Manifestation Tracker App

The Giving Child

Written by Derek Barker, Madison Long-Markakis, Kai Mizuno, and Amira Ramirez Gonzalez

June 10, 2020

Table of Contents

1. Introduction

1.1 The Giving Child

The Giving Child is a non-profit organization looking to help people and doctors in any way possible during the COVID-19 pandemic. To help both doctors and patients, they wanted to create a Symptom Tracker App that could help people remember their symptoms, and allow for them to share them with their doctors via printouts. The Giving Child has repeatedly been involved with the Computer Science field session at mines, allowing for the opportunity to develop apps for many different projects, from games to now a Symptom Tracking App.

1.2 High-Level Description of the Product:

In the midst of the COVID-19 pandemic, everyone has been told to monitor for symptoms of the virus. But, dealing with a virus that has dormant symptoms and new symptoms appearing regularly, which are often difficult to track, is difficult for anyone suffering from the outbreak. To offer a solution, at the request of The Giving Child, our team has developed a pandemic symptom tracker app for the iOS App Store called Manifestation Tracker. This symptom tracker helps a person that is worried about suffering from COVID-19, or any other sickness, to keep up with all of their symptoms on a daily basis. It does this by giving the user an easy way to log their symptoms, the days they experience them, and the intensity of the experience. There is also a graphing feature that shows how symptom intensities change over time, and the user can print or share this information to show their doctor. The app stays lighthearted and encourages users to keep coming back by rewarding them with new graphics and allowing them to customize their “Mushroom Forest”. Manifestation Tracker is an app designed to give users all the utilities they need to track the manifestation of their symptoms while maintaining a fun atmosphere designed to keep users relaxed and at ease.

2. Requirements

2.1 Functional Requirements:

1. The app starts with a welcome page and a disclaimer page, detailing that it cannot legally give professional medical advice.
2. Capture and record the symptoms and intensity of symptoms chosen by the user, this is done by using a text field, which a user can enter symptoms into. Whenever needed, the user can click on their symptom in the list and add a new day they had the symptom, as well as the intensity of the symptom on that day.
3. Display the symptoms and their duration and intensity on a symptom tracking graph.
4. The user can choose a nature graphic for each of their symptoms. These graphics display in a fun ‘Mushroom Forest’ page that grows with each consecutive day logged in and each new symptom recorded.
5. A rewards system, where the user can earn currency by putting in daily entries for their symptoms. These rewards can be used in the store to purchase new icons and backgrounds.
6. Icons can be attached to symptoms to let the user attach a unique identifier to symptoms and placed as stickers in the user’s forest.
7. The forest’s background can be updated to backgrounds unlocked in the store.
8. A sharing function that allows people to take photos of their forest and information about their symptoms.
9. Includes a symptom dictionary so people unsure of what they have can match the definition of one of the symptoms.
10. Includes a printable format for the symptoms graph, as well as other printable items such as a grab bag list, hospital tips, and an ER printout.

2.2 Non-Functional Requirements:

1. The app is developed for iOS via Xcode and the Swift programming language.
2. The app is user-friendly. Non-technical people are able to navigate and understand the app easily.
3. The app is reliable and without failures at run-time.

3. System Architecture

3.1 Model View Controller Design Architecture

This project follows a design pattern known as Model View Controller (MVC). This design pattern works well for this project because it is meant for user applications such as phone apps. The design revolves around splitting the code and files into three sections: models, views, and controllers. Models are the data being stored for the user, views are the images being displayed to the user, and controllers navigate the view and connect it with the data by controlling the flow of the app.

