datamatrix, Ean13, Ean8, Itf14, Maxicode, PDF417, RSS14, RSSExpanded, UPC_A, UPC_E, UPC_Ean, and QR. Now let's implement the Expo barcode scanner. After making the app.

Step 1: Open node.js, go to the app directory and install barcode scanner by typing expo install expo-barcode-scanner.

Step 2: Import this important API to ask for camera permission from the user by typing "import { BarCodeScanner } from 'expo-barcode-scanner';" in app.js

Step 3: Follow the rest of the steps in https://docs.expo.dev/versions/latest/sdk/bar-code-scanner/#supported-formats

Step 4: Change the type of barcode scanning you want to do to reduce battery usage by typing the type of barcode scanning you want to do in barcodeTypes variable definition.

There are a couple differences and similarities between doing expo and npm barcode scanners. One difference between Expo and Npm is that Expo is known to be compatible with

both IOS and Android while I am not sure that npm is. Another is that Expo requires less coding and supports more barcode types than npm. Npm has an integer variable that I can send to API for them to analyze and have them check-out the item/look it up in the system. I am also more familiar with npm and I understand the code more.

8.4.5 How to test and export react native to work on IOS?

We need to know how to test and export a react native app to work on IOS because one of our requirements is for our app. Our project requirements request us to make an App to work on both Android and IOS devices.

After doing research I found out that we can test on IOS devices by doing npm run ios. This will run the IOS simulator which will let us see how our react native app will look after making changes. I will be using https://blog.logrocket.com/how-to-build-ios-apps-using-react-native/ as a reference. There's also a version of Expo for IOS that we can use to test it on our own devices as well.

After we finish the app, we want our IOS users to be able to use it. We will export the react native app into a .apk using a website. Then we will send it to the sponsor which will then send it to all the employees.

8.5 Image Processing

8.5.1 Why Image Recognition?

At the request of our sponsors, we were tasked with recognizing the covers of journals in place of barcodes, so as to avoid labeling hundreds and thousands of individual journals with unique barcodes. Because we will be using Flask as our framework, the image recognition will probably be done in Python. Luckily for us, there's an open source machine learning framework for Python called TensorFlow [80]. It doesn't cost a lot of resources to use and it's well maintained as it was originally developed by Google. TensorFlow is also available as a JavaScript library called TensorFlow.js, and as a mobile framework under the name TensorFlow Lite.

8.5.2 Machine Learning Summary

Machine learning is a subfield of artificial intelligence, and a branch of computer science that focuses on the use of data and algorithms in such a way as to imitate the ways that humans learn. Through this process, a computer can be trained to do almost anything, whether that be recognizing faces or picking out objectively cute dog pictures. Through the process of training, the machine learning algorithm attempts to become more accurate at what it is trying to do, and the longer it trains, the more accurate it should be. “Should " is a keyword, because if trained improperly, the machine learning algorithm could end up spitting out garbage results and be completely wrong. Just like a dog must be trained well to perform the desired actions, so too must the machine learning algorithm be trained well to perform the actions desired by the user.

Machine learning generally follows a three step process [81]:

1. Predicting
    - Algorithm reads data and returns a prediction
2. Evaluating
    - Accuracy function measures how good the prediction was by comparing to known examples (provided by the user) and returns a quantity representing how good or bad the guess was
3. Optimizing
    - Algorithm looks at the guess and updates the prediction process to reduce the severity of future bad predictions

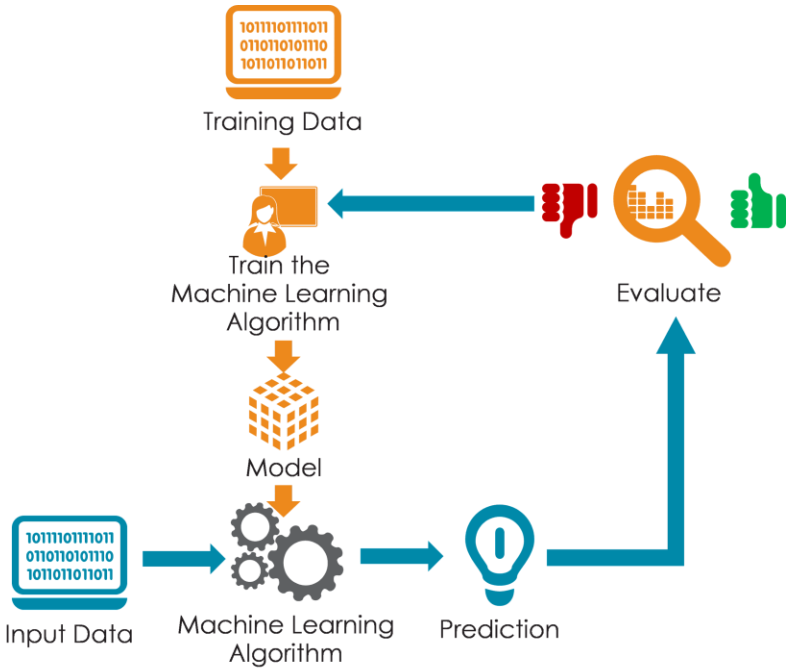


Figure 8.6.2.1 - Image taken from dev.to in an introduction to machine learning article

Just as an example, if your algorithm is trying to pick out good movies for you to watch, it might look at a movie and see how similar it is to movies you've watched previously and come up with a weighing system to grade how good a movie is based on that comparison. During training, the algorithm will go through your watch history and weigh different properties of the movies (genre, runtime, demographic). Then it will test itself by picking a movie based on those weights it generated. If it makes a bad choice, it will turn down the weights that led to the choice so it doesn't make that decision again. If it makes a good choice, It will leave the weights alone, or may even turn up the weights that led to the decision to make it more likely in the future.

This process happens automatically after the algorithm analyzes the data you gave it initially. Once it's done analyzing the initial data, it will go off on its own and analyze its own results and make improvements as it sees fit, becoming more accurate with each iteration. Human intervention is not required once it starts, and you can leave it unsupervised.

8.5.3 Methods of Machine Learning

- Supervised Learning
    - Dataset being used is pre-labeled and classified by the users to allow the algorithm to see how accurate its own performance is
- Unsupervised Learning

    - The raw dataset being used is unlabeled and the algorithm identifies patterns and relationships within the data without help from the users
- Semi-supervised Learning
    - The dataset contains structured and unstructured data which guide the algorithm on its way to making independent conclusions. You can consider this the equivalent of training wheels that you would find on a children's bicycle. The combination of the two data types in one training dataset allows machine learning algorithms to learn to label unlabeled data.
- Reinforcement Learning
    - The dataset uses a "rewards/punishments" system, offering feedback to the algorithm to learn from its own experiences by trial and error [82]. Similar to how dogs respond to rewards in the form of treats, you can do the same with a machine learning algorithm by giving it a "goal" of higher accuracy. The more accurate, the better it is doing.
- Deep Learning
    - This is a newer area of machine learning that automatically learns from datasets without human rules or knowledge. The issue is that this requires a massive amount of raw data for the model to be reliably accurate. But, given a large enough dataset, it can be even more accurate than the above described models.

8.5.4 Image Recognition Summary

Image recognition is the ability of a computer powered camera to identify and detect objects or features in a digital image or video [83]. It is a method of capturing, processing, and examining images. To do this, computers use machine vision technology powered by an artificial intelligence system (our aforementioned machine learning algorithm). Different image recognition algorithms include face recognition, license plate matching, scene identification, and optical character recognition.

Image recognition takes an image as input and outputs what the image contains. It works by finding the portions of the image that contain the most information about the image. It will isolate those informative portions of the image and ignore the rest. For an image recognition algorithm to know what an image contains, it must have been trained to learn the differences between the images we are giving it. For instance, in our application, the algorithm must be able to recognize the differences between different journal covers, and correctly identify the journal cover whose image we are providing. This is done by providing it with a collection of images containing journal covers, and images that do not contain journal covers.

8.5.5 Machine Learning Image Recognition Models

Note: These do not include deep learning algorithms as we do not have access to the massive raw data sets required to train a reasonably powerful image recognition algorithm.

- Support Vector Machines
    - SVMs work by making histograms of images containing the target objects and also of images that don't. The algorithm then takes the test picture and compares the trained histogram values with the ones of various parts of the picture to check for matches.
- Bag of Features Models
    - Bag of Features Models like Scale Invariant Feature Transformations (SIFT) and Maximally Stable Extremal Regions (MSER) work by taking the image to be scanned and a sample photo of the object to be found as reference. It then tries to pixel match the features from the sample photo to various parts of the target image to see if matches are found.
- Viola-Jones Algorithms
    - A widely-used facial recognition algorithm from pre-CNN (Convolutional Neural Network) times, Viola-Jones works by scanning faces and extracting features that are then passed through a boosting classifier. This, in turn, generates a number of boosted classifiers that are used to check test images. For a successful match to be found, a test image must generate a positive result from each of these classifiers.
    - This could be generalized to extract specific features of our journal covers as classifiers that it compares other journal cover features to.

8.6 Mobile View

8.6.1 Mobile Web Design

People don't only visit websites on their personal computers. They have access to the internet and your websites from computers, phones, tablets, and at this point, even their smart fridges. Designing a website solely for access on a desktop computer is not only lazy, but detrimental to your user experience. As such, we will be designing for desktop access, and for mobile and tablet access as well. There are two main methods by which one can design a website for multiple platforms. They are Adaptive and Responsive web design [84]. The following sections will elaborate on what each of these methods mean.

8.6.1.1 Responsive Web Design

Responsive Web Design, also called RWD, is a design approach that allows websites and pages to render on all devices and screen sizes by automatically adapting to the screen's size, regardless of platform (desktop, laptop, tablet, smartphone, etc.). Generally speaking, RWD uses CSS to change the dimensions and placement of objects in the website depending on different properties like screen size, orientation, resolution, color capabilities, etc. This can include conditional page formatting, like rearranging resources on-screen if the size of the display is within a certain limit.

Figure 8.6.1.1.1 - Responsive Web Design [85]

RWD is becoming the go-to approach for designing websites because of its flexibility. One does not have to explicitly detail the format of the website for each device, one merely has to define the behavior of the site given conditions of your choosing. It allows you to provide a consistent experience to users, and you only have to maintain a single website. As such, I believe this design approach would be the correct choice for us in developing our product.

8.6.1.2 Adaptive Web Design

Adaptive Web Design is a design approach that uses explicitly defined layouts for devices. This means that for each device used to view your site, it will display a different layout/design. This can be useful for websites that are dependent on load times, as loading a

specific view based solely on what device the user has is far quicker than having to calculate the size and position of every element based on screen size. However, our website will not be very resource intensive, so loading times should not be of concern to us. Just as well, you will have to maintain each view separately as if it were its own website, so upkeep costs may be higher.

Figure 8.6.1.2.1 - Adaptive Web Design [85]

That being said, even if you can define views per device type, you'd be hard pressed to find a one-size-fits-all solution using Adaptive Web Design. Even if you can account for the type of device being used, there are hundreds of different models of smartphones, all with their own dimensions. Not all phones are the same size, and as such, you can only pick the layout that will fit the *best*, not a layout that will fit perfectly.

8.6.2 Key Practices for Mobile Web Design

For the longest time, the desktop was the only method by which people could access the internet. Nowadays, time is money, and people will often use their phones to access information while on the go, or while they're busy. Building your website for mobile access is one of the most important things you can do to ensure quick and easy access to whatever information your users need. Accessibility and Simplicity are the two main points that the following sections will discuss [86].

8.6.2.1 Accessibility

Ensuring that a website is accessible is not much of a concern when designing for desktop. Everybody uses the same keyboard and mouse controls to interact with the site, and there isn't much variance from that norm. On mobile, however, the primary mode of interaction is touch. Accommodating touch controls requires a lot more finesse than mouse/keyboard. This is because mouse cursors are very precise in their placement, peoples' fingers are all different shapes and sizes, apply various amounts of pressure, and the touchscreens being used are all calibrated differently as well.

Elements on the page that are typically small on the desktop are not viable anymore. The chances that the user's fingers will tap and overlap onto other nearby elements are very high, so the larger you can reasonably go, the better. This includes buttons, links, and forms, so make sure you don't forget anything. Another note on forms, try to minimize usage of forms on the mobile site. Screen space is already hard to come by on a mobile device, and having to pull up the keyboard to type reduces that even further. Not to mention, filling forms on a mobile device is a lot more difficult and slow than it would be on a desktop. It's not a good user experience, so try to avoid it if you can.

Lastly, don't limit your thinking to how you would on a desktop computer. Since you're designing for mobile access, consider how you might leverage the capabilities of the mobile device. They have access to tools like GPS and phone calls. Instead of simply listing the phone number of a business in the mobile view, give the functionality of tapping on it and taking you to the phone app so you can call directly. Instead of listing the address of a business in the mobile view, link it to the user's preferred GPS app on their phone and have it automatically show navigation to the location. For the user, this is much easier than trying to copy the information from the page back and forth to the app they want to use, and makes the experience much more enjoyable.

8.6.2.2 Simplicity

On a desktop, you have all the screen space in the world to stuff information into. On mobile, screen space is worth its weight in gold. Instead of trying to miniaturize the desktop page, you should be trying to mobilize the information on the page. Don't just make everything smaller and more compact to fit a phone screen; you should be reformatting everything so that the most important information is easily accessible. You should be top-loading your webpage, meaning you should place all of the most crucial information and functionality of your mobile webpage at the top so that it's the first the user sees. Also, avoid multi-level menus as much as

possible on mobile sites. Trying to tap through those is already difficult enough, just send the user to another page that holds the relevant information to save space.

When designing for mobile, not only should you be making the information easy to access, you should also be removing any and all unnecessary clutter from the screen. Mobile users are on mobile most often because they need the information quickly. Having obnoxious colors, images, and ads on the screen will only prevent them from finding what they need in a timely manner. Follow the KISS (Keep It Simple Stupid) method and everything will be fine. Remove anything that could confuse the user and keep it as simple as possible. This means fun CSS effects like gradients or shadows, company logos and designs, and fancy text fonts.

8.7 Secure Password Storage

When building an application, especially when handling sensitive information like student IDs and passwords, it is important that bad actors aren't able to gain access to said information. When passwords are stored in the database as plaintext, if there is a data breach, attackers can retrieve all user information, including passwords, compromising everything. Therefore, it is imperative that developers somehow secure their users' passwords so as not to compromise their information. This can be done in a number of ways.

8.7.1 Hashing

8.7.1.1 Straight Hashing

Hashing is a process where, given an input, it will output a string of a set length. If you feed the hashing algorithm a password, it will spit out the hash of that password. Hashing functions are also known as "one-way functions" because it is relatively easy to find the hash of a given plaintext, but it is prohibitively difficult to find the plaintext of a given hash, even if you know what hashing algorithm created it. This method of securing passwords is far better than simply storing the plaintext, but there's still an issue.

Even if you store the hashed passwords in your database instead of plaintext, it can still be easy to determine the actual passwords related to those hashes. If an attacker knows what hashing algorithm you're using, they can calculate the hashes of popular passwords ahead of time and correlate that list of hashes with the hashes they found in your database. This is because the hash of a string never changes (the hash of "password" will be the same no matter how many

times you compute it). This method of attack is known as an "offline dictionary attack", and is rather common [87]. So, something better would be preferable to truly secure our users' data.

