Senior Team Portfolio ESOF 423

By Josh Harvey, Rory Schillo, Alex Maliziola, Jacob Rivers and Beau Baer

Section 1: Program:

You can easily download the current version of the program from our github page at: https://423s24.github.io/Group_1/

You can view and download our most up to date codebase and the .exe file we created to run it at: https://github.com/423s24/Group_1/releases

This program is an open source Java desktop app designed for use in connection with a configured Firebase Database. Running the code requires an up to date installation of Java. Additionally, there have been issues running the software on Macs, so we strongly recommend using this software exclusively on windows devices.

If there are any questions about implementation concerns or potential bugs and issues, please feel free to add an issue on our repo's Issues page at the following link: https://github.com/423s24/Group_1/issues

Section 2: Teamwork:

This program was created by our development team as the main course project for the CSCI_423 class in the Spring of 2024 taught by professor Daniel Defrance. Each member of our team contributed to the success of this project in different Capacities depending on their skillset and preferences. We did collaborate and troubleshoot together, and our Backlog does not represent the total sum of all the work put into this process, but each member has below given an estimate of our time, as well as a basic overview of our primary contributions and responsibilities.

Team Member 1

My Role:

Start: At the start, I mainly focused on research and planning. There was a lot of uncertainty around what we should prioritize and what technologies we should use. I spent a lot of time studying different potential options and reported my results to the

group so that we could all make informed decisions about what was best moving forward. Once we started to settle on a few ideas, I made some rough drafts of UML diagrams, UI mock-ups in Java Swing, and tested out some database connectivity. Once major contribution I made at this stage was getting early database connection working proving its feasibility using Firebase and paving the path forward.

Middle: Once we decided on what we wanted our product to look like and what technology we were going to use to achieve it I began working heavily on the backend in order to get the database system fully functional. At the time this was the top priority as without a database our product could not store any data and would be useless. Building the database system revolved around planning the JSON tree structure and creating a system to make API requests to read and write data from Firebase. One major contribution I made here was HTTP streaming which allowed our desktop application to listen for updates from the Firebase server asynchronously. This means that if one user updates the database all other users will get updated as well.

End: After finishing work on the database systems, we got to a position where not much else needed to be done on the backend. Instead, the most pressing concern became the UI and making sure that every resource the HRDC needed access to was present and easy to use in the UI. Because of this I switched to fleshing out the UI and designed the guest overview panel and bunk assignment page. I consider backend to be my strong-suit, so it took me a little bit to get comfortable using Java Swing. However, I enjoyed the challenge as good programmers should be able to adapt as needs change. The last thing I did was connect the bunk assignment page and bunk reservation panel to the database and made sure all the data was properly synching.

Backlog Items

Completed Solo:

Finalize JSON Tree with all necessary elements – Estimate 10 – Actual 10

Get HTTP Streaming working – Estimate 6 – Actual 4

Add list of bunks to database – Estimate 1 – Actual 1

Create bunk assignment page – Estimate 3 – Actual 6

Create bunk editor popup – Estimate 3 – Actual 4

Create guest overview panel – Estimate 8 – Actual 10

Create guest reservation panel – Estimate 2 – Actual 4

Connect bunk assignment page to database – Estimate 6 – Actual 5

Heavily Contributed:

Create backend interface for interacting with Firebase API – Estimate 10 – Actual 8

Decide on JSON tree structure format – Estimate 3 – Actual 3

Decide how Database connections work – Estimate 3 – Actual 3

Moderately Contributed:

Decided on technology and architecture – Estimate 6 – Actual 7

Create UML class diagram – Estimate 2 – Actual 1

Team Member 2

My Role:

I have largely focused on developing the backend architecture and the connector class that interfaces with the database. I am functionally speaking the expert on the DBConnector class, having written and documented it myself with some overlapping assistance from Rory Schillo.

As far as the project backlog goes, I worked on creating the backend interface for the firebase API. I also worked on the UML Diagram for the classes, the JSON tree structure, Deciding the methodology of database connection, Updating dbconnector, creating guides for dbconnector, extending the db structure, and finalizing the JSON tree.

In addition to the backlog items below, a lot of my input on the project has been patching bugs errors and troubleshooting different elements, as well as expanding existing documentation. I did not wish to retroactively change these elements, and most of them were not discussed in our sprint meetings as they came up spontaneously, but I did put in significant additional time working on elements that were not in our backlog.

Backlog Items

Completed Solo:

Get initial Java Project Hosted on Github (Estimated: 10, Actual: 10)

Set up Internal JSON Structure (Estimated:2 , Actual:2)

Create Class to update and pull from the database easily (Estimated: 5, Actual: 14)

Update DBConnector to be used by everyone (Estimated: 15, Actual: 15)

Documentation Comment Coverage (Estimated: 5, Actual: 5)

Heavily Contributed:

Decide How Database Connections work (Estimated: 10, Actual: 15)

Team Member 3

My Role:

I have been one of the frontend designers for the duration of this project, as I rather dislike backend work. I've generally been the main individual designing and developing the various UI panels in our program, though as the backend has reached its final form other group members have stepped in to assist and divide work.