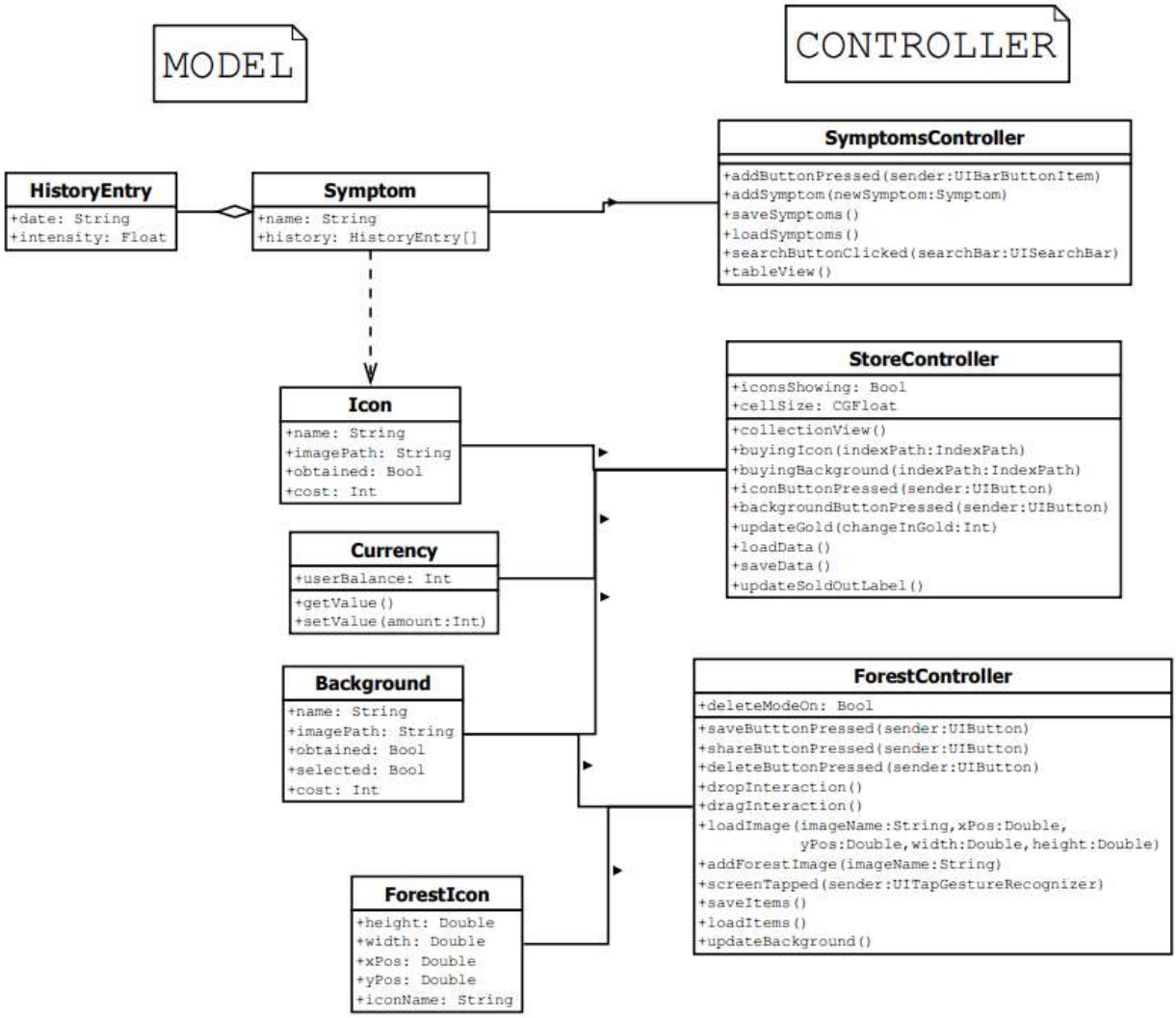


Figure 1: Model and Controller UML

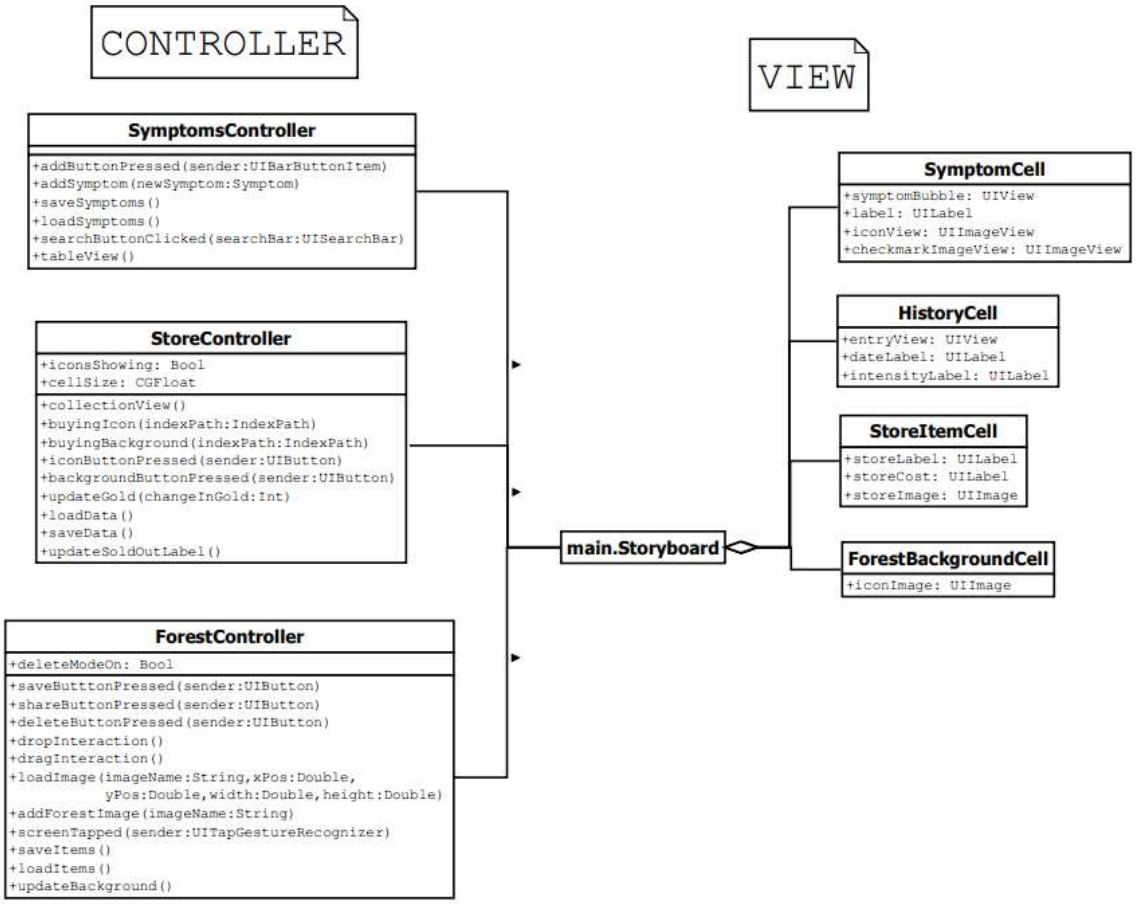


Figure 2: Controller and View UML

Figure 1 shows a UML design for part of the project, which shows how the model and controllers are set up and interact. Classes in the model are on the left side of Figure 1. The model is made up of data that needs to be stored for the user. There exists information that shouldn't be reset even if the user shuts down the app and restarts it. All this information gets stored as classes in the model.

Controllers act as the brains of the app. These classes are shown on the right side of Figure 1 and each have 'Controller' in their class name. They determine the points in time when it is necessary to retrieve data from the model, or save new data. Each scene in the app contains a corresponding controller that connects the model to the scene's view. For example, there's a tab in the app that shows the symptoms that the user has entered. The controller fetches the stored symptoms data, and sends it back to the view. The model and view never interact with each other, instead using the controller as an intermediary.

Figure 2 shows a UML design for how the controllers interact with the views. The views are made up of storyboards and scenes that the user sees as they use the app. Views contain interactables such as buttons and tables. Tables can contain custom mini-views from xib files, which are built off custom classes. For example, Figure 2 shows that there is a SymptomCell class within the view. Within the symptom tab there is a table that displays all the user's symptoms, and the SymptomCell class determines how each cell appears.

The view itself does not know what to do when a user interacts with the app. It becomes the job of the controllers to continue to act as the brain and determine how the view should update in reaction to the user. The UML shows that many of the functions contained in the controllers are ones that are called in reaction to a change in the view, such as a button being pressed. Controllers get signaled when these events occur and determine the course of action to take afterwards. For example, after the search button is pressed in the symptoms tab, the SymptomsController calls its searchButtonClicked function, which gets the data from the model corresponding to the search, then tells the view which symptoms to display based on the search.