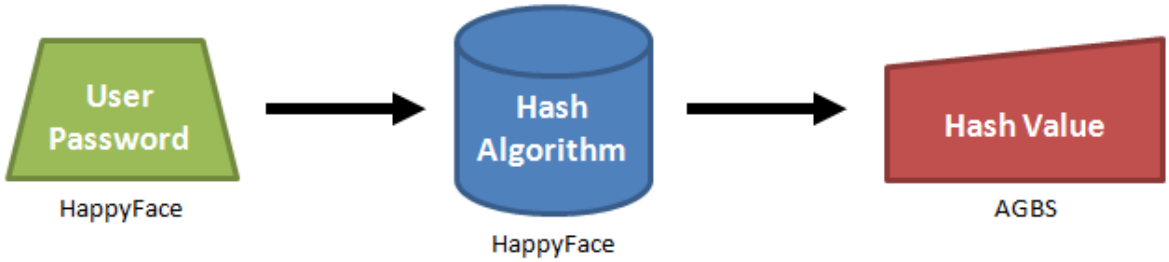


Image 8.7.1.1.1: Password to Hash [88]

8.7.1.2 Salting

The next logical step would be to address the problem of identical passwords resulting in identical hashes. This can be done with what are called "salts". A salt is a long, random string that is unique for each user. This salt is appended/prepended to the password at hashing, so the resulting password that is stored in the database is equivalent to: *stored = hash(password + salt)*. This per-user salt is then stored alongside the hashed password in the database. Because the salts are unique per-user, even if two users have the same password, their stored password hashes will be different, meaning that offline dictionary attacks aren't as viable.

However, there's another problem. Let's assume that there's another data breach and our bad actor has received the contents of the user database. They have access to the password hash and to the users' salts. What they can do is append/prepend that salt to their collection of popular passwords and run another offline dictionary attack using those pre-salted passwords. This certainly takes a lot more time than before, as the bad actor has to modify their password list with a unique salt for each user they want to break into, but it's still not very secure.

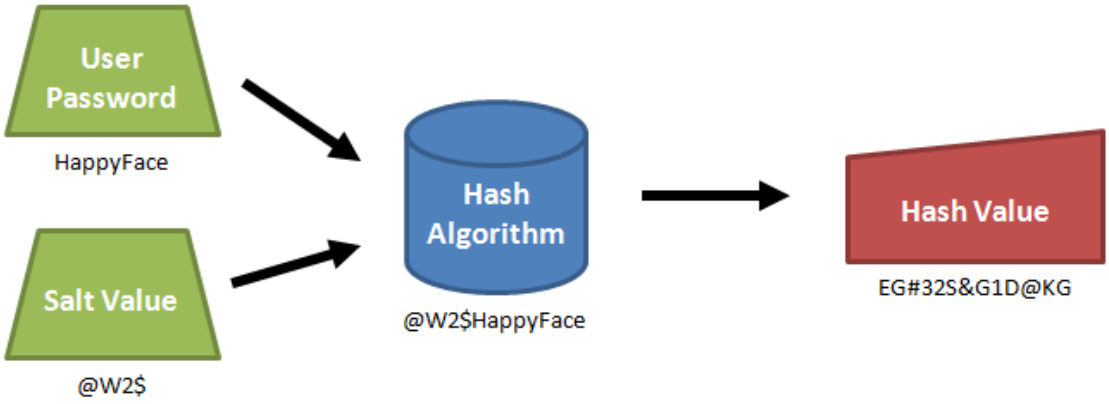


Image 8.7.1.2.1: Password + Salt to Hash [89]

8.7.1.3 Slow Hashing and Multi-Hashing

The difference between your typical hashing algorithms like MD5 and SHA-1 and slow hashing algorithms like bcrypt, scrypt, and PBKDF2 is the amount of time it takes to calculate the hash. Slow hashing algorithms are intentionally designed to take a (relatively) long time to calculate the hashes. Unless the attackers have a couple million years to spend calculating hashes to salted passwords, even if they do steal the password database, they're not going to get much use out of it.

Additionally, you can use multiple hashing algorithms in series to make it even more difficult to work out the original. Mozilla suggests the use of HMAC and bcrypt in conjunction as a reasonably secure password storing method. First, they hash with HMAC and a per-user local salt stored on the server (not in the database). Then they run the HMAC hash through bcrypt with another per-user salt. This is the final string that you will store in the database as the password. This process will be used every time you want to authenticate the user against their database credentials.

In the end, we decided to go with the bcrypt hashing algorithm alongside a per-user salt. We felt that this would put us far enough ahead of the game in terms of password safety, especially considering the huge number of services that use straight hashing with common algos like SHA or MD5.

8.7.2 Secure User Passwords

Just as it is important for the service provider to secure your information with password protection, it is equally as important for you, the user, to ensure that your password is not easily

broken as well. In fact, a lot of popular sites and services have a minimum requirement as to what your password needs to contain in order to even use it for your account. We as humans do not have perfect memory, so we often use some sort of convention to make our passwords memorable. The problem is that this can make the passwords easy to guess. We as humans are also limited in how much information we can remember, so our passwords can't be too long either. As such, it is important that we find a balance between our ability to remember our passwords, and the protection that each password provides our information [90].

8.7.2.1 Four Random Words

This method of creating a password is probably the simplest one, and yet it is relatively secure. All you need to do is take four seemingly random words and put them together to form your password. Even so, it is recommended that you take other steps to ensure your password is more difficult to guess:

1. The password should be at least 12 characters long
2. The words should not have a natural flow to them
    - (such as *My name is Richard*)
3. You should separate words with spaces or other special characters

Here are some examples:

- Gold Kabab Zenith Prosciutto
- Phoenix'Drive"Cafe;Office
- Seattle, Kindle, Coffee, Planes

8.7.2.2 Phrases

If you have a hard time remembering four random words, then consider taking a long phrase as your password. These can be exceedingly long, and as such, will be difficult to brute force. However, make sure you don't choose a famous quote or any other sort of well-known phrase, as those can be relatively easily guessed. You can also mix capitalization, spacing, and mixing in odd characters between the words:

- itsoveranakinihavethehighground
- I Reject Your Reality And Substitute My Own
- myMilkshakeBringsallTheboysTotheYard

8.7.2.3 Acronyms

If you feel those are too easy to remember, consider taking a longer phrase and turning it into an acronym. Take the first letter of each word in the phrase and make that your password. You can also replace letters with numbers or other special characters if you feel you can remember them. Given a long enough phrase, this can be more secure than the previous example as it introduces randomness into your password, along with length, making it more difficult to guess. However, the same rules apply; if you choose a well known phrase, it can be more easily guessed:

- DyehtToDPtw?
    - Did you ever hear the Tragedy of Darth Plagueis the wise?
- Iafyds,daetyt.
    - If at first you don't succeed, destroy all evidence that you tried.
- Ttootws,ylylo.
    - To the owner of the white sedan, you left your lights on.

8.7.2.4 Alternate Keyboard Layouts

If you really just cannot remember the four words, or can't figure out the acronyms, then consider using an alternate keyboard layout for inputting your passwords. Go into your settings and switch your language from English US (QWERTY) to English US (DVORAK) and watch your OCD go crazy. Other keyboard layouts include Colemak and Alphabetical:

- Oyayg. ru Ncx.pyf
    - Statue of Liberty (DVORAK)
- Yd. nrpe ru yd. Pcbio
    - The Lord of the Rings
- Ocmrb abe Iapugbt.n
    - Simon and Garfunkel

8.7.2.5 Basic Ciphers

You can use ciphers to turn your plaintext password into an unreadable jumbled mess of characters. Basic ones include Caesar and Vigenere ciphers. You can use these to impart your own sort of encryption into your passwords:

- Slfnohv duh glvjxvwlqj
    - Pickles are disgusting
    - Caesar Cipher - Shift 3
- dlgc mq k zgqilovc mmnrip
    - this is a vigenere cipher
    - Vigenere Cipher - Key = "key"

8.7.2.6 Miscellaneous Methods

This section encompasses a wide variety of possible methods.

- IwiIhamotitothofbepa
    - I wish I had more time to think of better passwords
    - Take the first two letters of every word
- llyedayshrysion
    - I really do hate Mondays with a fiery passion
    - Remove the first 3 letters of every word
- can mex fra deu jpn
    - Canada, Mexico, Francy, Germany, Japan
    - ISO Codes of countries
        - Could also use airport codes for this

8.7.2.7 Password Managers

If you don't fancy remembering passwords at all, you can use a password manager to do it for you. A password manager is essentially a vault that stores all of your online credentials [91]. Many password managers also have the capability to generate passwords for you to use in new accounts that you may create. These are all stored securely in an encrypted database with a master password that you must, unfortunately, remember. However, in exchange for having to remember one password, you have a computer program that remembers the unique (and relatively unbreakable) password for each of your online accounts. They're very powerful, very useful, and take a lot of the weight off of the user for their online accounts.

However, the same principle applies to your master password that secures your vault. If bad actors gain access to that, then you're out of luck. So, be sure to apply the above methods (and enable two-factor authentication for your manager) when creating your master pass.

9 Project Design

9.1 Database

9.1.1 Database Design Summary

The application's database is to be hosted on a MySQL server which is accessible through PHPMyAdmin. Our database design must fulfill the requirements listed for check-in, check-out, inventorying, and reservations. As such, the tables and relationships between those tables have been designed to reflect these requirements as closely as possible.

9.1.2 Database Design Requirements

1. Database has a flag indicating that an item can be checked out
2. Shows admins who has an item checked out
3. Shows users if a moveable item is available
4. Holds email records
5. Holds reservation information
6. Holds user information and their role in the system
7. Reservations can be approved by an administrator
8. Keeps records of all items
9. Allows images to be attached to an item
10. Keeps records of the quantities of a particular item type
11. Keeps records of the location of the item
12. Keeps records of the barcode identification of the item

9.1.2.1 Requirement 1

This is accomplished by placing a "moveable" flag in the individual item table. The "moveable" flag will indicate if the item stays in the facility or if the item can be taken out of the facility. An example of a moveable item would be a virtual reality headset, and an example of a non-moveable item would be a study room.

9.1.2.2 Requirement 2

This is accomplished through the reservation table, as administrators, through the front end, will be able to view the most recent reservation of the item. If the item is currently checked out, then the item belongs to whoever the most recent approved reservation is. For example, if a user with ID 25 checked out an item with ID 4 on date 1/2/2022, then a search of the reservation table for item 4 would show an approved reservation under user ID 25. The reservation table doubles for both reserving items in advance and actually checking them out. If a user fails to show up for their reservation, the front-end can send a signal to update the status of the reservation.

9.1.2.3 Requirement 3

This is accomplished, again, through the reservation table. If a user requests an item that falls within the window of an on-going, approved reservation, then the front-end can tell the user that the item is not available.

9.1.2.4 Requirement 4

This is accomplished through use of an email field in the user table.

9.1.2.5 Requirement 5

This is accomplished through the reservation table. The reservation table will hold references to the item that is reserved, the user who reserved it, the admin who approved it, the start and end dates, whether or not it is approved, and the status of the reservation.

9.1.2.6 Requirement 6

This is accomplished through the user table. The user table will contain the nid, hashed password, email, and role of the user in the system.

9.1.2.7 Requirement 7

This is accomplished in the reservation table through the presence of an identification number of the administrator who approved the reservation and a boolean that marks whether or not the reservation is approved.

9.1.2.8 Requirement 8

This is accomplished through use of the items tables. The items tables are several tables that all contain information about the item such as the barcode info, whether or not the item is moveable, any children items associated with the main item (sets of items), images associated with the item, and the type and location of the item. More information can be found in subsequent sections.

9.1.2.9 Requirement 9

This is accomplished through the itemImage table. The itemImage table will contain a path to the location of the item in a static directory. It is important to note that it is possible to store actual images in the database through use of the BLOB datatype, but this is generally considered bad practice as it can take up excessive amounts of memory in the server.

9.1.2.10 Requirement 10

This is accomplished through the quantity field in the item table. The quantity field will measure how many of that particular type of item there is. For example, if we have ten virtual reality headsets, even though each of those items will have their own barcodes and other identifying information, they will belong to the parent type of “Virtual Reality Headset,” and so, the quantity field will be ten.

9.1.2.11 Requirement 11

This is accomplished through the location field. The location field is a string which will hold the name of the cabinet in which the item is stored.

9.1.2.12 Requirement 12

This is accomplished through the ”barcode” column of the “item” table. Each individual item (or set of child items) will have a label associated with it. Therefore, this field is necessary to store that identification information. We will store the “label” field as a string, as some of the laels will have letter characters incorporated into their identification information.

9.1.3 Entity Relationship Diagram

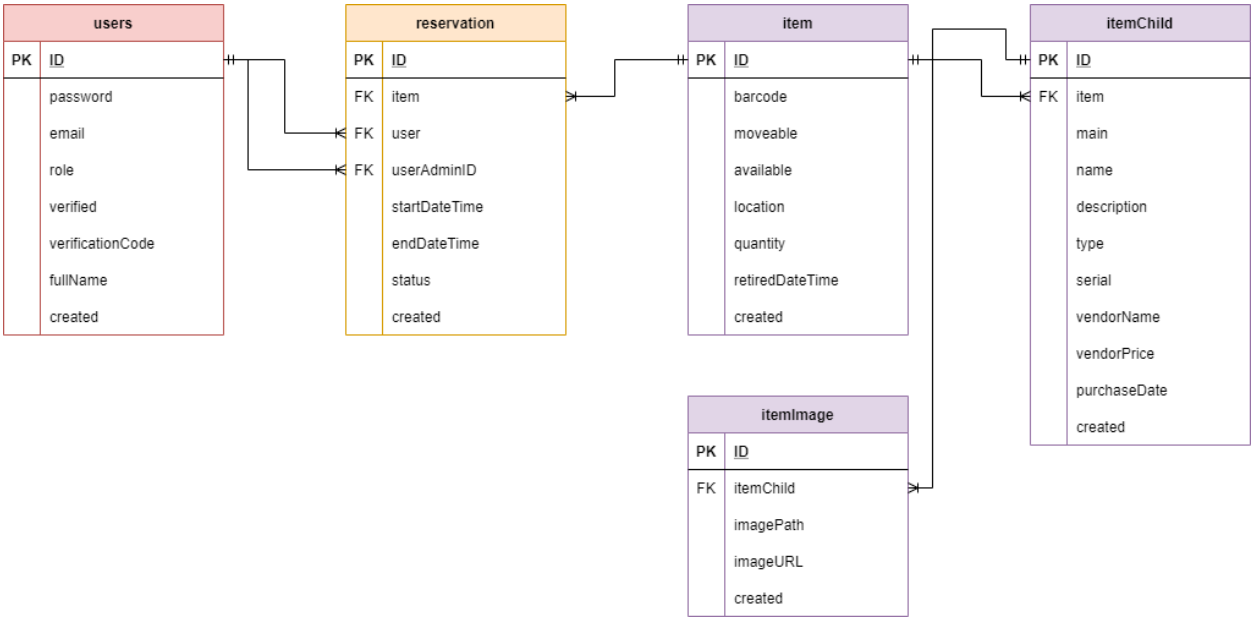


Figure 9.1.3.1: Entity Relationship Diagram

9.1.3.1 User Table

- ID; integer(8); primary key and identifier of this record
- password; varchar(128); stores the hashed password associated with the user
- email; varchar(256); stores the user's email
- role; enum of {"Super", "Admin", "User"}
- verified; bit(1); stores whether or not the user is verified
- verificationCode; varchar(36); stores the code required for a user to verify themselves
- fullName; varchar(64); stores the first and last name of a user
- created; datetime; the date of creation for this record

9.1.3.2 Reservation Table

- ID; integer(8); primary key and identifier of this record
- item; integer(8); foreign key relating to the item table
    - denotes the specific item associated with the reservation
- user; integer(8); foreign key relating to the user table

    - denotes the specific user that requested the reservation
- userAdminID; integer(8); foreign key relation to the user table
    - denotes the specific administrator that approved the record
- startDateTime; datetime; this is the start date and time of the reservation
- endDateTime; datetime; this is the end date and time of the reservation
- status; enum of {'Pending', 'Denied', 'Approved', 'Missed', 'Checked Out', 'Late', 'Returned', 'Cancelled', 'Blocked Out'}
- created; datetime; the date of creation for this record

9.1.3.3 Item Table

- ID; integer(8); primary key and identifier of this record
- barcode; varchar(32); this is the label or barcode printed on the item
- available; bit(1); this is a flag indicating if the item is available
- moveable; bit(1); this is a flag indicating if the item can be moved
- location; varchar(16); the location where this item is stored
- quantity; int(8); the number of this item that is available
- retiredDateTime; datetime; the date and time of this item being retired (taken out of circulation)
- created; datetime; the date of creation for this record