3.2 Storing Data with a SQLite Database

The model needs to be stored somewhere in order to access the data later. Apple's CoreData stores a SQLite database on the user's device, allowing this app to manage and store all the information necessary while keeping the user's privacy secure. Everything in the model gets stored as entities in a SQLite database stored on the user's personal device. As objects get created in the code, they get saved to the database by the controllers. The information is not sent over any networks and is hosted solely on the user's own device.

4. Technical Design

4.1 SQLite Database Design

The SQLite database is the portion of the app that handles most of the saving functionality. The tables within the database were created, and follow the Entity Relationship Diagram (ERD) shown in Appendix A. Most of the information regarding the state of the app gets stored in the SQLite database. First, the starting data gets loaded into the database on the first boot of the app. After it is saved, the controllers request any information needed, like information regarding the Forest Icons available for purchase, or if an item was placed, as well as where it was placed at. Figures 3-6 all show a relation between the pictured entity and the user entity.

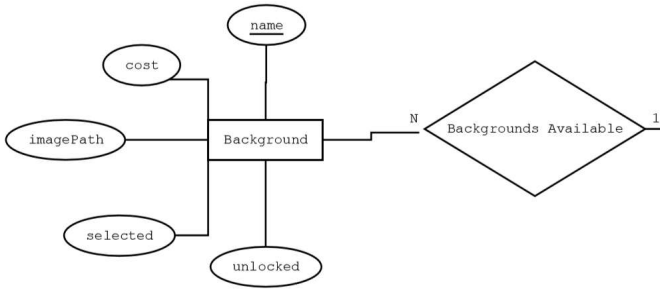


Figure 3: Background Relation

At the center of the ERD is the user entity which acts as a bridge between all other entities in the ERD. To the left of the user entity is the Background entity, shown in Figure 3, this tracks the name of the background, the image path, if it is unlocked, if it is being selected, and its cost. Every possible background is stored in this Background table. This has a one to many relation with the user, as there are many backgrounds for one user. Each background has the booleans ‘unlocked’ and ‘selected’. Only backgrounds that have unlocked=true are able to be selected in the forest, and there is always only one background with selected=true, which indicates which background the user has selected for their forest.

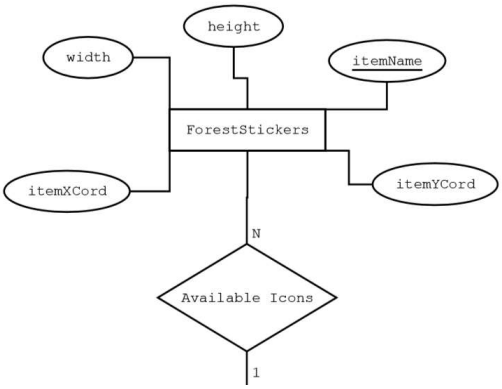


Figure 4: Forest Stickers Relation

Above the user entity is the Forest Stickers entity, shown in Figure 4. This deals with all of the information regarding the position of the icons that have been placed and can be moved in the forest. This in particular stores the name of the icon being stored, its position in the x and y axis, as well as the items width and height. This also has a one to many relation with the user, as the user can put as many icons as they want onto the screen.

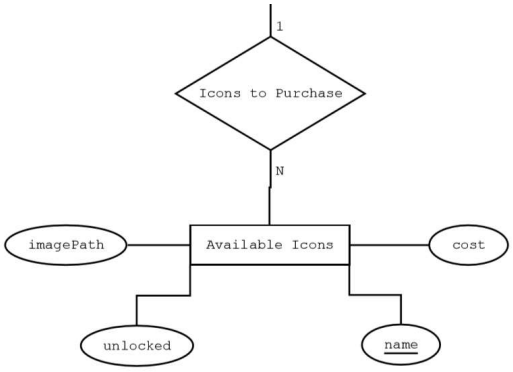


Figure 5: Available Icons Relation

Below the user entity is the Available Icons entity, shown in Figure 5. This information deals with the icons available for purchase in the store. It stores information about the icon's name for display, the image path for the icon, the cost of the icon in the store, and if the icon is unlocked or not. Similarly to the previous two entities, this also has a one to many relation with the user, as the user has access to all of the icons.

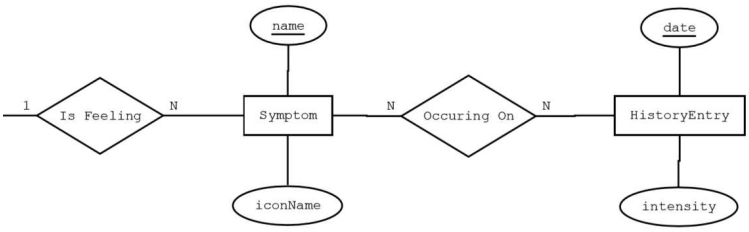


Figure 6: Symptom-History Entities

To the right of the user entity is the Symptom entity, followed by the HistoryEntry entity, shown in Figure 6. These deal with storing all of the information about what symptoms the user has logged. The Symptom entity stores the name of the symptom as well as the icon that the user has associated with the entity. The HistoryEntry entity stores the information regarding the date of the symptom entry as well as the intensity. The Symptom entity has a one to many relation with the user, as the user can log as many symptoms as they want, and the HistoryEntry entity has a many to many relationship with the Symptom page, as many symptoms can have many logged entries.

4.2 App Wireframe

Manifestation Tracker has a tabbed layout with the ‘Your Symptoms’ page set as the default home page, depicted by Figure 7 below. Here, the user sees a current list of symptoms with options to add a new symptom or view graphs based on recorded entries. When adding a new symptom, the user is asked for the name and is offered the option to change which icon represents it. To add entries, which are composed of a date and intensity on a scale from 0 to 10, one can click on the symptom after it has been added and a history of the entries will come up. The graphs display plots intensity versus time graphs of all symptoms with the option to change which symptoms are visible as well as change the time interval being viewed.