9.1.3.3 Item Child Table

- ID; integer(8); primary key and identifier of this record
- item; integer(8); foreign key relating to the item table
    - denotes the specific item associated with the child
- name; varchar(128); the name of this object
- description; text; the description of this object
- type; varchar(64); the type of this object (camera, printer, etc.)
- serial; varchar(128); the serial number of this object
- vendorName; varchar(128); the name of the vendor of this object
- vendorPrice; float(16, 2); the price of this object
- purchaseDate; date; the date of purchase for this object
- main; bit(1); whether this object is the main item in the set
- created; datetime; the date of creation for this record

9.1.3.3 Item Image Table

- ID; integer(8); primary key and identifier of this record
- itemChild; integer(8); foreign key relating to the itemChild table
    - denotes the specific item child associated with this image
- imagePath; varchar(512); the file path of this image
- imageURL; varchar(256); the URL for this image
- created; datetime; the date of creation for this record

9.2 Backend

9.2.1 Packages

Creating an application as stated prior will include many files and that is not any different from our web application that we are tasked to accomplish. That is why it is best to use packages rather than modules when you have flask as your backend framework. A module is a single file and again a larger application will include many modules. To create a package, you just include all the files you would like to have in your directory and then be sure to include an __init__.py file, while ensuring that you have no .pyc files since it causes errors to occur. [92]

```
/yourapplication
    /yourapplication
        /__init__.py
        /static
            /style.css
        /templates
            layout.html
            index.html
            login.html
            ...
```

Table 9.2.1.1: The layout of a package for simple application

Now that the application is within a package, we still need to run it and order for it to accomplish what it needs to do. To do so we must create a python file alongside your application in the package that you created, and this will allow you to run the application. We must do this since python does not allow for files in the application to be the startup file.

```
from yourapplication import app
app.run(debug=True)
```

Table 9.2.1.2: Running your package properly

9.2.2 Blueprints

Flask allows the use of blueprints to create components within the application and make patterns that make the application flow easier. Can be thought to be the same as a Flask application object, but blueprints are not applications and more or so is a feature which provides a way to build and extend an application. Through a blueprint you can provide template filters, static files, templates, and other utilities. [93] Once you have registered your blueprints to an application, you are not able to unregister the blueprint and the only way to do so is to delete the whole application object you've registered it to.

```
from flask import Blueprint, render_template, abort
from jinja2 import TemplateNotFound

simple_page = Blueprint('simple_page', __name__,
                        template_folder='templates')

@simple_page.route('/', defaults={'page': 'index'})
@simple_page.route('/<page>')
def show(page):
    try:
        return render_template('pages/%s.html' % page)
    except TemplateNotFound:
        abort(404)
```

Table 9.2.1.3: The layout of a blueprint for simple application

9.2.2.1 Static Files

When a folder includes a static file, we provide a path to a folder by using the static_folder keyword argument. This can either be absolute or relative to the folder that the blueprint is in.[93]

```
admin = Blueprint('admin', __name__, static_folder='static')
```

Table 9.2.1.4: A proper call to Static file

9.2.2.2 Templates

In order to view templates you must provide the template_folder just as you did for static files and paths can also be absolute and relative to the blueprint resource folder. However, template files will have a lower priority and therefore can be easily overridden.[93]

```
admin = Blueprint('admin', __name__, template_folder='templates')
```

Table 9.2.1.5: A proper call to a Template

9.2.2 Python Testing Frameworks

Because our web app is being built off of Flask, we'll be using Python for our backend control, including routing and API control. As such, a Python testing suite would be very useful for us to manage and run our tests for us. There is a long list of Python testing frameworks to choose from, but in the end we decided on PyTest. Here, we'll discuss the most popular that I've found:

- Robot
- Pytest
- Unittest
- DocTest
- Nose2
- Testify

Here, we will summarize the frameworks using a brief table comparing their qualities:

Testing Framework	License	Part of...	Category	Special Feature
Robot 	Free Software (ASF License)	Python generic test libraries	Acceptance Testing	Keyword-Driven testing approach
PyTest 	Free Software (MIT License)	Stand alone, allows compact test suites	Unit Testing	Special and simple class fixture for making testing easier
unittest 	Free Software (MIT License)	Part of Python standard library	Unit Testing	Fast test collection and flexible test execution
DocTest 	Free Software (MIT License)	Part of Python standard library	Unit Testing	Python Interactive Shell for command prompt and inclusive application
Nose2 	Free Software (BSD License)	Carries unittest features with additional features and plugins	unittest extension	Large number of plugins
Testify 	Free Software (ASF License)	Carries unittest and nose features with additional features and plugins	unittest extension	Test discovery enhancement

Table 9.2.2.1: (Abbreviations: **MIT** = Massachusetts Institute of Technology, **BSD** = Berkeley Software Distribution, **ASF** = Apache Software Foundation) [94]

9.2.2.1 Robot

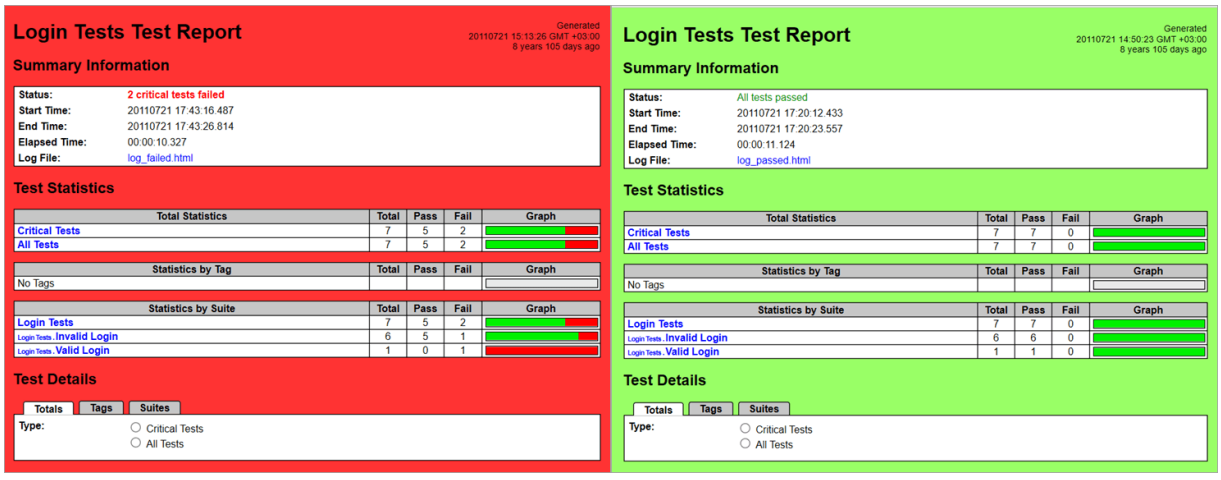
Robot is an open-source framework that is not only used for Python, but also for robotics process automation [G16, 95]. It is used for acceptance testing and general test-driven development. "Acceptance Testing" wasn't covered in the above sections, but suffice it to say that acceptance testing is commonly used interchangeably with "functional testing". It can run Java and .Net and also supports cross-platform compatibility with Windows, MacOS, and Linux. Robot is available through PIP (Package Installer for Python).

Robot's special feature is that it uses keyword-syntax for its testing. Put simply, this keyword syntax is made to resemble plain English, enhancing readability and making it as easy as possible for the testers to understand. An example of Robot keyword-syntax is given below:

```
*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Maximize Browser Window
    Set Selenium Speed    ${DELAY}
Login Page Should Be Open
    Title Should Be    Login Page

Go To Login Page
    Go To    ${LOGIN URL}
    Login Page Should Be Open
```

Image 9.2.2.1.1: Robot example feature text



Images 9.2.2.1.7 and 9.2.2.1.8: Robot Failed and Passed Testing Reports [96, 97]

9.2.2.2 PyTest

PyTest is an open-source framework based in Python that is generally all-purpose, accommodating small-scale API unit testing, but also scaling well in large-scale functional testing. [98] PyTest extends the class-functionality of Python to organize and execute tests in an organized fashion. As it is developed for Python, it uses standard Python syntax, primarily leveraging a built-in "assert" statement. It allows you to compare your expected result to the actual result and output explicitly defined failure codes based on the pass/fail result. It also has a large selection of open-source plugins to choose from (all cataloged on their site). [99] PyTest is available through PIP.

We decided to go with PyTest for our testing since it seemed to be the most easily accessible and we found it to have the most documentation when it came to use in a Flask environment.

PyTest fixtures can be used to run code before executing test methods. These are useful especially when running large sets of tests because it allows you to execute preparatory steps, such as connecting to the database, before executing all of your tests. Any variables/functions/etc. declared in the fixture (defined by `@pytest.fixture`) will be available to all tests being executed, saving lots of space in redundant code in each of your tests. Below is an example of a PyTest test:

```
import pytest                                   //Import unittest module//
def test_file1_method():                    //Function inside class//
      x=5
      y=6
      assert x+1 == y,"test failed"
```

Image 9.2.1.2.9: PyTest example code

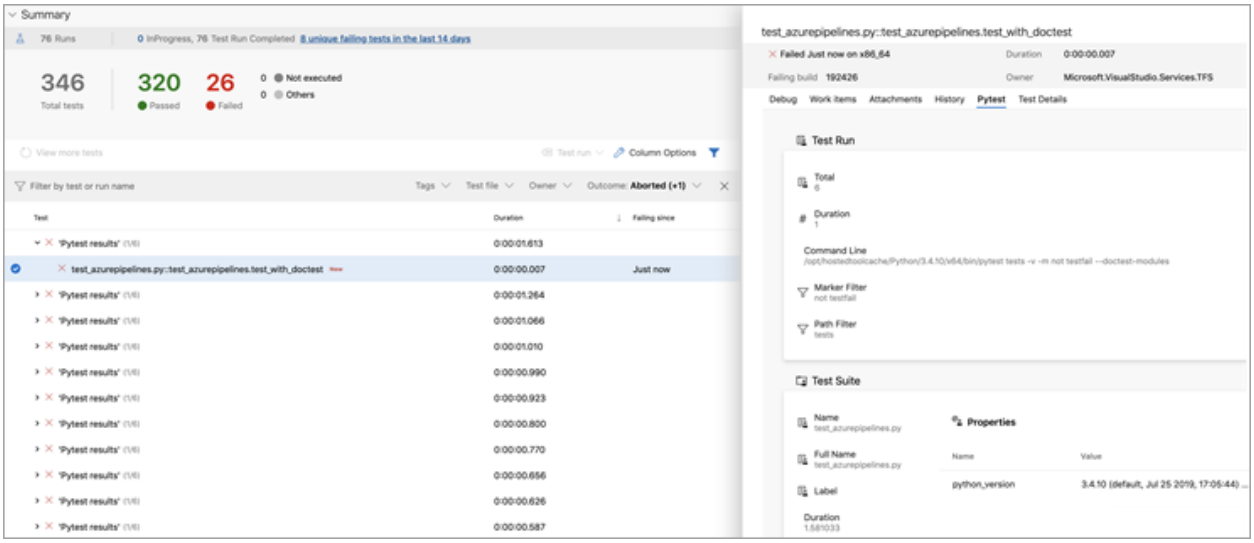


Image 9.2.2.2.1: PyTest software screenshot [100]

9.2.2.3 Unittest

Unittest (also known as PyUnit) is the first Python-based automated unit-test framework designed to work with the Python standard library, so it has a lot of followers despite its age. It was inspired by JUnit, a Java testing framework, and supports test automation, test collections, and setup code for tests (similar to PyTest's fixtures). Unittest is unique in that your tests are run alongside your code. More specifically, you place your tests inside of the same section of program code that you are testing. Below is an example of unittest code and the commandline:

```
import unittest                                    //Import unittest module//
def add(x,y):
   return x + y

class Test(unittest.TestCase):                  //Define your class with testcase//

   def addition(self):
      self.assertEquals(add(4,5),9)<strong>//Function inside class//

if __name__ == '__main__':
   unittest.main()<strong>//Insert main() method//
```

Image 9.2.2.3.1: Unittest example code

```
viniciusd at centos7-ny in ~/app on python?
☿ python testApp.py
F..             Unit Test Report
Start Time: 2015-07-27 21:01:54
Duration: 0:00:00.000858
Status:
    Pass: 2
    Failure: 1

Description:
Summary:
    +---------------------+-------+------+------+-------+
    | Test group/Test case | Count | Pass | Fail | Error |
    +---------------------+-------+------+------+-------+
    | TestStringMethods    | 3     | 2    | 1    | 0     |
    | Total                | 3     | 2    | 1    | 0     |
    +---------------------+-------+------+------+-------+
    TestStringMethods
        +-------------+-----------------------------------------------+------------+
        |  Test name  |                    Stack                      |   Status   |
        +-------------+-----------------------------------------------+------------+
        | test_isupper | Traceback (most recent call last):           | fail       |
        |              |   File "testApp.py", line 11, in test_isupper |            |
        |              |     self.assertTrue('Foo'.isupper())         |            |
        |              | AssertionError: False is not true             |            |
        | test_split   |                                               | pass       |
        | test_upper   |                                               | pass       |
        +-------------+-----------------------------------------------+------------+
```

Image 9.2.2.3.2: Unittest command line testing [101]

9.2.2.4 DocTest

DocTest is available to everybody as a module included in Python's standard library, and is used primarily for white-box unit testing, especially regression testing. White-Box unit testing (as opposed to black-box unit testing) is a form of testing where the tester knows the internal structure/design/implementation of the item being tested. [102] Regression testing is a specific type of test whose purpose is to ensure that a feature that previously functioned properly has not stopped functioning properly after a change in another section of code.

DocTest functions primarily in the Python Interactive Shell, leveraging specific integrations that allow display of current tests, expected vs resulting output, and success/failure. Below is an example of DocTest functioning in the Python Interactive Shell:

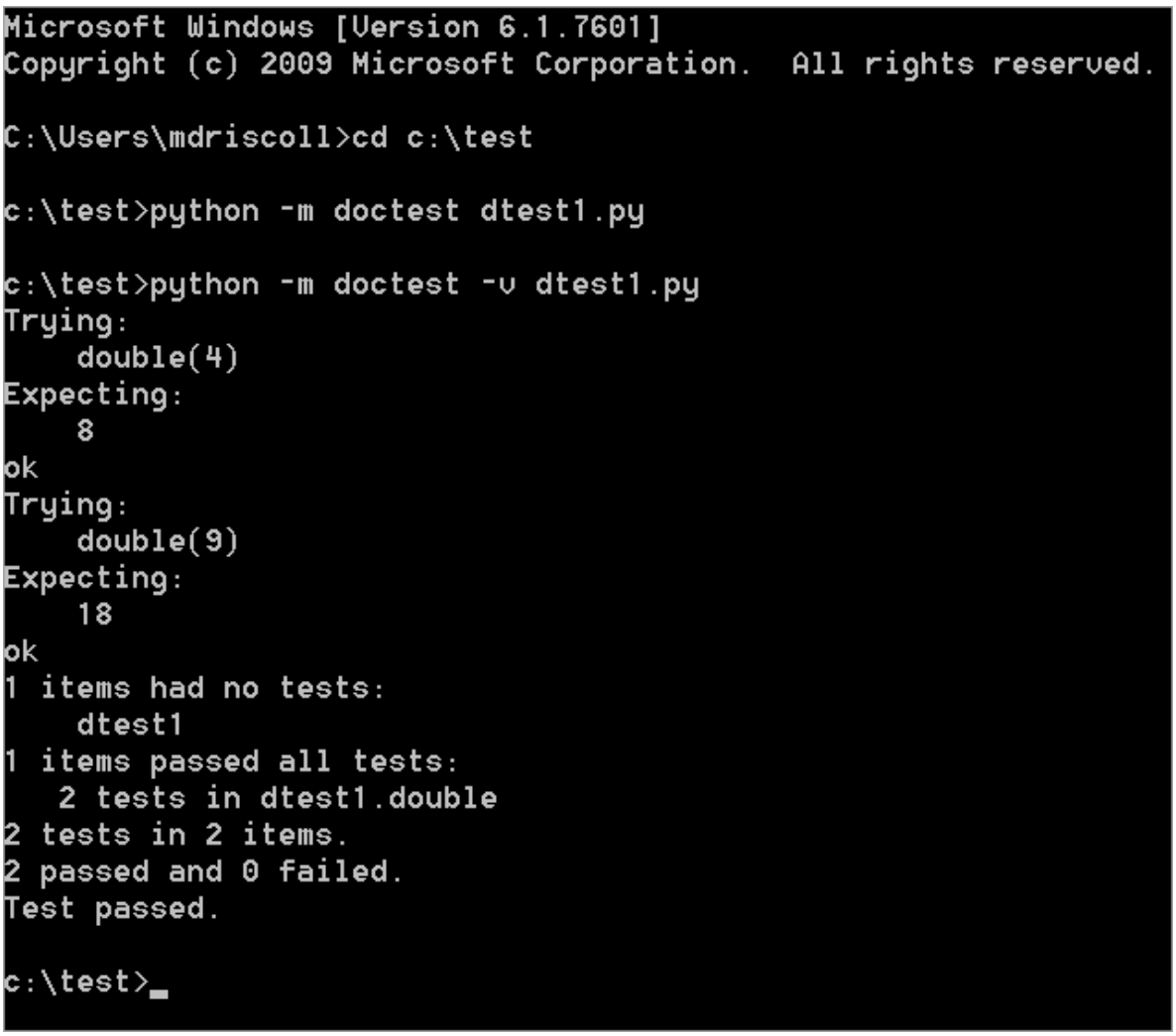


Image 9.2.2.4.1: DocTest command line testing [103]

9.2.2.5 Nose2

Nose2 is the direct successor to Nose, and is a Python-based unit testing framework. [104] Nose2 is based on unittest, hence why it's called a "unittest extension". To put it more simply, Nose2 is unittest with plugins that make it more user-friendly and easier to understand. As such, Nose2 and unittest are nearly identical, with Nose2 having some extra functionality like test generators. Nose2 supports package fixtures, classes, modules, and complex initialization at startup (saving on redundancy). Below is an example of Nose2:

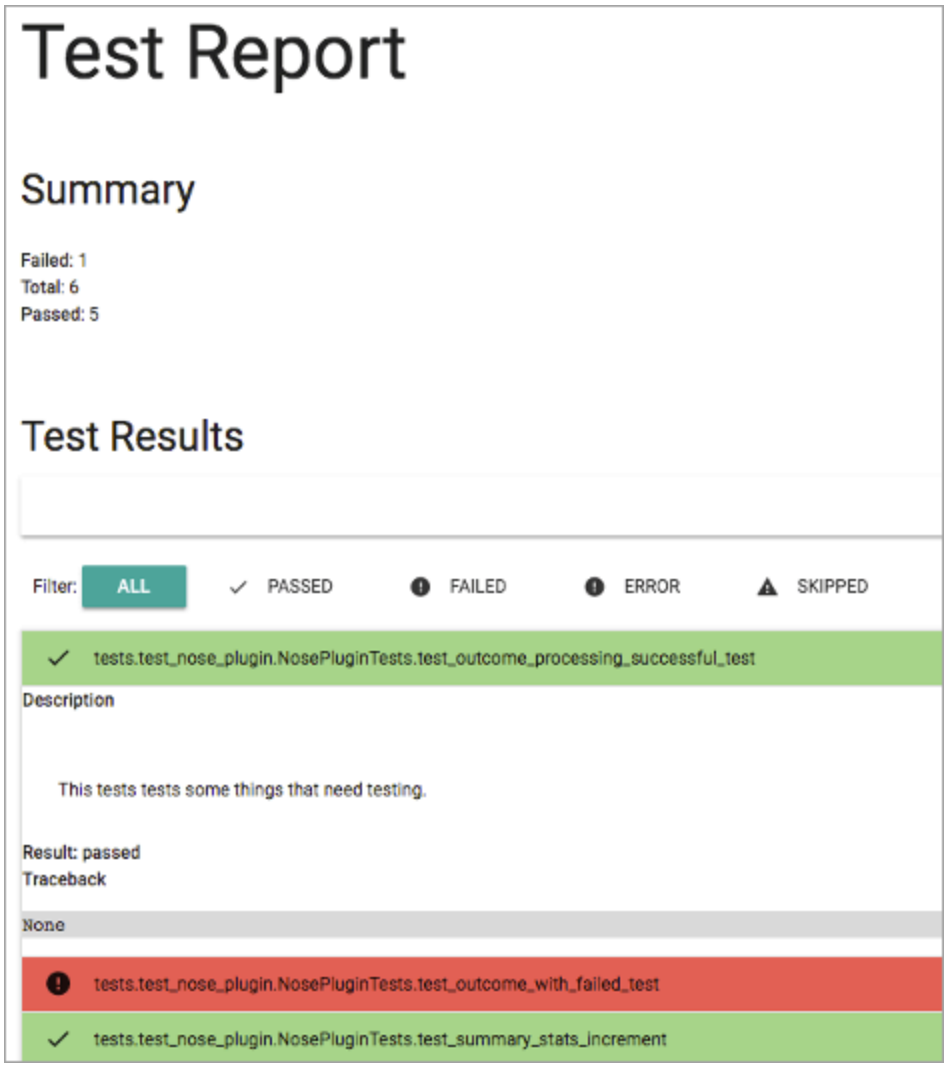


Image 9.2.2.5.1: Nose2 Testing Report

9.2.2.6 Testify

Testify is a Python-based unit testing framework built to be a replacement for unittest and nose. [105] It was built to be more "Pythonic", and less Java-inspired (remember that unittest was inspired by JUnit). New features compared to unittest include class-level setup/teardown fixtures, better test discovery (looking into packages to find tests), and support for collecting/running tests by collecting modules/classes/methods into organized testing suites. Below is an example of Testify testing, including the setup/teardown fixtures, along with a visual showing the Testify software:

```
from testify import *

class AdditionTestCase(TestCase):

    @class_setup
    def init_the_variable(self):
        self.variable = 0

    @setup
    def increment_the_variable(self):
        self.variable += 1

    def test_the_variable(self):
        assert_equal(self.variable, 1)

    @suite('disabled', reason='ticket #123, not equal to 2 places')
    def test_broken(self):
        # raises 'AssertionError: 1 !~= 1.01'
        assert_almost_equal(1, 1.01, threshold=2)

    @teardown
    def decrement_the_variable(self):
        self.variable -= 1

    @class_teardown
    def get_rid_of_the_variable(self):
        self.variable = None

if __name__ == "__main__":
run()
```

Image 9.2.2.6.1: Testify example code

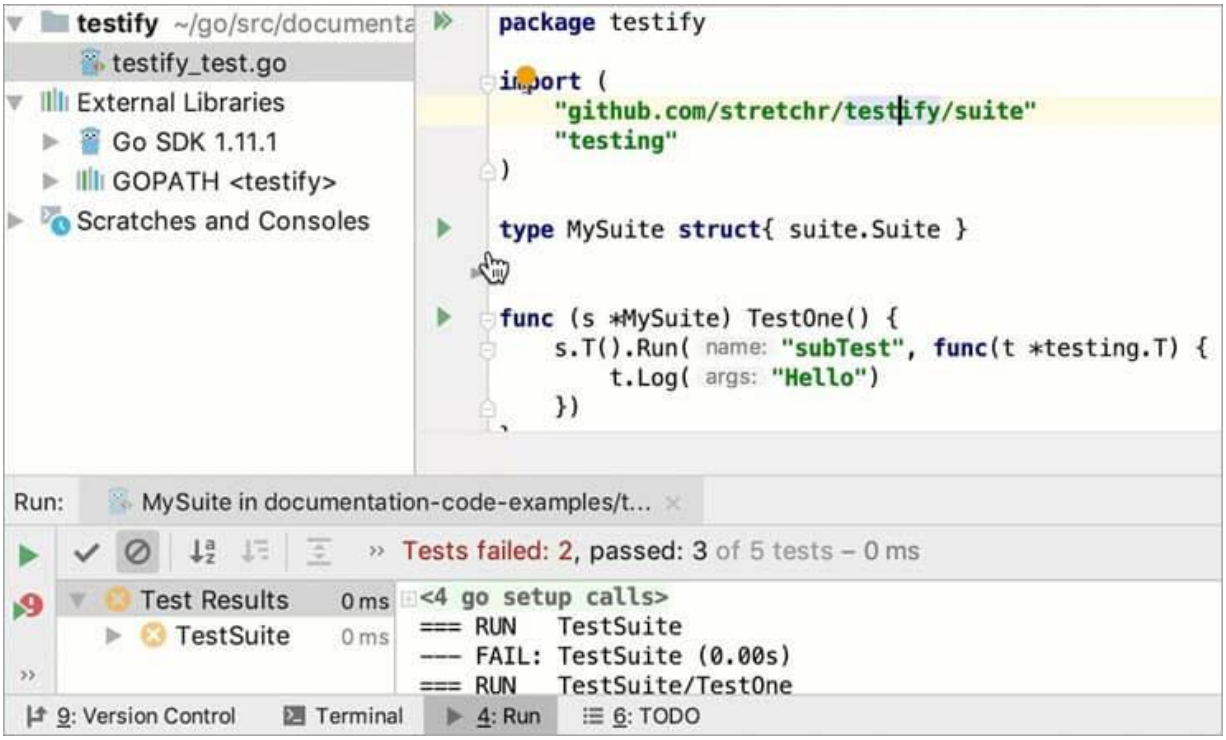


Image 9.2.2.6.2: Testify testing report [106]

9.2.2.7 Behave

Behave is a Behavior Driven Development (BDD) testing framework for Python that implements black-box testing. [107] Behave uses a similar style to Robot in that it uses what they call a "natural language style", that is backed up by Python code to actually implement. [108] Like Robot's keywords, this improves readability and understanding among the testers. This "natural language style" is used in "feature files" that define the behavior of the tests, and the Python code actually implements the testing based on the format of these feature files. Below is an example of how the feature files and python testing code interact:

```
# -- FILE: features/example.feature
Feature: Showing off behave

  Scenario: Run a simple test
    Given we have behave installed
     When we implement 5 tests
     Then behave will test them for us!
```

```
# -- FILE: features/steps/example_steps.py
from behave import given, when, then, step

@given('we have behave installed')
def step_impl(context):
    pass

@when('we implement {number:d} tests')
def step_impl(context, number):  # -- NOTE: number is converted into integer
    assert number > 1 or number == 0
    context.tests_count = number

@then('behave will test them for us!')
def step_impl(context):
    assert context.failed is False
    assert context.tests_count >= 0
```

```
$ behave
Feature: Showing off behave # features/example.feature:2

  Scenario: Run a simple test          # features/example.feature:4
    Given we have behave installed     # features/steps/example_steps.py:4
    When we implement 5 tests          # features/steps/example_steps.py:8
    Then behave will test them for us! # features/steps/example_steps.py:13

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
```

Images 9.2.2.7.1-3: Behave example code and testing report

9.3 Website Design

9.3.1 Choosing an Appropriate Color Palette

A good color palette is an important part of any brand, company, or product. It's what allows us to recognize a product almost instantly. For example, red could be associated with Coca-Cola, or yellow, McDonald's. Although somewhat overused, blue is often associated with health and security. It's often a favorite choice for hospitals, insurance companies, and banks for this very reason. The University of Central Florida has a simple yet easily recognizable color palette consisting of only two colors: black and yellow. By applying shades to a color, one can achieve a near-infinite number of combinations with only two colors.

Most popular palettes usually consist of two or three colors. Brands that use a three-color color palette have the main color, a secondary color, and a primary (or also called accent) color. When choosing a color palette for a brand, company, or product, it's important to think about what the colors mean and what they represent. Red often represents passion, anger, excitement, and importance. Orange represents playfulness, vitality, and energy. Yellows trigger happiness. Green invokes a sense of stability and growth. Blues signify trust, security, and formality. Purple signifies royalty and luxury. Pink is usually taken to be feminine. Brown can create an old-fashioned and stylistic look. White represents cleanliness and virtuosity. Gray is a neutral color and can take on many different characteristics based on the colors with which it is surrounded, and Black can feel powerful, edgy, and give off a modern feel.

To choose a proper color palette for UCF's Center for Humanities and Digital Research, it's important to understand what they stand for and what they represent. To quote directly from UCF's Center for Humanities and Digital Research homepage: "UCF's Center for Humanities and Digital Research (CHDR) is a collaborative research hub in the College of Arts and Humanities connecting faculty researchers and graduate research associates with community partners and grant-making institutions through project planning, technical expertise, grant writing, and ultimately dissemination in conjunction with the scholarly and literary journals also housed in the college. The intellectual and technical resources of CAH come to bear in CHDR, where groundbreaking interdisciplinary digital research is fostered and shared through the academy and the surrounding communities of Central Florida and beyond." If we dig through the keywords, we see that CHDR stands for collaboration and community with a focus on intelligence. As such, a color palette needs to appeal to these keywords and represent what CHDR stands for.

After careful research, planning, and feedback, the following color palettes were considered:

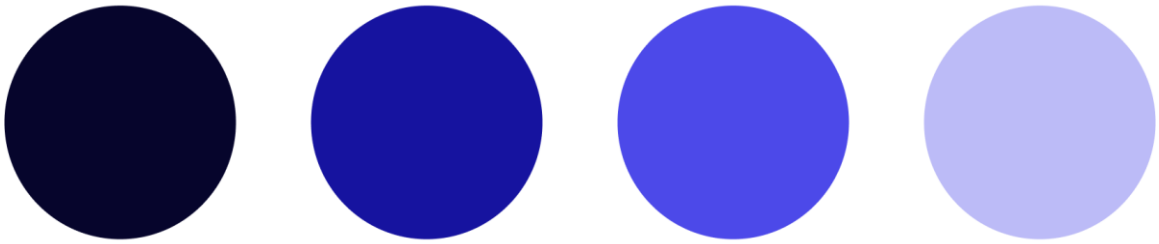


Figure 9.3.1.1 - Monochromatic color palette

The above is a monochromatic color palette consisting of one base color split into four different shades. This color palette blends between a dark blue and a light purple. As mentioned earlier, blue can sometimes be overused, however, changing the shade of blue by blending it with purple can help fight this problem. By using blues and purples, the color palette can give off a sense of trustworthiness, security, and luxuriousness.

Figure 9.3.1.2 - UCF's color palette

This color palette takes its inspiration from UCF's brand and style guide. It contains black, the main text color for UCF, a neutral gray, usually used for softer headings and links, a bright gold, for items that need to steal attention, and a softer muted gold to compliment. Because the Center for Humanities and Digital Research (CHDR) is UCF owned, students and faculty that attend UCF will feel at home with these colors.

Figure 9.3.1.3 - CHDR homepage color palette

Inspired by the colors used on the homepage for the Center for Humanities and Digital Research, this color palette features three bright soft pastel colors with an off-white color to pair with it. Truthfully, this color palette may be difficult to work with, but it stands out and lends itself to being easily recognizable.

9.3.2 Considerations in Logos

Complementing a good color palette is a good logo to go along with it. In today's times, the COVID-19 pandemic has shifted most consumer interaction towards a digital-first physical-second approach. This means that a product's digital impact is more important than ever. As such, if a company wants consumers to remember their product, they must leave a memorable impression, and a well-balanced logo is a great way to do so.

Most popular logos can be put into the following categories: minimalist, variable, responsive, and symbolic. Minimalist logos, as the name implies, have a minimal design usually consisting of only the company name. This is a good fit for larger companies where the name already invokes a sense of familiarity. Google, Venmo, and Uber are a few examples of this. They've become big enough that their logo no longer needs to define who they are. People today can say terms such as "Let me Google that", or "I'll call an Uber", or "Venmo me five dollars for the bus fare".

Figure 9.3.2.1 – Popular minimalist company logos

These logos use a simple flat sans-serif typeface that's readable at medium and large font sizes. Because of the simplicity of these logos, they can scale amongst various font sizes.