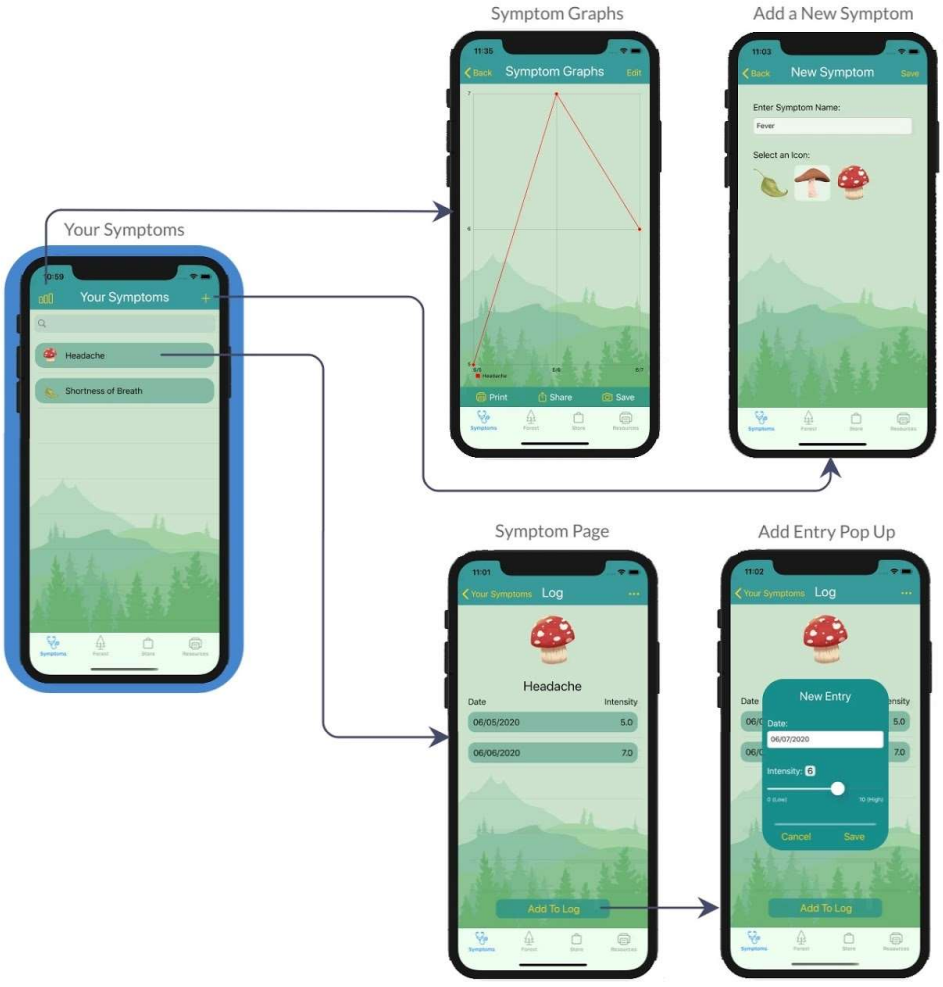


Figure 7: Symptom Tracking Wireframe

The other three main tabs are the forest page, the store, and a resources page, the corresponding wireframe is depicted in Figure 8 below. The forest tab navigates to a display of forest scenery which the user can customize by changing backgrounds and placing the various icons they can unlock. The store page is where these unlockable icons and backgrounds can be purchased with in-game currency that is earned by regularly visiting the app and recording entries. Lastly, the resources tab provides a link to a symptom dictionary in addition to other useful and printable PDFs such as hospital tips.

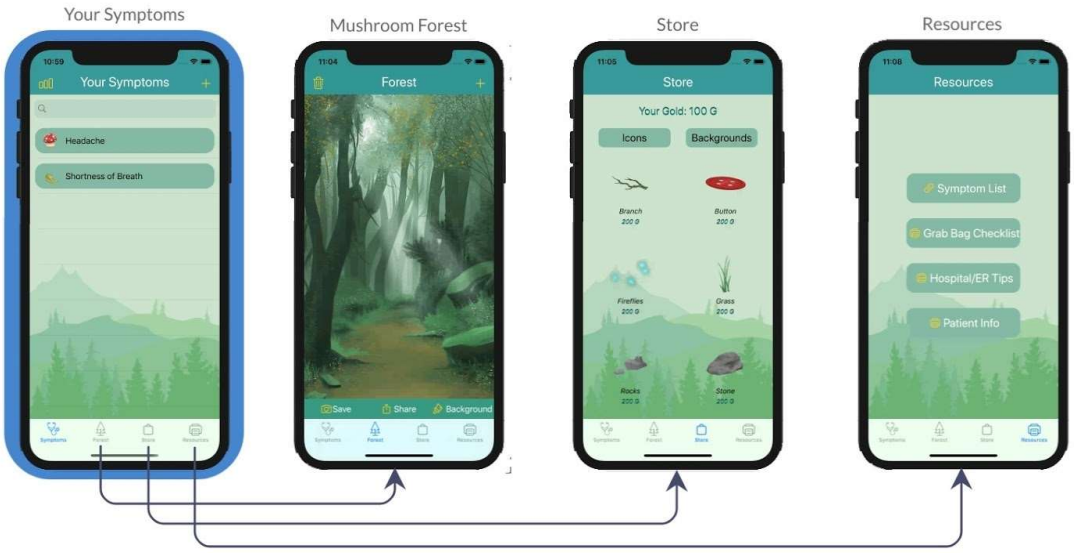


Figure 8: Tabbed Layout Wireframe

The team aimed to create a simple UI that matches Apple's preferred style by utilizing a restricted palette, limiting the number of tabs and keeping components nicely spaced out, aligned, and balanced. The green tones were chosen to ease the eye and create a calming feel. More details on the background options as well as all the icon graphics can be found in Appendix B and the fully connected wireframe can be found in Appendix C.

5. Quality Assurance

5.1 Automated Testing

This app uses automated testing in the Xcode project. This helps to maintain the project's quality because it required the team to continuously think about edge cases in the code while testing to ensure these edge cases are handled properly in an efficient manner. Automated tests for iOS development can be broken up into two categories: unit tests and UI tests.

Xcode allows new projects to be initialized with a connection to a unit test framework with XCTest.

XCTest is the unit test framework used by Xcode that allows for unit testing immediately upon initialization of a new project. This framework was used to test the edge cases on how the data in the model is being handled. For example, the project contains unit tests to ensure that the expected graph is displayed under every scenario of combinations of symptoms and histories. These tests ensure that our implementation for the model and how the controllers interact with the model stays consistent and without errors.

XCTest also contains functionality to create UI tests for Xcode projects. These tests allow developers to simulate a user interacting with the app and test whether or not the expected app flow occurs. These UI tests are used to ensure that each of the tabs in the app interacts with each other correctly. For example, this project contains tests that simulate the app flow when a user tries to create a new symptom and ensure that the expected outcome occurs when the user interacts with each item in any given order. These tests ensure that the view and controllers are behaving as expected and that the storyboard is staying consistent throughout development.

5.2 Manual Tests

The team also ran the application through user tests. Receiving feedback from real users helped assure the project's user experience is as desired, in terms of both validating how attractive our layout is for users and evaluating whether the provided features meet the targeted user needs. Specific test goals tested how effective the project's design is at keeping users using the app regularly or test how intuitive navigating the app is to easily track symptoms on a regular basis. Given the circumstances, tests were done with the help of family members and other people close to the team members or done remotely.

In order to test the general characteristics of the application, the team created a test checklist. This includes items to manually check by running the app. The test items included characteristics such as how long the application takes to install and to launch, check that the icon and app name display properly, closing the application maintains the expected status, relaunching the application saves appropriate data, testing saved data with the various ways to exiting an app, and so on. Team members went through the items on this checklist before committing another change to ensure that all the key components of the app are still behaving as expected.

5.3 Code Design

The code also needs to be maintainable and expandable by future developers. This is achieved by following the Model View Controller design architecture described in Section 3.1. This architecture is a staple for app development, favored even by Apple themselves. This makes looking at the app easier for people familiar with the design as they can hop right into what they want to change and change it, instead of looking for the correct class and code segment for extended periods of time. This also helps with people trying to expand on the code, as they have all of the base code that we created to split up and can easily add another model, view, or controller to expand our design. Following this design ensures that the code stays consistent and easy to understand.

The project's code also has to ensure the security of the data. This app stores information about the health of the user, and it cannot be guaranteed the user would be comfortable letting a company keep this information in the cloud. This issue is solved by storing data using CoreData and a SQLite database, as described in Section 3.2. All of the data stored for the user is hosted solely on the user's own device. No company or individual has access to this information without the user's intent to share it. Apple prides themselves on having secure devices, so the user's information should stay safe locked in the files of their own device.

5.4 Code Reviews

The team split the tasks amongst its team members, so there was the possibility that members diverged in their approaches to the code's structure. The team conducted code reviews to ensure a cohesive and high quality product was delivered to the client. Team members went through each others' code together to ensure that the code functioned as intended and made sure that the coding practices made sense. One of the primary focuses of these code reviews was to ensure that the design was following Model View Controller design practices, and code was refactored to better follow the established design. Code reviews also helped to ensure proper commenting and naming conventions were followed.

6. Results

6.1 Unimplemented Features

All of the main features that the client requested to be in the app at the beginning of the field session are included in the final product, and most the stretch goals laid out have also been included. Of the stretch goals, there were only a few features that we were not able to get to by the end of the field session.

- Daily Notifications - The process of getting the notifications working was too large and time consuming for us to implement in the time frame of this project.
- Rotate/Resize Stickers - We did not have the time to make a workable implementation in our code, but created the foundational requirements in the code for future expansion.
- Background Music - This became infeasible due to our time limitations and inability to create original music.

6.2 UI Tests

The Xcode library, XCTest, was used to develop user interface tests for this app. The goal of these tests was to make sure menus, buttons, icons, alerts, and other UI components appear when and where appropriate. Tests have been implemented for each main tab of the app which are the forest, store, symptoms, and resources tabs. Some of the tested aspects include:

- Buying icons and backgrounds
- Creating a new symptom
- Creating, editing, and deleting entries
- Checking all resources buttons are present and PDF resources have a printing option
- Launch performance on a simulator

These tests focus on detecting any display errors, for example, when buying from the store do popups confirming a purchase appear and do the correct icons show up in the forest tab's inventory. All of the tests pass, no inconsistencies between the expected display and actual display have been found; however, they have brought attention to details that the team could improve on such as not allowing duplicate symptoms. Launch performance is fairly good, using the simulator gave an average of as low as 1.45 seconds and up to only 3.72 seconds average.

6.3 Unit Tests

Unit tests allowed us to check edge cases for the functionality of our model. Each unit test made new instances of objects contained in the model and tested the helper functions of those objects to ensure they were behaving as expected. Some of the parts tested include:

- The creation of a new symptom
- The creation of a new entry in a symptom's history
- Updating the user's currency
- Receiving rewards for daily entries.

All of these tests were created to ensure that the behavior of the model was working as expected in every edge case. All of the tests pass and meet established expectations. There were a few points where developing the unit tests helped point out existing issues that were not accounted for. For example, testing receiving rewards for daily entries had many edge cases to consider due to there being many different points in time the user could be logging in a new entry. The tests pointed out that there was an issue where the app was crashing when more than 48 hours had passed since the last entry and the app tried to reset their streak. This edge case was fixed quickly.

Unit tests are also able to test the performance of parts of our code. XCTest allows developers to run code blocks through time measurements and find the average speed of the app. The project also contains unit tests to check the speed of the functions being tested from the model. The tests showed us the code that has been developed is not very intensive and takes an average of about 0.1 milliseconds to run each function.

6.4 Usability Tests

To ensure quality performance and ease of access, the team tested a variety of users' interactions and opinions of the app. A graduate of the user experience design field provided feedback on a professional scale, and other testers of different ages and backgrounds shared suggestions for improvement through a Google form. The aspects checked by these tests include:

- Ease of dragging forest graphics
- Comfortable flow of functionality
- Applicable features for any user of any medical background
- Effective incentive to keep returning to the app
- Enjoyable features
- Consistent visual style

The first tests resulted in mixed reactions, most importantly that the app was too complicated for users. If the app's purpose relied on users logging in daily and updating existing symptoms, the version of the app that was tested was not simple enough to warrant daily logins. Additionally, the instructions for the app functionality itself were unclear and users were unsure of what to do. Thus, the appropriate changes were made to ensure these shortcomings were addressed. A text-based tutorial was added upon the first launch of the app and first-time visits to each tab, and some of the controls were condensed and expanded to make the process of logging symptoms quicker and easier.