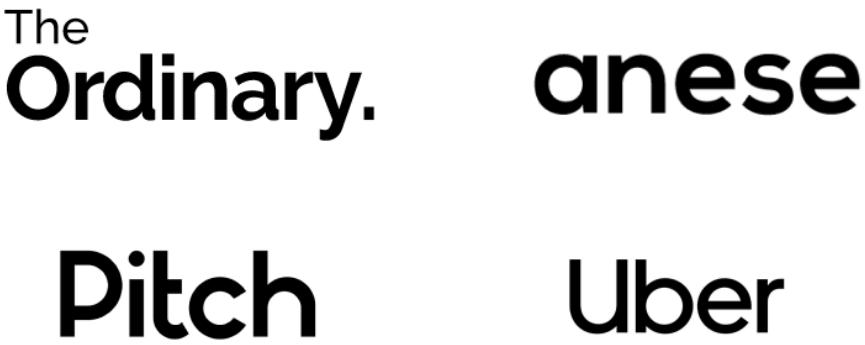


Figure 9.3.2.2 – Ultra minimalist company logos

With logos like these, simplicity is key, and less is more. When a minimalist logo is used, the focus is put not specifically on the company itself, but rather the content the company produces.

Variable logos are a newer modern design trend. Because they're difficult to implement correctly, they aren't seen in the wild as much as other types of logos. Variable logos are logos that can take on many different forms. One popular example is Adobe's Creative Cloud suite. Each logo has the same initial design and layout, but their styles differ. Each logo is unique to each product, yet still invokes a sense of similarity and familiarity.

Figure 9.3.2.3 – Adobe Create Cloud logos

Responsive logos, as the name implies, are logos categorized by their ability to scale responsively to a variety of media. Whether it be digital screens of different sizes, print media, or on a billboard, logos in these categories can fit anywhere. Responsive logos have elements that differ slightly depending on the size of the media on which they are displayed. Responsive logos don't only differ in size, they may also differ in shape or color. A company may choose to have their main logo in print media use a bright and colorful color palette while their logo in a digital landscape uses a monochromatic color palette.

Figure 9.3.2.4 - Slack's variable logo design

Because responsiveness is dependent on the context, simply shrinking the logo isn't enough. The ubiquity of high-resolution displays means that the quality of the icon needs to be taken into consideration as well. While responsive logos may always seem like the way to go, there are a few caveats that need to be addressed.

One drawback is that responsive icons take much longer to design than regular icons. One could argue that iconography is already a tedious and time-consuming process, so why add extra work into the mix? Another caveat is that responsive icons are by nature less recognizable than other types of icons. If an icon merely changes in size, it's clearly still recognizable as the same icon the user knows. A change in color is a little more noticeable, but can still be recognizable. Although, even a small change like this might have serious implications. When popular soft drink company Coca-Cola changed their vibrant red soda can to a light shade of blue, some were concerned that the company may have been trying to imitate their rival Pepsi [109]. The most drastic of changes to an icon is changing the appearance. As seen in figure C9.3.2.4, Slack's logo changes in size, color, and shape, but is still recognizable. When done correctly, changing a well recognizable logo can attract the user's attention and lead to higher engagement. The University of Central Florida has a very recognizable logo that also happens to be variable as well. The official UCF seal comes in three different shades, each keeping the same text but differing in the color palette. The UCF pegasus logo is most notably seen across campus

and on official UCF documents. It features a golden pegasus displayed on a black background with white text underneath. However, that same pegasus can also be seen preceding the names of various colleges within UCF. For example, the College of Arts and Humanities features the pegasus logo with a white background and black text.

Figure 9.3.2.5 - UCF's main logo

Figure 9.3.2.6 - CHDR's unit identity

Figure 9.3.2.7 - The logo for UCF's Center for Humanities and Digital Research. This is an appropriate logo to incorporate into our application.

9.3.3 A Comparison of UI Libraries

Front-end UI design can be a tedious process. If done from scratch, a developer needs to create every component and style every detail by hand. For a solo developer on a small and simple project, this is an easy task. For a team of developers on a project that could take months to develop, developing components from scratch would be too arduous of a task to even consider. A team of developers could however create their own reusable component library. Components would only have to be written from scratch once, and could then be reused everywhere. This is still cumbersome as most would say it's "reinventing the wheel". There are

numerous UI libraries available to the public, so in most cases, when development is under a strict time limit, it's generally better to use a pre-existing solution rather than creating that which has already been created. There is still one problem: not every UI library is created equally. Some have strict licensing guidelines that prevent them from being used in projects that aren't open-sourced, while some UI libraries may make developers pay a fee to use. Some UI libraries might have complicated APIs that slow down development time, while others may be too simplistic and lack important features required by a client. When choosing a UI library, it's important to consider a few topics: cost, documentation, usability, compatibility, and style.

9.3.3.1 Bootstrap

Bootstrap is an extremely common design library that's super easy to customize, has great documentation, and is well maintained. It's open-sourced, MIT licensed, has over 154 thousand stars on GitHub, and is updated on a regular basis. The fact that bootstrap just works right out of the box is what makes it such an easy library to use. It even has wrappers around it for various JavaScript frameworks, like React, Vue, and Angular. Any library isn't without its downsides, however. A very popular criticism is that all bootstrap websites have the same appearance. This is usually true if bootstrap is used without any customizations applied. When beginners learn to code for the front-end, they usually reach for bootstrap because it doesn't require customization. The same can be said for developers who have extensive programming experience but lack design experience. Bootstrap is quick and gets the job done, but carries the weight of needing extensive customization in order to stand out from the rest of the crowd. If used, developers should be aware of the fact that they may need to spend extra time customizing typography, color, and various other aspects of the library.

9.3.3.2 Ant Design

Ant Design is a library popular among Chinese websites. Like bootstrap, it requires no customization, works with React out of the box, and has even been ported to Vue and Angular. With an MIT license, over 76 thousand stars on GitHub, and thousands of commits, Ant Design is well maintained and well documented. And plus for this library is that the default styling looks less recognizable than bootstrap simply because it's used less often than bootstrap. However, it's not quite as easy to customize. Changing the styling of components has to be done manually with SCSS selectors that can sometimes become confusing and complicated. Unlike bootstrap, Ant Design's documentation also features the design patterns they themselves follow. They give fantastic advice on how to appropriately utilize proximity, contrast, repetition, typography, and tons of other design patterns and design principles.

9.3.3.4 Material UI

Material UI is a React component library that bases its design principles on Google's own material UI guidelines. It features ready-to-use components that work right out of the box and requires no styling and little customization. Like the others, this library is open-sourced and MIT licensed. It has over 73.3 thousand stars on GitHub and is actively maintained with over 18 thousand commits as of today. Popular tech companies like Spotify, Amazon, and even Netflix use Material Design in their own products, yet, Material Design suffers from the same problem as Bootstrap: most (if not all) uncustomized Material Design websites look the same. The Material Design language was created in 2014 by tech giant Google to revamp their design of Android [69]. Since then, it can be seen in pretty much every Google service and Android phone today. Therein lies the problem. Because Material Design is so widely used, it's easily recognizable as "Google's design". Websites that don't customize Material Design's default styling can come off as trite and sometimes even effortless.

9.3.3.5 Materialize.css

Materialize.css is a UI framework based on Material Design. It has its focus on user experience and aims to speed up development. What makes this UI library stand out from the others is the fact that Materialize.css doesn't necessarily come with premade components. Instead, it's essentially just one large CSS (or SCSS) file that you include in your project. The benefit of this is that the adoption of Materialize.css becomes framework agnostic, in that it doesn't matter if you're using React, Vue, or native HTML, it'll work in any case. Although Materialize.css has great documentation and over 38 thousand stars on GitHub, it appears that it's no longer actively maintained. It has an ever-growing number of issues that have yet to be resolved thus making it a less than ideal candidate to use in a production-grade application.

9.3.3.6 Chakra UI

Chakra is a UI component library for React that prides itself on accessibility and modularity. It offers quite a few premade components that are easy to customize to fit your needs. It is actively maintained with over 22 thousand stars on GitHub and offers quite detailed documentation with plenty of examples. While this may sound promising, Chakra UI suffers the same fate as some of its contenders in that the default look of Chakra UI can come off as overused, bland, and commonplace. Because of this, a lot of effort and time will need to be put into the customization of the framework alone.

9.3.3.7 Fluent UI

Fluent UI is a UI and UX design library that features a collection of premade components as well as guidelines on the application of those components. Created and maintained by Microsoft, it has over 12 thousand stars on GitHub and is also actively maintained. Compared to

other UI design libraries, this one has a unique feel to it, almost as if it borrows some of its design choices from material design. Because this library is seen less often in the wild, it doesn't have the same bland and common look as other libraries seen on the web. While it may have a unique look and feel, the style may remind users familiar with the Microsoft product suite of another Microsoft product. As this project is affiliated with the University of Central Florida, a Microsoft style may not fit with the style intended for this project.

9.3.3.8 Semantic UI React

Semantic UI React is a spin-off of the popular UI library Semantic UI. Similar to other UI libraries, Semantic UI offers ready-to-use and easy-to-customize components. Semantic UI doesn't have its own styling system and instead relies on the theming of Semantic UI. This means that with a few modifications of LESS or CSS, it can be customized to your liking, albeit the default styling is partially reminiscent of Bootstrap. With over 12 thousand stars, this library is somewhat actively maintained

Because we're looking for a UI library that's easy to customize, offers a number of components that work out of the box, and stands out from the crowd, Ant Design seems to be the most viable choice. It has no association to Google or Microsoft with Material Design or Fluent UI, the default styling doesn't have the same bland look like Bootstrap, and it's MIT licensed, open-sourced, and actively maintained on GitHub.

9.3.4 A Comparison of Front End Hosting Services

A web hosting service is an internet service that runs servers connected to the internet, allowing individuals and organizations to serve content or host their own services [110]. Using a hosting service offers various benefits over self-hosting. For development, an Apache server will be provided, however, it is still important to consider other services to host the front end. One benefit of a hosting service is that it offers improved website security. There are engineering teams dedicated to making sure your website is secure. Some hosts even give users the ability to install extra security plugins and perform regular backups. Another important benefit is uptime. Most hosting services boast an uptime of 99.9% or greater. This means that you won't have to worry about your site going down. Unfortunately, not every hosting service is created equal, and one of the main drawbacks of using a service is the cost. While most services do offer a free tier, it's quite limited in terms of features, and may not scale well enough to fit one's needs.

9.3.4.1 Apache - Self Hosting

As mentioned earlier, an Apache server will be provided for development, but Apache could also be considered for hosting the front end as well. Owned by CHDR, we as developers would have near full control over various configuration options, most of which have already been set up. Because our site will most likely only be accessed by those in the general UCF area, using a hosting service that distributes its servers across the map won't benefit us much in this situation. Another benefit is user security. Because this Apache server is managed by UCF staff, we know exactly where our data is stored and who can access it. Not to mention we'll also have access to log files in the event of some error. As for the drawbacks, if the Apache server is down, then the website will be inaccessible. This means we'll need to be extra cautious with how we may handle potential server restarts. As for another drawback, one of the features of this project is a feature to automate the sending of future emails. This can be accomplished with the use of cron jobs. However, in the event that we need to perform a server reboot, there's no guarantee that those scheduled cron jobs will persist after a reboot completes. Unless we keep track of log files, it's also difficult to tell if and/or why a cron job failed. Another caveat of self-hosting on an Apache server is the time it takes to deploy the site. Most hosting services offer a one-click deploy solution with the ability to easily roll back to a previous version if necessary. Apache on the other hand doesn't have this functionality. One small change in any part of the site means the entire site will have to be built and uploaded to the server manually. This is a mundane error-prone task that ultimately slows down development time.

9.3.4.2 AWS Amplify Hosting

AWS Amplify is a collection of purpose-built tools and features that lets front-end developers build, launch, and scale applications with little to no cloud expertise needed. It has a number of useful features including continuous integration and continuous deployment (CI/CD) workflows that automatically deploy your site when changes are pushed to a GitHub branch, pull request previews that allow you to preview your changes to the site before it goes live, real-time monitoring with the ability to create custom alarms, and easily configurable redirects, rewrites, and custom headers. Using AWS Amplify to host the front end could greatly speed up development time and would remove the need to manage configuration files. AWS prides itself on its 99.999% uptime so we'd rarely have to worry about outages or server crashes. With CI/CD, we wouldn't need to upload a build of the site every time we make a change, a simple push to a git repository will trigger an update. Because Amplify is a part of the AWS ecosystem, it would also pair well with other Amazon services. As mentioned earlier, an integral part of this project is the ability to schedule and send future emails. A cron job isn't a production-level solution, but AWS Lambda could be. While AWS Amplify has a surprisingly large list of pros, any choice isn't without its cons. AWS Amplify follows the typical pay-as-you-go monetization model with a slight caveat. Hosting is free, but only for 5 gigabytes stored per month, 15 gigabytes served per month, and 1000 build minutes per month for the first year. Any usage that tops that limit gets charged $0.023 gigabytes stored per month, $0.15 per gigabyte served, and

$0.01 per build minute. These prices sound reasonable but will eventually add up in the long run, especially considering that we'll be storing multiple images that might take up quite a bit of storage.

9.3.4.3 Netlify

Netlify, similar to AWS Amplify, is a platform that makes it easy for developers to deploy and host a static site. Netlify offers continuous integration and continuous deployment through GitHub, enabling you to update your site by pushing to a branch on a git repository. Unline AWS Amplify, Netlify offers plugins built by themselves and the open-source community that aim to add extra functionality to your site build. If you've already purchased a domain name, it's easy to configure the site to use that custom domain instead of the default domain name Netlify provides. With thorough documentation and great customer support, it's easy to get a site up and running in no time. One major con of using Netlify is their pricing plan. For small projects, the free tier is great. It offers CI/CD, live site previews, and instant rollbacks to previously deployed versions. However, you're limited on a few other features. The free tier only supports a team of one member, only 100 gigabytes of bandwidth per month, and only one thousand unique visitors per month. The next plan up is $19 per month per member, then after that, $99 per month per member.

9.3.4.4 Vercel

Vercel is another front-end hosting service that supports static sites and serverless functions. Vercel is similar to Netlify in that it has a simple one-click-deploy solution. Simply import a git repository and with the click of a button and your site is live. Although it won't matter much in this project's situation, Vercel has a global edge network. This is a type of topology often referred to as a content delivery network that handles routing site traffic depending on a user's location to a specific server. As most, if not all traffic into this project will be coming from UCF, this is a feature that isn't a top concern at the moment. Vercel also automatically handles caching. Caching is the process of storing copies of files in a cache (or temporary storage location) so that they can be accessed quicker. Most modern web browsers already cache HTML and JavaScript files to allow websites to load quicker but Vercel also caches the response of a serverless function. As mentioned earlier, serverless functions (similar to that of AWS Lambda) could benefit us when it comes time to implement email reminders. As for pricing, Vercel doesn't have as flexible a model as AWS Amplify. There is a "Hobby" tier, but it's only for non-commercial and hobby sites. The next tier up is $20 per month per member but includes up to one terabyte of bandwidth, unlimited function requests, and one terabyte-hours of execution.

9.3.5 Testing the Front End

9.3.5.1 Test Driven Development - TDD

Testing is an important part of any application. Developers should write unit tests to ensure that their application behaves as expected. This helps detect and prevent future bugs and mistakes. Some teams write their tests before they write their code. This technique is known as test-driven development or TDD. In TDD, project requirements and milestones are turned into test cases. Code that implements new features specified by some requirement automatically has its test written. In this approach, no new code is added that hasn't been proven to meet the requirements.

9.3.5.2 Types of Testing for Front End Applications

Testing front-end code is a little different than testing backend code, or code that runs on a server. Some developers may only test backend code with the argument that testing the front-end isn't necessary because you can see your changes as you develop. This is far from the truth as a large front-end application contains many complex and interwoven moving parts such as state management, authentication, routing, asynchronous data queries, complex rendering, and much more.

Testing can be broken down into three main categories: unit testing, end-to-end testing (also commonly referred to as E2E testing), and integration testing. Unit testing is the most common type of test for front-end applications. There are a number of open-source libraries that aid in the process and there are countless examples of how to write proper unit tests for front-end applications regardless of framework. Unit testing involves rendering a part of the UI and testing it in isolation. In practice, this would involve writing a test case that renders one single component and compares its output to some expected value. The next type of testing is E2E testing. This is the process of testing if the execution of front-end code performs as desired from the start and to the end. The entire application is tested in a real-world situation that involves testing the communication between certain components and some database, network, or API. This also involves testing your code in a variety of browsers, not only web browsers, but mobile browsers as well. While E2E tests for the front-end do require a lot of time and effort, they're certainly beneficial in the long run as you can be certain that your code not only communicates as expected but also works in different browsers. Integration is the final type of testing that consists of testing the communication between the user interface of an application and some API. It takes the shortest amount of time to write and is even supported through GitGub with continuous integration testing. This would allow us to run tests on our front-end code every time a commit is made to the master branch. This also allows us to make sure all pull requests pass the tests we write before any code gets merged into the master branch.

9.3.5.3 A Comparison of Testing Libraries and Frameworks for Front End

Testing libraries and frameworks help speed up the writing of tests by introducing helpful functions that remove boilerplate code and eliminate mundane repetitive tasks. Because our application will be written with React, we'll need to consider only testing libraries and testing frameworks that can integrate smoothly with React. As such, we've considered the following frameworks: Jest, Enzyme, React Testing Library, and Cypress.

Jest is probably the most popular testing framework for testing React applications. It's open-source, maintained by Facebook developers, has over 37 thousand stars with an MIT license, and commits are pushed to the repository weekly. Jest allows you to write tests contained in files that relate to specific parts of an application. For example, one could write a suite of tests that check if state management is being handled correctly. Jest also allows developers to test React components in isolation. You can render some component (with or without props passed to it) then verify that what was rendered is what was expected. Jest also has the ability to show code coverage. This is a measurement of how much code and what specific lines have been touched by test cases. Although, some may argue that code coverage statistics provide a false sense of security and normalize the value of code, forcing developers to treat all code equally instead of prioritizing more important parts of the codebase.

Enzyme is another popular testing framework created and maintained by Airbnb. It's open-source, has almost 20 thousand stars on GitHub, and is MIT licensed. Enzyme's API is meant to be intuitive and flexible, with a syntax that borrows from jQuery's DOM manipulation syntax. Enzyme is very similar to Jest in that it allows you to test components in isolation to verify if what was rendered matches some expected output. Enzyme offers one feature that sets it apart from Jest: it allows the use of shallow and deep rendering. Shallow rendering is useful to constrain yourself to test a component as a unit and to ensure that your tests aren't indirectly asserting the behavior of child components. Like shallow rendering, a deep render not only renders the component, it also renders all of its children. This is useful for testing the lifecycle of a component or for testing if a component has children that are causing a render to slow down. Enzyme can also be paired with Jest to create powerful custom assertions and convenience functions to test a React application.

React Testing Library is a lightweight testing library that comes bundled by default with projects created using the Create React App command-line interface. It provides small utility functions on top of the React DOM library in a way that encourages better testing practices. In fact, their primary motto is "The more your tests resemble the way your software is used, the more confidence they can give you" [111]. React Testing Library is open-source, MIT licensed, and maintained by Facebook. It has over 15 thousand stars on GitHub and is actively maintained. This framework can be thought of as a lighter version of Jest without all of the bells and

whistles. It allows you to render components in an isolated environment and compare the output to some expected output. The downside of React Testing Library is that it isn't made for production-level applications. In fact, the library itself states that it's not a framework and that they recommend Jest for more complex functionality.

Sitting at almost 35 thousand stars on GitHub with an MIT license and plenty of recent commits, Cypress is different from the other testing frameworks and libraries mentioned above because it offers E2E testing. Cypress takes snapshots of your site as tests run and compares those snapshots so you can keep track of how changes in your codebase affect other parts of your application. Cypress, similar to the other testing frameworks, can also render certain components in an isolated environment, however, Cypress does so with the use of a real browser called Cypress Component Test Runner. This allows code to run in an actual browser instead of a virtualized document object model. The benefit is that it's closer to a real-world scenario and you can visually keep track of what's being rendered. This also gives access to the browser's dev tools so you can browse through Cypress's DOM.

After all testing frameworks have been considered, Jest and Enzyme will best fit our needs to test the front-end of our application. Jest has great documentation, pairs well with React, offers helper functions that are compatible with Enzyme, and works great when integrated with GitHub's continuous integration and continuous deployment pipeline. This will help us only commit code that has been thoroughly tested.

9.3.6 TypeScript vs. Javascript: Choosing an Appropriate Language

When choosing a language for our front-end, we need to take into account multiple considerations. Because our project is being handed off to other developers, we need to write code that is readable, scalable, and maintainable. For the front-end, we're limited to two languages: JavaScript and TypeScript. JavaScript is a great language in that it tries to work regardless of syntax errors, grammatical errors, or logical mishaps. There are hundreds of thousands of open-source libraries that are available to use and there are tools like ESLint and Prettier that can help with static code analysis and consistent formatting. Our sponsors are familiar with the language so developer-to-client handoff will be made simpler. However, JavaScript doesn't have types or static code analysis so it becomes difficult to tell why a variable has the value it does or what type a function returns. There's also a lot of room for errors as JavaScript lets you re-assign a variable's type without thinking. For React, there are libraries like Flow and PropTypes that aim to solve this dilemma by adding type hints to props passed to components, but this means more time for development and introduces additional dependencies into the project. It is for these very reasons we have decided to use TypeScript for our front-end. TypeScript is compatible with every JavaScript library, is maintained by Microsoft, and it

integrates well with React. TypeScript offers static type checking and prevents the developer from re-assigning a value of a different type to a variable. With this philosophy, developers can fail fast and catch type errors before they ever make it into production. As for developer-to-client handoff, TypeScript is extremely similar syntactically to JavaScript making the learning curve small. With types, it also becomes easier for an external developer to hop into the codebase when they know what type a variable has and why a function returns the type that it does. TypeScript also has a feature that allows it to run a test compilation of any TypeScript file. That test compilation will throw an error if there are any mismatched types. With CI/CD through GitHub actions, this will make for a great testing pipeline to ensure we produce production-grade code that runs without error.

9.3.7 UI Mockups

As with any front-end-based application, one of the most important parts involves creating a UI that is accessible, responsive and serves its purpose while looking aesthetically pleasing. As such, we've used Figma to create mockups of separate components that can then be pieced together to form a larger picture. Overall, our prototypes incorporate a minimalistic design with a monochromatic color scheme. This allows us to easily introduce additional colors and build on top of our layout in the event we decide to change some aspect of our design.

An integral part of the application is the ability for CHDR admins to view or make changes to an item in the inventory. As such, the following mockups were made in Figma to prototype possible designs to fit the sponsor's needs:

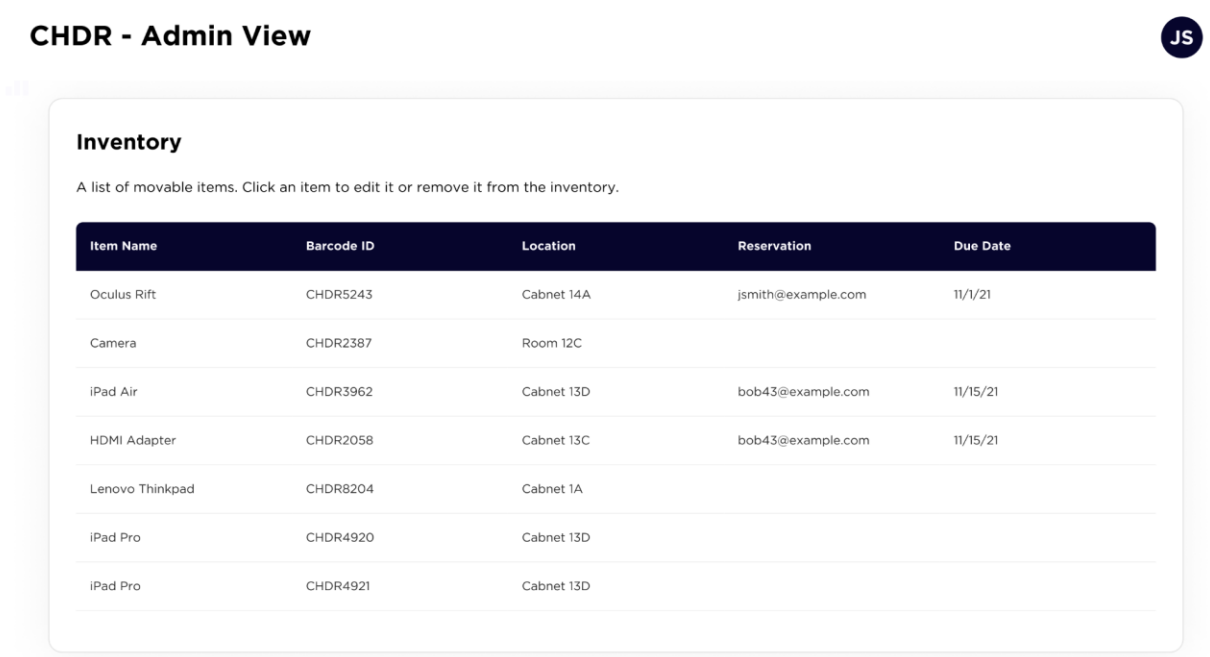


Figure 9.3.7.1 - A mockup of the inventory landing page. This page will be restricted to CHDR admins only. The purpose of this page is to allow admins to have a quick way to view what items they have in their inventory, who has an item checked out, and when that item is due (if it is checked out). Clicking on an item brings up an edit modal, as seen in the figure below.

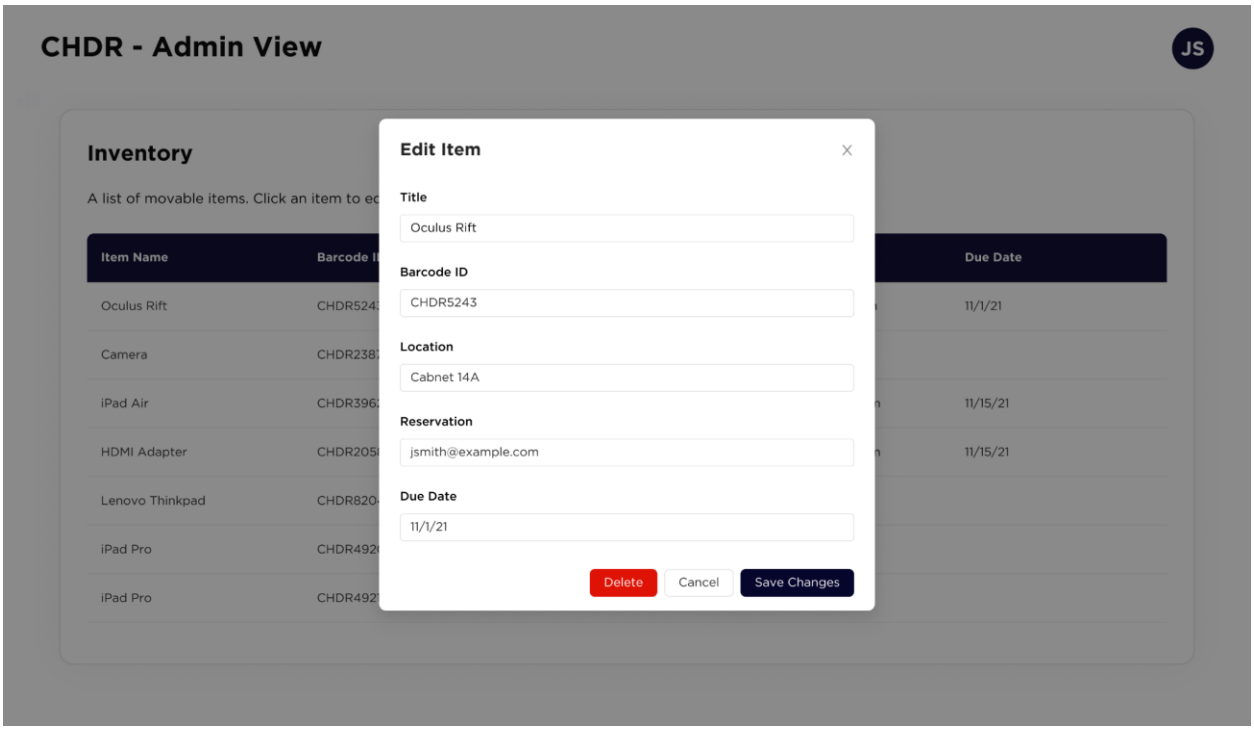


Figure 9.3.7.2 - This edit item modal pops up when an admin clicks on an item in the table.

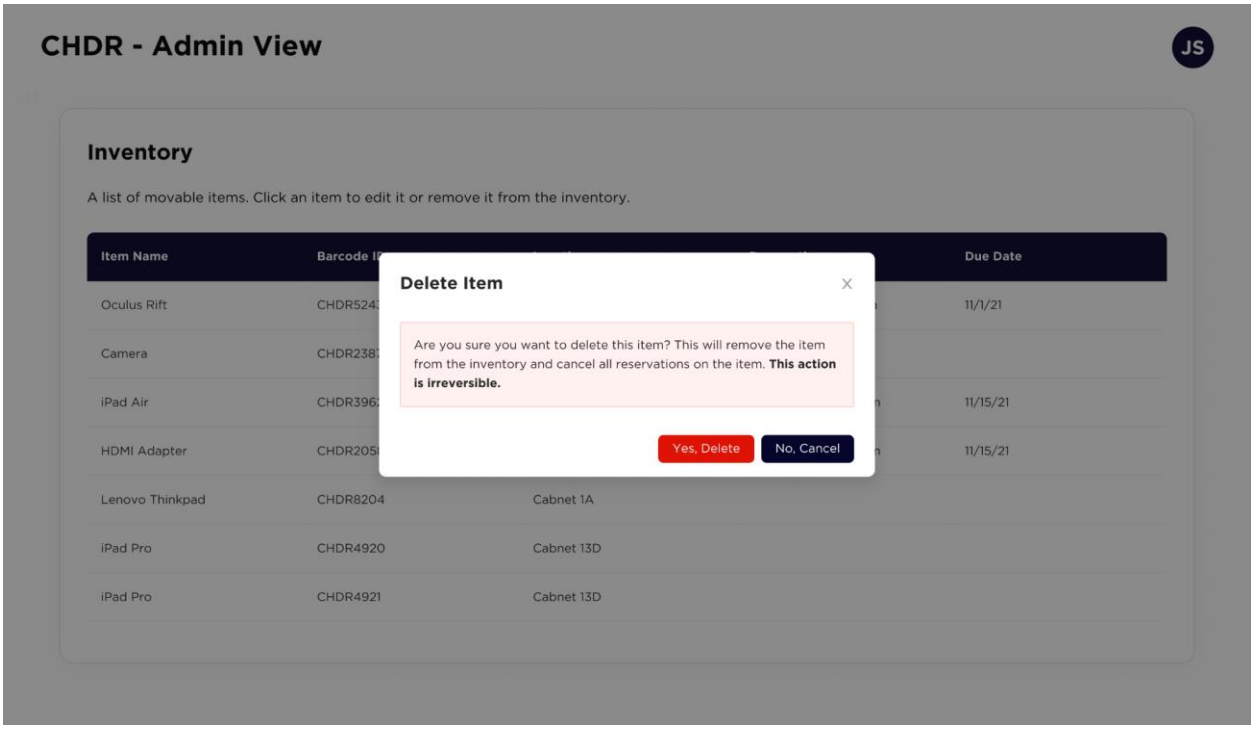


Figure 9.3.7.3 - This delete item modal opens after an admin clicks the “Delete” button (pictured in the previous figure above). Here, we make use of warning and danger colors to ensure an admin knows the consequences of this action. We refrain from using short actions such as “Ok” and “Confirm” and instead use more informative actions such as “Yes, Delete”

Another important part of this application is the ability for users to view items that are in the inventory and check them out. Because not all web traffic comes from desktops, we need to create a design that’s responsive and can easily resize to different screen widths. As such, the following prototypes were designed to meet these requirements.