6.5 Future Work

The code has a simple structure to ensure future modifications and allow for additions without breaking any of the existing tests. It should be easy for anyone editing the app to add more graphics to the pool of forest visuals and to add any future functionality. Ideally, the app will be simple for users to continue to use after the COVID-19 pandemic has passed for any medical worries and symptom tracking regardless of the illness, as we have tried avoiding naming the COVID-19 virus specifically as the only purpose for the app. Another project that could stem from the work done so far is to create a port of this app for Android, allowing even more users to be able to use the app.

6.6 Lessons Learned

- We learned how important it is to let team members specialize in certain areas when delegating tasks. Development of this project got split into two categories: coding and graphics. We assigned two people to each category. If we had tried to split the work in both categories evenly, the project would likely not have been finished. Letting each person specialize in a certain area kept the project consistent and discouraged members from wasting time on doing something that someone else could do more efficiently.

- Nobody on the team had experience working for a client before. This process forced us to determine good communication methods and planning methods to ensure the client best understood our process and progress. For example, we learned how valuable storyboarding can be in the development of apps. The first step of this project was developing a plan for the GUI through storyboards, which allowed us to immediately be able to delegate development tasks and get feedback on what to change. Without thorough planning, we would have been far less time efficient and likely would not have been able to put a finished product on the app store.

- We learned a fair bit about the process of app development in swift. None of the team members had any background in iOS development before this project. This has forced us to learn new skills and hone our ability to find useful resources.

- Most of the team did not have access to a physical Mac to program on. Xcode can only be run on MacOS, so this was a big problem. Many members of the team were forced to use a program called MacInCloud to remote into a MacOS to program with. This was a difficult process because MacInCloud can be a laggy and unreliable service. Due to these struggles, we learned how important it is to have the right hardware when going into a project. Quick fixes to hardware issues can cost time and quality during development.

- We ran into some issues when putting the app on the App Store, mostly due to communication issues between our team and the app reviewers. We were initially rejected by the review team with a comment saying “your entertainment or gaming app inappropriately referenced COVID-19”. After explaining to them that it is not an entertainment or gaming app and it never explicitly references COVID-19, we were able to get it onto the store. It was a time consuming process that taught us to be patient with the App Store review team, and to be more clear about the goals of the app when submitting for review.

Appendix A

Entity Relationship Diagram

The following diagram is Figure A-1. The app uses a SQLite database to store a majority of the information regarding the app's current state.

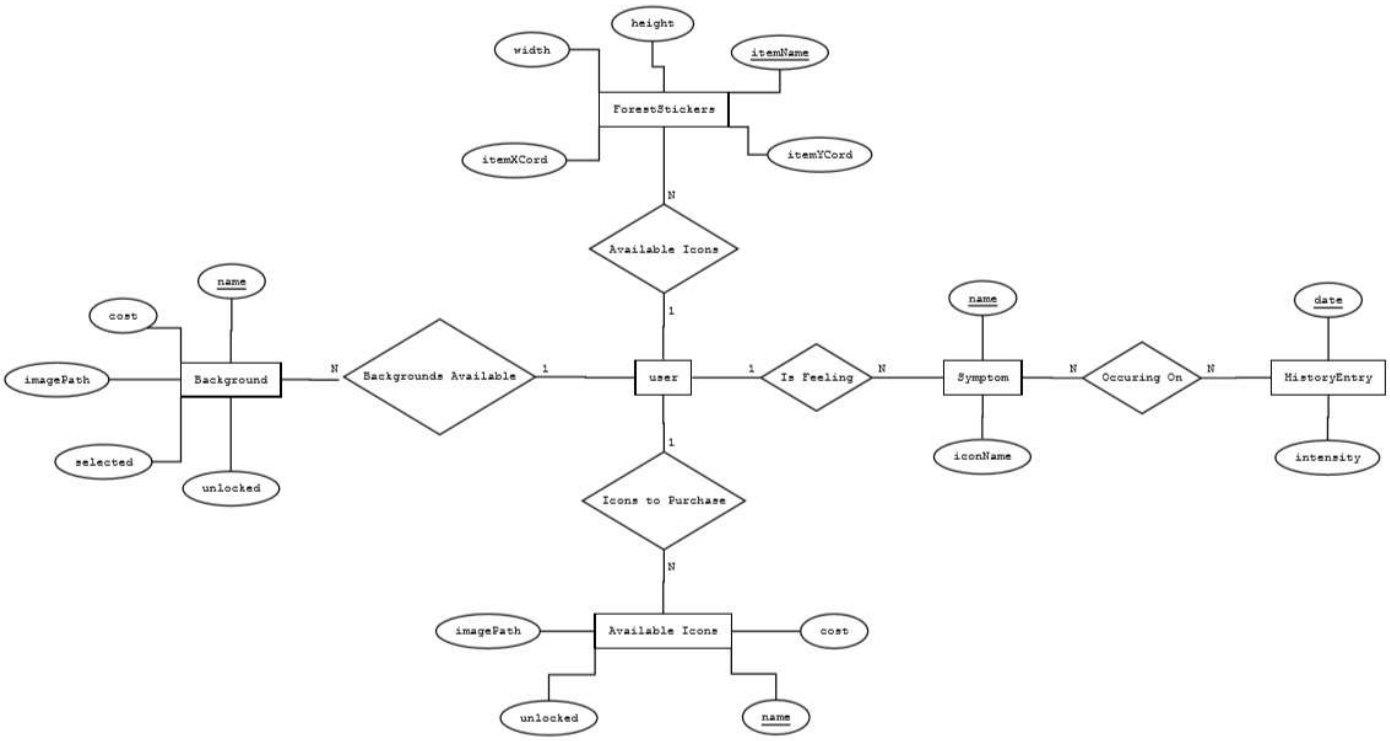


Figure A-1: Entity Relationship Diagram

Appendix B

App Graphics

The following images are symptom icons (Figure B-1). The design of the app focuses on a whimsical forest theme, allowing players to populate a forest with stickers obtained by logging symptoms for currency and using it to trade for an object in the in-app store. The style of the final designs is lineless and detailed with effort placed on making the objects feel appropriately placed in the forest and are easily recognizable on their own. These designs include many different mushrooms, some small animals, and other objects and plants. These designs are also available for associating them with a symptom the user has logged.