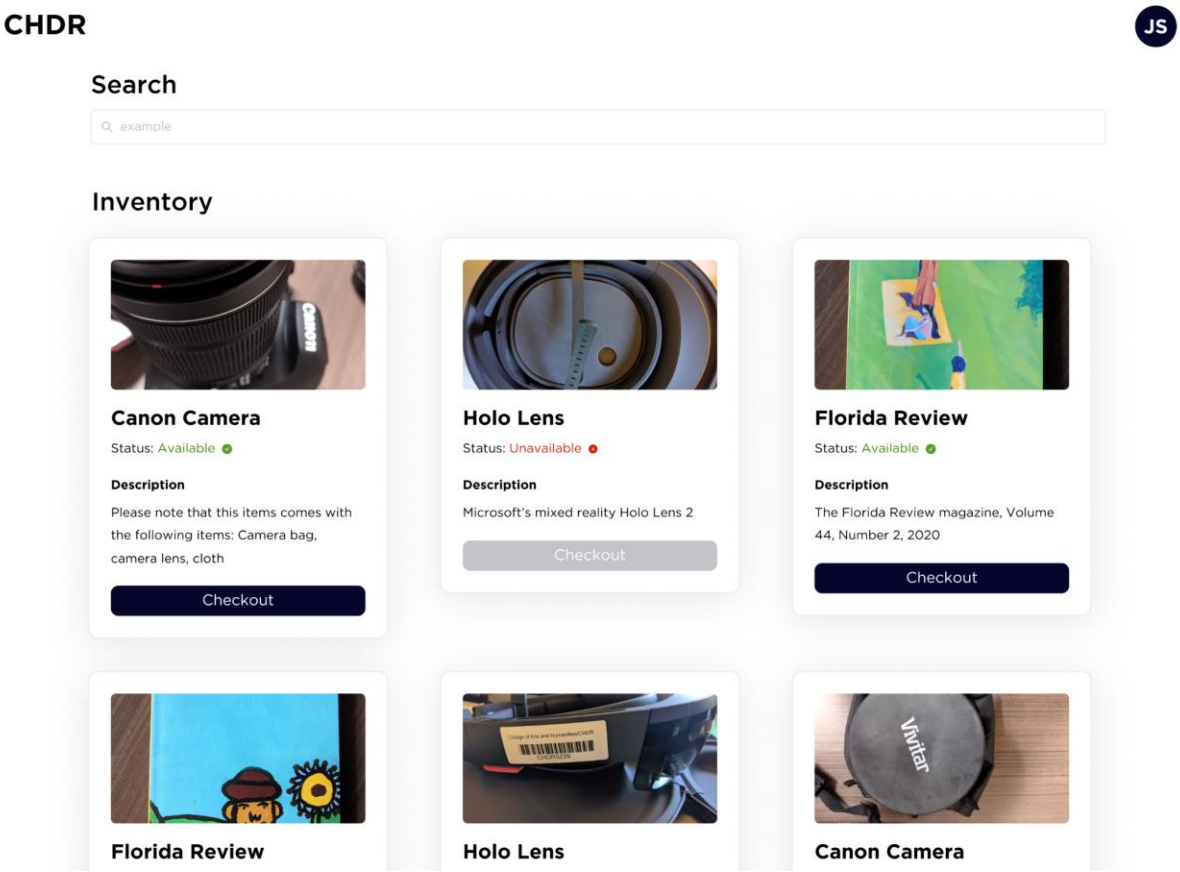


Figure 9.3.7.4 - A mockup of the inventory page. This may potentially be the landing page of the CHDR inventory application. Here, we make use of a responsive grid that can show a variable number of items depending on the user's screen width.

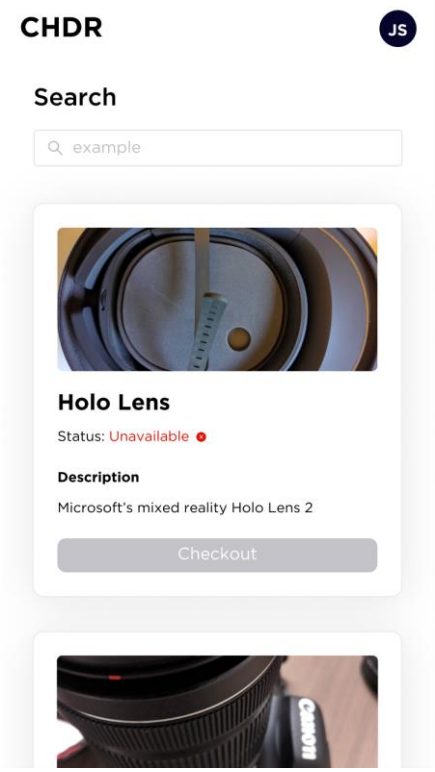


Figure 9.3.7.5 - A mockup of the inventory page on a mobile device.

In order to add items to the inventory, our sponsors require a mobile application that has the ability to take pictures, upload images, and scan barcodes. The following mockups demonstrate this functionality:

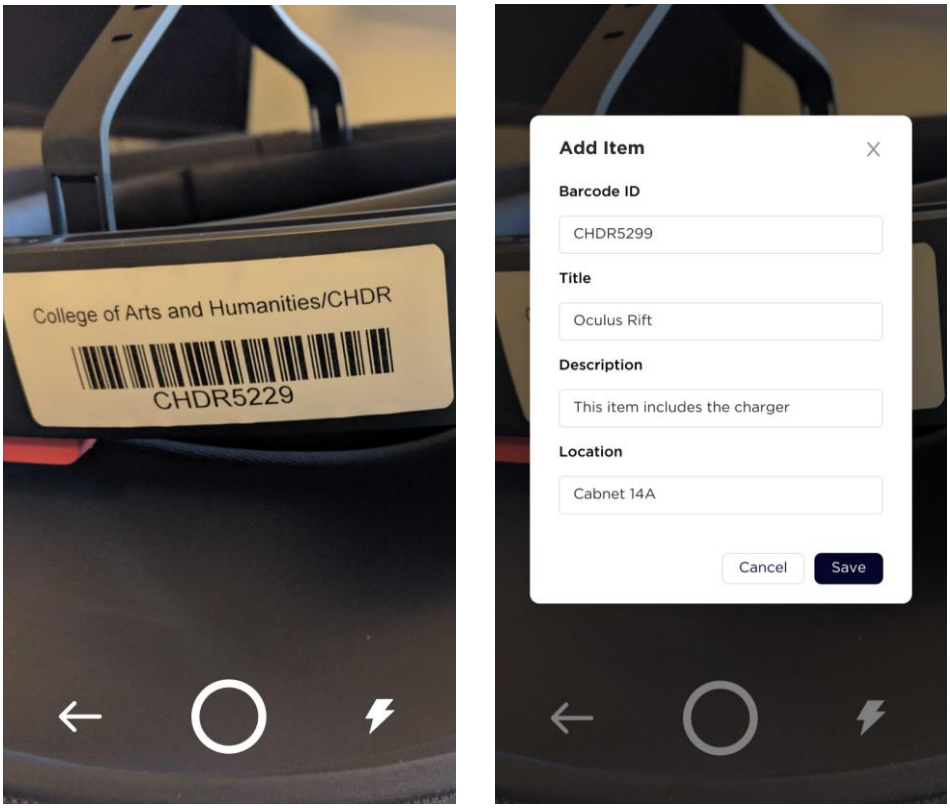


Figure 9.3.7.6 - This is a mockup of the mobile app's ability to scan barcodes. Once the barcode is in view, an "Add Item" modal pops up to allow the user to enter details about this item. To explain the controls below: the leftmost button exits the camera view, the center button captures a picture to upload, and the rightmost button turns on the camera's flash in the event the environment is too dark.

9.4 Mobile App

One of our requirements is to make a mobile application for both Android and IOS to use. We will use React Native to make the mobile application. The app will only be accessible to administrators working in the Humanities facility. They will have the ability to look up items in the database via search and barcode scanner.

Our app needs to be able to get access to the user's camera, scan a barcode and determine what item this is. The app needs to be connected to the Apache server set up by the sponsors and have access to the database. This is critical so the admin will be able to check-out/check-in items, and it will update in the system.

Once we complete the above task and have extra time, we will try to accomplish some mobile application stretch goals. One of those is to make a regular user login. The regular user

will be able to download the app, login with his UCF NID, and be able to lookup items to check-out or even rent a room for a specific time. Another stretch goal is to have the administrator access the facility statistics from their phones.

10 Project Milestones

- 9/22: Project Selection (Completed)
- 9/24: Meeting with project sponsors and discussing access to development server and database (Completed)
- 9/25: Connecting to development server and database; diagnosing any problems with connecting (Completed)
- 9/29: TA Check-In 1 (Completed)
- 9/30: Initial Design Document (Completed)
- 10/4: Receive CHDR Inventory from Sponsors (Completed)
- 10/18 Project Status Review with Professor (Completed)
- 11/8: Deployment Meeting w/ Leinecker (Completed)
- 11/10: TA Check-In 2 (Completed)
- 11/20: Database Implemented (Completed)
- 11/27: API (Completed)
- 12/4: Database/API Testing (Completed)
- 12/6: Final Design Document Due
- 12/12 - 1/9: Winter Break
- 1/10: Begin Development on Web and Mobile Front Ends
- 1/15: Identify Product Purchase Information
- 1/23: Begin Development on Login Pages
- 2/2: Begin Development on Barcode Scanner
- 2/10: Item Check-in/Check-out System Completed
- 2/18: Implement Usage Statistics
- 2/26: Implement Child Window Calendar for Separate Monitor
- 3/1: Implement Journal Cover Scanning System
- 3/6 - 3/13: Spring Break
- 3/23: Integrate All Components, Begin Unit Testing
- 4/25: Final Presentation

11 Project Summary, Conclusion, and Post-Thoughts

11.1 Project Summary

As mentioned before, the objective of this project is to provide the Center for Humanities Digital Research lab an application to manage their recently acquired inventory of items. We aim to provide functionality through our application for them to conduct an inventory of their current items and for them to distribute their items to students who want it. This will be accomplished with a scheduling and checking system that will allow students to not only check items in and out, but also to reserve items in advance. The mobile component will offer functionality for the inventorying system, whereas the web component will offer functionality for the checking and reservation of various items.

The Center for Humanities Digital Research lab was given a $500,000 dollar grant to purchase items and equipment that are intended to be used to facilitate the education of the College of Arts and Humanities' student body, but unfortunately, the grant did not include any form of software that could be used to facilitate the allocation of this equipment. Our application will bridge that gap: it will enable students to have access to the equipment they need to enrich their education and enhance their studies. In essence, our software will directly contribute to the education of the university's students.

Over the last few months, our team has diligently researched each of our prospective parts: database, application programming interface, image recognition, web application development, and mobile application development. We have considered many possible alternatives and options, weighing the pros and cons of each, and have made our choices for the tools and technologies we intend to use. Going forward, we will leverage our newly acquired knowledge to create this application such that it meets both the basic requirements and the stretch requirements of our sponsors.

We hope that our project will have the beneficial effect on the students that we predict, and we also hope that the information this project can provide will motivate the CHDR grantholders to continue giving the institution money for more equipment, thus further enriching the experience of the students. We also hope that this project can contribute to the education of students in a meaningful way, a way that is appropriate for a project with the scale and scope of a Senior Design project.

11.2 Lessons Learned and Insights Gained

11.2.1 Importance of Planning

One of the early issues our group encountered was inadequate planning and scheduling of the various milestones of the project. We had initially underestimated the scale of the project, and as a result, we did not adequately create milestones for various features of the project. As the timeline continued, our progress quickly became unsatisfactory, and it was eventually made clear to us that we needed to shape up. We adopted accurate and clean milestones as well as a Gantt chart and a Trello board to keep track of the progress of the project, and through these charts, we were able to get on track.

11.2.2 Knowledge Gained on the Various Technologies Used in the Project

Each member of our group gained a wealth of knowledge pertaining to the particular component of the project they were assigned. That knowledge was not limited to the basics of that technology, but also knowledge in debugging issues that occurred with the technology as the project was created.

11.2.3 Interpreting Sponsor Requirements

We learned how to cope with evolving sponsor requirements. During our bu-weeekly meetings with the sponsors, we sometimes had to adjust some of our project's features to address particular concerns that were brought up in these meetings. For example, at the time of writing, we had recently discovered that the use of the UCF logo for the tab image was not permitted, thus requiring us to change it to the CHDR logo. Furthermore, the structure of our database underwent several changes in both table fields and overall structure to satisfy requirements. The adaptability we learned will be useful in our future professional endeavors.

11.3 Overview of Completeness

- Backend
    - A scheduler runs daily to remind users of upcoming reservations via email
    - Users are sent an email upon getting their reservations approved
    - All routes are functional and tested
    - ISSUE: Because we don't have access to the CHDR email account, we had to use a temporary gmail in order to test the routes. As a result, some of the functionality of email notifications is restricted until the sponsors add their information.

    - ISSUE: UCF NID login services were initially on the list of requirements. However, after implementing it, we were contacted by UCF IT and informed that the storage of certain information was not permitted. As a result, we switched from UCF NID login to email/password.
    - ISSUE: Image Processing for journal covers was initially a requirement, but without a large pool of images to train with, along with a lack of experience or knowledge on machine learning generally, led to us being unable to implement this feature. Instead, we suggested to the sponsors that they have a single barcode to represent each journal set, and simply scan that barcode to reference the individual journals when checking them out.
- Mobile
    - Barcode scanning is functional and allows for administrators to scan items to retrieve information
    - Adding images to inventory items
    - Built android app via .apk and shared it with sponsors
    - Wrote instructions on how to build IOS app using Xcode.
- Frontend
    - Calendar that shows all reservations on a large screen
    - Functionality to distinguish between users, admins, and super users
    - Ability for admins to view usage statistics for all items
    - Users can create reservations and admins/super users can approve or deny those reservations
    - Only super users can promote/demote other users and admins

12 References

[1] Google. (n.d.). *US7156311B2 - system and method for decoding and analyzing barcodes using a mobile device*. Google Patents. Retrieved December 6, 2021, from https://patents.google.com/patent/US7156311B2/en.

[2] Google. (n.d.). *US9033243B2 - Barcode Rendering Device*. Google Patents. Retrieved December 6, 2021, from https://patents.google.com/patent/US9033243B2/en?inventor=Mark%2BA.%2BRoss.

[3] Google. (n.d.). *US10915857B2 - supply chain management using mobile devices*. Google Patents. Retrieved December 6, 2021, from https://patents.google.com/patent/US10915857B2/en.

[4] UCF. (2021, September 30). AMY LARNER GIROUX, PH.D. In *Faculty and Staff*. Retrieved from https://tandt.cah.ucf.edu/faculty-staff/?id=1039

[5] UCF. (2021, September 19). MIKE SHIER, PH.D. In *Faculty and Staff*. Retrieved from https://tandt.cah.ucf.edu/faculty-staff/?id=42

[6] *What is a database management system?* IBM. (n.d.). Retrieved from https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system.

[7] 29, A., & Raza, M. (2018, August 29). *DBMS: An intro to database management systems*. BMC Blogs. Retrieved from https://www.bmc.com/blogs/dbms-database-management-systems/.

[8] *What Is a Database?* Oracle. (n.d.). Retrieved from https://www.oracle.com/database/what-is-database/

[9] *14 Characteristics of a Database Management System.* What is DBMS | Let's Define DBMS. Retrieved from https://whatisdbms.com/characteristics-of-database-management-system/.

[10] Manghnani, C. (2019, December 4). *Metadata in DBMS - overview and types*. The Crazy Programmer. Retrieved from https://www.thecrazyprogrammer.com/2019/12/metadata-in-dbms.html#:~:text=Metadata%20in%20DBMS%20is%20the,data%20for%20the%20contextual%20data.