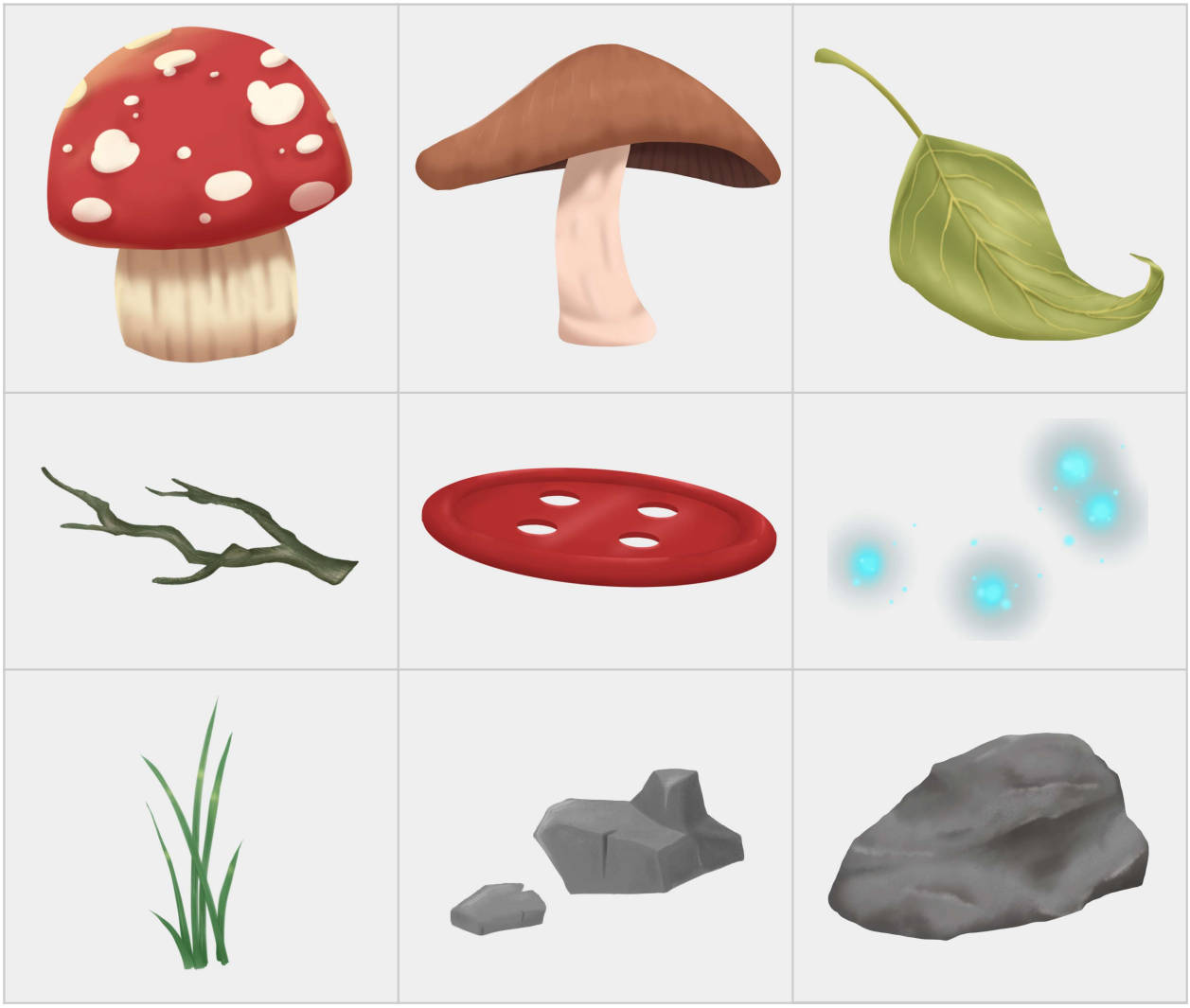
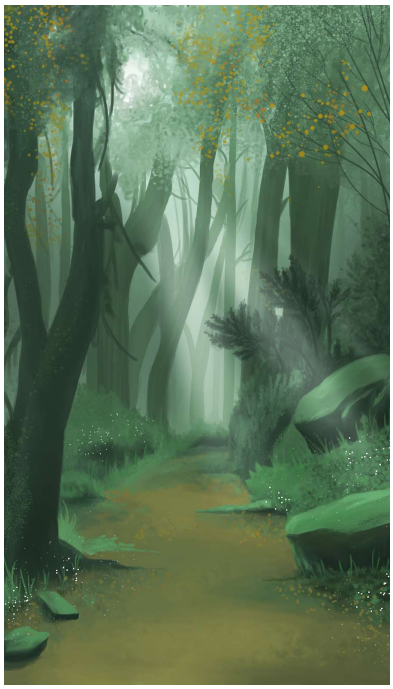


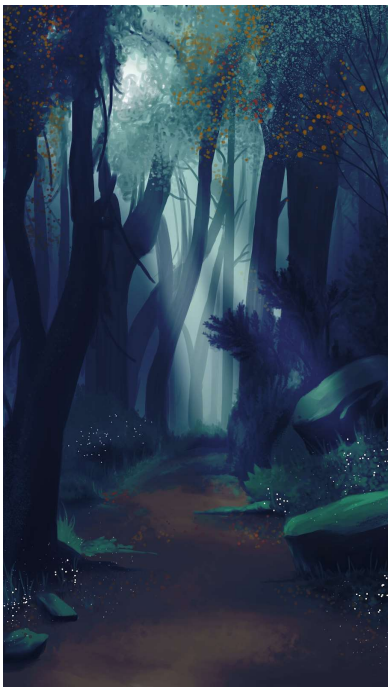
Figure B-1: Icon Graphics

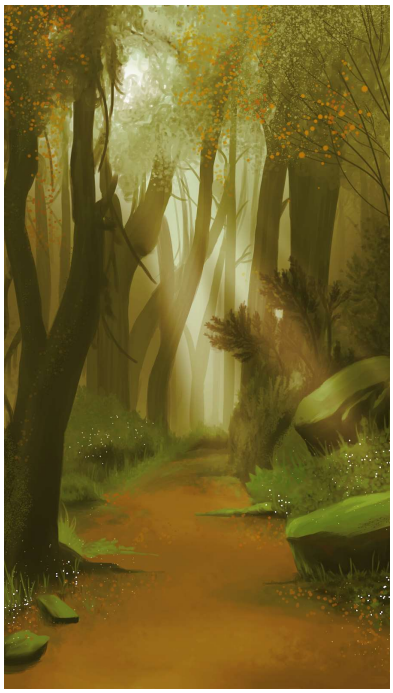
The images below are the available backgrounds. The first, in Figure B-2, is a basic background found in most of the application's tabs, and is a free option for the forest tab.

Figure B-2: Simple Background

The rest of the backgrounds, Figure B-3 below, are only for customizing the forest tab. The image in the top left of the figure is the default forest background while the other variations can be unlocked in the store with in-app currency.







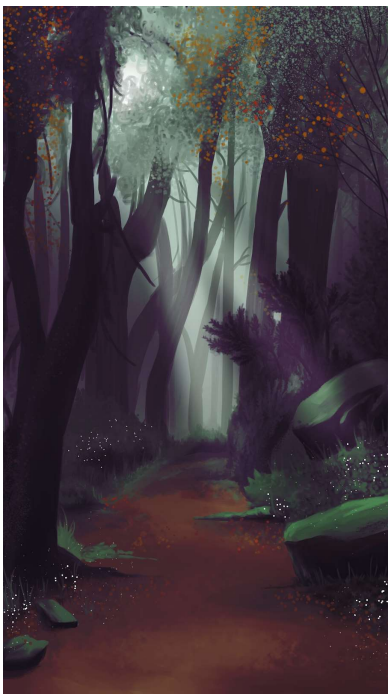


Figure B-3: Forest Backgrounds

Appendix C

App Wireframe

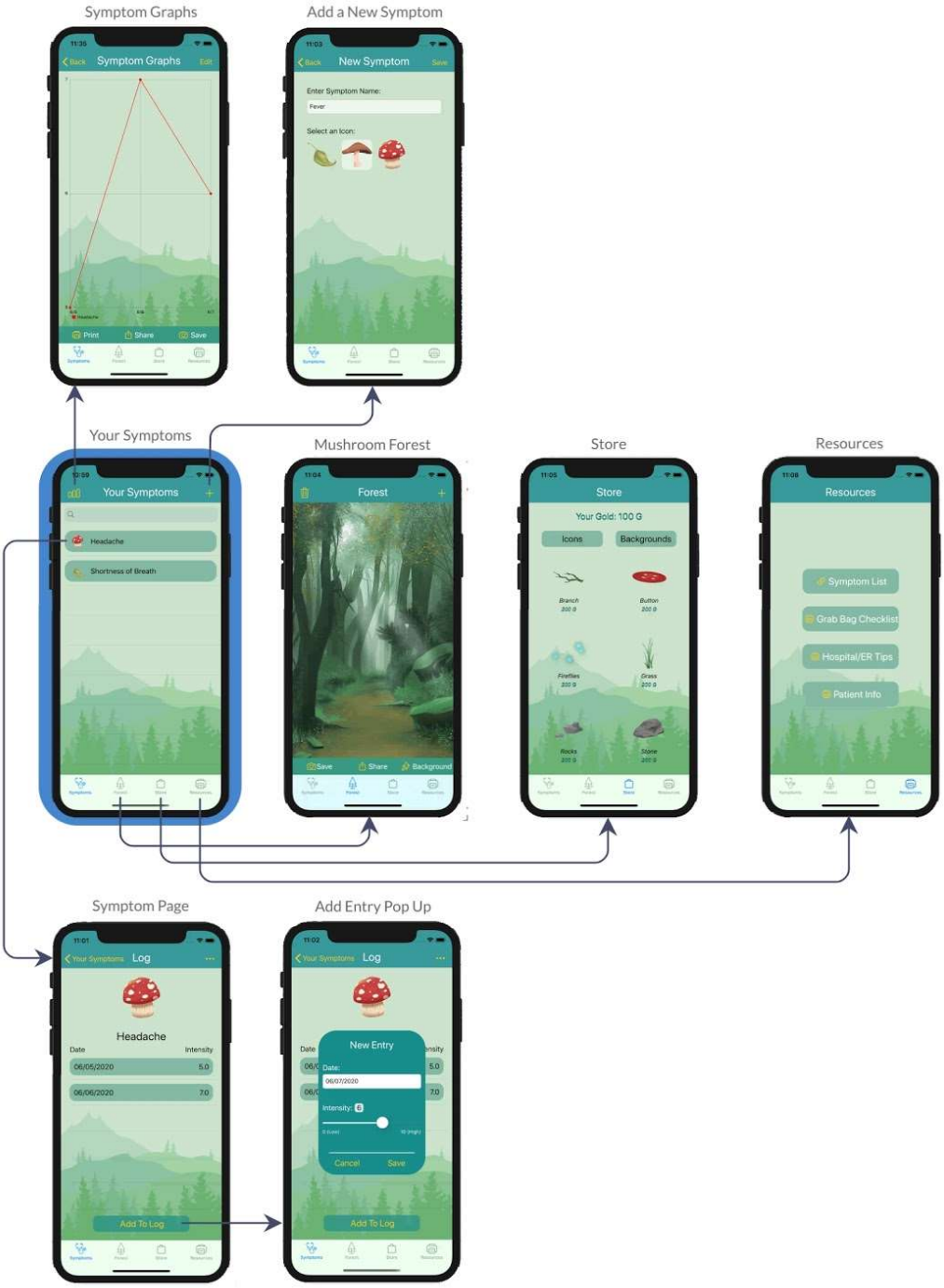


Figure C-1: App Wireframe