[11] *DBMS transaction property - javatpoint*. www.javatpoint.com. (n.d.). Retrieved from https://www.javatpoint.com/dbms-transaction-property.

[12] Wikimedia Foundation. (2021, March 19). *Atomicity (database systems)*. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Atomicity_(database_systems).

[13] Wikimedia Foundation. (2021, June 19). *Consistency (database systems)*. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Consistency_(database_systems).

[14] Wikimedia Foundation. (2021, July 3). *Isolation (database systems)*. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Isolation_(database_systems).

[15] Wikimedia Foundation. (2021, October 16). *Durability (database systems)*. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Durability_(database_systems).

[16] Techopedia. (2020, May 13). *What is database concurrency? - definition from Techopedia*. Techopedia.com. Retrieved from https://www.techopedia.com/definition/27385/database-concurrency.

[17] *Transaction states in DBMS*. GeeksforGeeks. (2021, October 22). Retrieved from https://www.geeksforgeeks.org/transaction-states-in-dbms/.

[18] *MySQL 8.0 Reference Manual :: 3.3.1 creating and selecting a database*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/creating-database.html.

[19] *MySQL 8.0 Reference Manual :: 13.2.6 insert statement*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/insert.html.

[20] *MySQL 8.0 Reference Manual :: 13.2.10 select statement*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/select.html.

[21] VanMSFT. (n.d.). *Update (transact-SQL) - SQL server*. SQL Server | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/sql/t-sql/queries/update-transact-sql?view=sql-server-ver15.

[22] *MySQL 8.0 Reference Manual :: 13.2.2 delete statement*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/delete.html.

[23] *MySQL 8.0 Reference Manual :: 13.1.32 drop table statement*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/drop-table.html.

[24] Cawrites. (n.d.). *Create a full database backup - SQL server*. SQL Server | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/create-a-full-database-backup-sql-server?view=sql-server-ver15.

[25] SQLSourabh. (n.d.). *Automatic, geo-redundant backups - azure SQL database & azure SQL managed instance*. Automatic, geo-redundant backups - Azure SQL Database & Azure SQL Managed Instance | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/azure/azure-sql/database/automated-backups-overview?tabs=single-database.

[26] Cawrites. (n.d.). *Differential Backups (SQL Server) - SQL server*. SQL Server | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/differential-backups-sql-server?view=sql-server-ver15.

[27] *DBMS - Data Backup*. DBMS - data backup. (n.d.). Retrieved from https://www.tutorialspoint.com/dbms/dbms_data_backup.htm.

[28] *What is Data Integrity in a database. why do you need it?* Astera. (2021, October 22). Retrieved from https://www.astera.com/type/blog/data-integrity-in-a-database/.

[29] Www.techonthenet.com. (n.d.). *MySQL: IS NOT NULL*. MySQL: Is not null. Retrieved from https://www.techonthenet.com/mysql/is_not_null.php.

[30] *MySQL 8.0 Reference Manual :: 3.6.9 using auto_increment*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/example-auto-increment.html.

[31] *MySQL 8.0 Reference Manual :: 1.7.3.1 primary key and unique index constraints*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/constraint-primary-key.html.

[32] Ian. (2020, November 18). *What is Referential Integrity?* Database.guide. Retrieved from https://database.guide/what-is-referential-integrity/.

[33] Chapman, T. (2007, June 26). *Defining cascading referential integrity constraints in SQL server*. TechRepublic. Retrieved from https://www.techrepublic.com/blog/the-enterprise-cloud/defining-cascading-referential-integrity-constraints-in-sql-server/.

[34] *MySQL 8.0 Reference Manual :: 11.6 data type default values*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/data-type-defaults.html.

[35] *Database solutions and downloads for Microsoft Access*. databasedev.co.uk - database solutions and downloads for microsoft access. (n.d.). Retrieved from http://www.databasedev.co.uk/domain_integrity.html.

[36] *Chapter 11 data types*. MySQL. (n.d.). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/data-types.html.

[37] *Designing and maintaining Essbase Cubes*. Oracle Help Center. (2019, January 17). Retrieved from https://docs.oracle.com/en/cloud/paas/analytics-cloud/adess/multiple-data-views.html.

[38] WilliamDAssafMSFT. (n.d.). *Create view (transact-SQL) - SQL server*. SQL Server | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver15.

[39] By: IBM Cloud Education. (n.d.). *Database security: An essential guide*. IBM. Retrieved from https://www.ibm.com/cloud/learn/database-security.

[40] *Query processing in DBMS - javatpoint*. www.javatpoint.com. (n.d.). Retrieved from https://www.javatpoint.com/query-processing-in-dbms.

[41] Kim, J. (2014, December 5). *How sharding works*. Medium. Retrieved from https://medium.com/@jeeyoungk/how-sharding-works-b4dec46b3f6.

[42] *Sharding*. Hazelcast. (2020, July 8). Retrieved from https://hazelcast.com/glossary/sharding/.

[43] Wojcik, M. (2020, July 24). *Scaling horizontally vs. scaling vertically*. Section. Retrieved from https://www.section.io/blog/scaling-horizontally-vs-vertically/#:~:text=Horizontal%20scaling%20means%20scaling%20by,as%20%E2%80%9Cscaling%20up%E2%80%9D).

[44] DBMS indexing in DBMS - javatpoint. www.javatpoint.com. (n.d.). Retrieved from https://www.javatpoint.com/indexing-in-dbms.

[45] *DBMS - Indexing*. DBMS - indexing. (n.d.). Retrieved from https://www.tutorialspoint.com/dbms/dbms_indexing.htm.

[46] *What is a relational database?* Oracle. (n.d.). Retrieved from https://www.oracle.com/database/what-is-a-relational-database/#:~:text=A%20relational%20database%20is%20a,are%20related%20to%20one%20another.&text=The%20columns%20of%20the%20table,the%20relationships%20among%20data%20points.

[47] What is a non-relational database? MongoDB. (n.d.). Retrieved from https://www.mongodb.com/databases/non-relational.

[48] Ultimate Guide to Backend Web Development: Details and Required Skill Set. Temok. (n.d). Retrieved from https://www.temok.com/blog/backend-web-development/

[49] Alexander S. Gillis. *Web server.* What is. (n.d). Retrieved from https://whatis.techtarget.com/definition/Web-server

[50] Aaron. (2020, January 10). *Installing Apache web server on Linux Cloud Servers.* LayerStack. Retrieved from https://www.layerstack.com/resources/tutorials/Installing-Apache-server-on-Linux-Cloud-Servers

[51] Vyom Srivastava. *How to Create an API Using The Flask Framework.* Nordic APIS. Retrieved from https://nordicapis.com/how-to-create-an-api-using-the-flask-framework

[52] Jason Van Schooneveld. (2021, July 28). *Python and REST APIs: Interacting With Web Services.* Real Python. Retrieved from https://realpython.com/api-integration-in-python/

[53] Jose Santa Cruz G. (2019 September, 28). *API calls and HTTP Status codes.* ITNext. Retrieved from https://itnext.io/api-calls-and-http-status-codes-e0240f78f585

[54] Lucia Cavero-Baptista. *Web Application Testing: The basics of web App Test Automation. Leapwork*. (n.d). Retrieved from https://www.leapwork.com/blog/web-application-testing-the-basics-of-web-app-test-automation

[55] Thomas Hamilton. (2021 October, 8). *What is Interface Testing? Types & Example.* Guru99. Retrieved from https://www.guru99.com/interface-testing.html

[56] (2021) *Web Application Testing Complete Guide (How To Test A Website).* Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/web-application-testing/

[57] TestComplete. *What is Automated testing.* (n.d). Retrieved from https://smartbear.com/learn/automated-testing/what-is-automated-testing/

[58] *Git vs. GitHub: What's the Difference.* DevMountain. (n.d). Retrieved from https://blog.devmountain.com/git-vs-github-whats-the-difference/#:~:text=GitHub%E2%80%A6-,what's%20the%20difference%3F,help%20you%20better%20manage%20them.

[59] *An introduction to version.* Beanstalk Guides. (n.d). Retrieved from http://guides.beanstalkapp.com/version-control/intro-to-version-control.html

[60] Supratim Samanta. *Why I choose Insomnia over Postman.* Geek Culture. (n.d) Retrieved from https://medium.com/geekculture/why-i-choose-insomnia-over-postman-fb6690749443

[61] Wikimedia Foundation. (2021, September 18). Johann Carolus. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Johann_Carolus.

[62] Wikimedia Foundation. (2021, October 24). Newspaper. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Newspaper.

[63] Encyclopædia Britannica, inc. (n.d.). The first newspapers. Encyclopædia Britannica. Retrieved from https://www.britannica.com/topic/publishing/The-first-newspapers.

[64] Wikimedia Foundation. (2021, October 16). Gestalt psychology. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Gestalt_psychology.

[65] Wikimedia Foundation. (2021, September 30). Mockbuster. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Mockbuster.

[66] World Leaders in Research-Based User Experience. (n.d.). How chunking helps content processing. Nielsen Norman Group. Retrieved from https://www.nngroup.com/articles/chunking/.

[67] The magical number seven, plus or ... - university of toronto. (n.d.). Retrieved from http://www2.psych.utoronto.ca/users/peterson/psy430s2001/Miller%20GA%20Magical%20Seven%20Psych%20Review%201955.pdf.

[68] Google. (n.d.). Mobile web traffic statistics - think with google. Google. Retrieved from https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-web-traffic-statistics/.

[69] Mobile vs. desktop usage in 2020. / Perficient, Inc. (n.d.). Retrieved from https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage.

[70] Duffner, G. A. (n.d.). EB Garamond. Retrieved from http://www.georgduffner.at/ebgaramond/.

[71] Wikimedia Foundation. (2021, October 19). Bauhaus. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Bauhaus.

[72] Wikimedia Foundation. (2021, October 24). Comic sans. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Comic_Sans.

[73] Wikimedia Foundation. (2021, October 25). Gotham (typeface). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Gotham_(typeface).

[74] Why color matters. Colorcom. (n.d.). Retrieved from https://www.colorcom.com/research/why-color-matters.

[75] Porter, J. (2017, February 1). The Button Color A/B test: Red beats green. HubSpot Blog. Retrieved from https://blog.hubspot.com/blog/tabid/6307/bid/20566/the-button-color-a-b-test-red-beats-green.aspx.

[76] Colour assignment - preferences. (n.d.). Retrieved from http://www.joehallock.com/edu/COM498/preferences.html.

[77] Raza, M. (2021, July 6). In *javascript-barcode-reader*. Retrieved from https://www.npmjs.com/package/javascript-barcode-reader

[78] Fernando, D. (2019, February 8). In *Implementing A Barcode Scanner by Using React Native Camera*. Retrieved from https://medium.com/@dinukadilshanfernando/implementing-a-barcode-scanner-by-using-react-native-camera-b170de4b7f51

[79] Seng, W. (2021, October 19). In *Host react application on Apache server*. Retrieved from https://gist.github.com/ywwwtseng/63c36ccb58a25a09f7096bbb602ac1de

[80] Meel, V. (2021, September 24). *Image Recognition: The Basics and Use Cases (Guide in 2021)*. Viso.ai. Retrieved from https://viso.ai/computer-vision/image-recognition/

[81] Tamir, M. (2020, June 26). *What Is Machine Learning? - I School Online*. UCB-UMT. Retrieved from https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/

[82] IBM Cloud Education. (2020, July 15). *What is Machine Learning?* IBM. Retrieved from https://www.ibm.com/cloud/learn/machine-learning

[83] Image Recognition – What is Image Recognition? (n.d.). Sightcorp. Retrieved from https://sightcorp.com/knowledge-base/image-recognition/

[84] How to build a mobile website. WebFX. (n.d.). Retrieved from https://www.webfx.com/design-build-mobile-web-site.html.

[85] Strachan, J. (2017, December 11). Adaptive vs Responsive Web Design [Illustration]. UXPlanet. Retrieved from https://miro.medium.com/max/2000/1*-3wqJQI4Ou36nqrtlXmjnQ.png

[86] Jain, R. (2005, April 8). 7 Best Practices for Designing a Mobile User Experience. SitePoint. Retrieved from https://www.sitepoint.com/7-best-practices-designing-mobile-user-experience/

[87] Wenzel, F. (2012, June 8). Let's talk about password storage. Mozilla Web Development. Retrieved from https://blog.mozilla.org/webdev/2012/06/08/lets-talk-about-password-storage/

[88] PasswordHash. (2014, August 20). [Illustration]. TheSpreadsheetGuru. Retrieved from https://images.squarespace-cdn.com/content/v1/52b5f43ee4b02301e647b446/1408471418132-DDC0U7GY7QGQX4FXB0UZ/How+Microsoft+Excel+2010+%26+2007+Password+Protection+Algorithm+Works?format=750w

[89] PasswordSaltHash. (2014, August 20). [Illustration]. TheSpreadsheetGuru. Retrieved from https://images.squarespace-cdn.com/content/v1/52b5f43ee4b02301e647b446/1408470758135-QZBH2K5P8FMWGMZ70IAV/How+Microsoft+Excel+2013+Password+Protection+Algorithm+Works?format=750w

[90] Velimirovic, A. (2021, November 12). Strong Password Ideas For Greater Protection. PhoenixNAP Blog. Retrieved from https://phoenixnap.com/blog/strong-great-password-ideas

[91] Password Manager. (n.d.). Malwarebytes. Retrieved from https://www.malwarebytes.com/what-is-password-manager

[92] Armin Ronacher. (2012). Welcome to Flask web development one drop at a time.Read the docs. Retrieved from https://flask-doc.readthedocs.io/en/latest/patterns/packages.html

[93] Armin Ronacher. (2012). Welcome to Flask web development one drop at a time.Read the docs. Retrieved from https://flaskdoc.readthedocs.io/en/latest/blueprints.html#blueprints

[94] Top 6 BEST Python Testing Frameworks [Updated 2021 List]. (2021, November 30). Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/python-testing-frameworks/

[95] Robot Framework. (n.d.). Robot Framework. Retrieved from https://robotframework.org/

[96] RobotFailedTest. (n.d.). [Photograph]. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/Robot-Failed-tests.png

[97] RobotPassedTest. (n.d.). [Photograph]. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/Robot-Successful-tests.png

[98] pytest: helps you write better programs — pytest documentation. (2021). PyTest. Retrieved from https://docs.pytest.org/en/6.2.x/#

[99] Plugin List — pytest documentation. (2021). PyTest. Retrieved from https://docs.pytest.org/en/latest/reference/plugin_list.html

[100] PyTestSoftware. (n.d.). [Photograph]. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/PyTest-1.png

[101] UnittestCommandLineTest. (n.d.). [Photograph]. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/unittest.png

[102] Shah, H. (2021, October 12). Unit Testing vs Functional Testing: A Detailed Comparison. Insights on Latest Technologies - Simform Blog. Retrieved from https://www.simform.com/blog/unit-testing-vs-functional-testing/

[103] DocTestShell. (n.d.). [Photograph]. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/doctest.png

[104] Nose2. (n.d.). Welcome to nose2 — nose2 0.6.0 documentation. Retrieved from https://docs.nose2.io/en/latest/

[105] testify. (2016, August 3). PyPI. Retrieved from https://pypi.org/project/testify/

[106] TestifySoftware. (n.d.). [Photograph]. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/Testify.jpg

[107] GeeksforGeeks. (2020, August 5). Differences between Black Box Testing vs White Box Testing. Retrieved from https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/

[108] behave. (2018, February 25). PyPI. Retrieved from https://pypi.org/project/behave/

[109] Los Angeles Times. (1987, December 16). Coca-Cola changes its stripes, adding a bit of blue to 'new' Coke Cans. Los Angeles Times. Retrieved from https://www.latimes.com/archives/la-xpm-1987-12-16-fi-19563-story.html.

[110] Wikimedia Foundation. (2021, November 21). Web hosting service. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Web_hosting_service.

[111] Guiding principles: Testing library. Testing Library Blog RSS. (n.d.). Retrieved from https://testing-library.com/docs/guiding-principles.