Backlog Items

Completed Solo:

Designed the final look of the windows that were eventually implemented into our software (3 estimate/7 actual)

Created prototype wireframes of our UI (5 estimate/5 actual)

Heavily Contributed:

Scouted out the HRDC Warming Center to get an understanding of exactly what they needed (2 estimate/50 actual)

Created the UI for the Check-in Page (10 estimate/30 actual)

Created the UI for the Guest Details Page (7 estimate/25 actual)

Documentation work (3 estimate/10 actual)

Team Member 4

My Role:

I've been a sort of middleman developer, responsible for connecting UI elements to the backend and writing functionality for them. I made some design pattern choices for the program that had a positive impact on code writability, such as using singletons and focusing on an event-driven architecture. I was also responsible for creating all the github releases and building the installer applications for windows.

Backlog Items

Completed Solo:

"Get releases downloadable" – Estimate 3 – Actual 3

"Make sure usage instructions INCLUDE updating java" – Estimate 3 – Actual 1

"Set up product website - GitHub pages" – Estimate 10 – Actual 6

"Security Research - Token storage" – Estimate 5 – Actual 5

"Autocomplete guest entry" – Estimate 5 – Actual 5

"Functionality- Connect Backend for Guest Details panel" – Estimate 40 – Actual 50

"Usability- Create installer application" – Estimate 15 – Actual 50

"Need a way to set the Firebase "Client Secret" Key" – Estimate 20 – Actual 15

"Create add, remove, and edit functionality for the Disciplinary Tables" – Estimate 10 – Actual 60

Heavily Contributed:

Create Check-In Roster UI and connect to the database – Estimate 75 – Actual 75

Team Member 5

My Role:

I have taken a hybrid role in the group, filling in wherever there is work to be done and usually playing off of other members' contributions. I have done lots of work gathering information and setting out plans of attack for coordinating certain features so others can take that work and elaborate on it. Early in the project I took on lots of research on the tools and infrastructure we are using and made sure they were suitable for the project. I also have been in charge of testing and implementing continuous integration on our project repository. Another role I have taken on is that of a go-between with the client. I have been in regular contact with the client getting lots of questions asked and answered

when we needed clarification or just to adjust and understand the scope of our project. I was the one to organize the field trip and spend the most time doing in person research at the warming center.

I also have made significant contributions to the HMIS+CS reporting and check in features of our application. I also did a lot of bug fixes on our final release.

Backlog Items

Completed Solo:

Created JUnit Testing Framework - 20/25

Final documentation review and editing - 15/10

Field Trip - Organize visit to Warming Center - 5/5

Client Communication - contacting, emailing, and live meetings with client to refine product goals and functionality - 25/35

Implemented Continuous Integration on GitHub - 15/40

UI - set window maximized by default - 1/1

Create data page to assist manual entry - 15/15

Heavily Contributed:

Field Trip - Visit Warming Center - 15/20

Field Trip - Interview employees, diagram workflows, synthesize notes to assess main product goals - 25/25

Check-In needs to be date specific (I.E. selecting a new date should start a new roster) - 40/40

Created actual roster objects and check in times separate from the guest info - 30/20

Create way to set DB secret key from within the app - 5/5

Create views to query and display guest data - 20/30

Moderately Contributed:

Creation of UI mockups and designs - 15/10

Section 3: Design Pattern:

We utilized the Adapter Pattern as a core element of our design. Our api of choice is the firebase API, this tool was incredibly helpful as it is affordable and easy to access using Java.

However, the only key utility firebase provides is pulling and pushing a json file. This was challenging as we needed to implement some complicated data relationships and interactions, so for that reason we created a local database object called Database that would be more workable.

In order to connect these two systems we created an adapter class called the DBconnector and used it to translate the json into the more workable Database class and then back to JSON for upload.

The current working version of this code is in the file DBConnectorV2.jave file which can be found at Group_1/Project_Files/warming_hut/src/main/java/sssp/Control/Helper. The Database class is in the Database.java file in the same folder. The entire DBConnectorV2 class serves as the adapter in our project.

Section 4: Technical Writing:

Below is a copy of our full documentation:

HRDC Database Manager User Manual

Table of Contents

Introduction

Welcome to the user manual for the HRDC Database Manager, an all-in-one program for managing user information and database logging at the HRDC Warming Shelter! This document contains all relevant information required to install, operate, and edit the functionality of the Database Manager to peak efficiency across the Warming Shelter's services.

System Requirements

Before installing the HRDC Database Manager, ensure that your device meets the following requirements:

- Operating system: Windows
- Hard disk space: 73.9 MB
- Stable Internet Connection
- Latest Java Release

Java updates often; when encountering issues, ensure that you have installed the latest Java release to your system before submitting a bug report.

Installation Guide

Please follow these steps to install the HRDC Database Manager:

1. Follow the **Download Dataview** link on our product website or go to the downloads page
2. Navigate to the latest stable release, marked with a version number in a [vx.y.z] format
3. Open the **Assets** dropdown and click the .jar file within
4. Save the .jar file to a location on your computer that you can navigate to later

User Interface Overview

The HRDC Database Manager is separated into three sections, handling different departments; these sections can be navigated between using the button panel on the left side of the window.

Section overview

- The **Check In** tab contains functionality for guest check-in, including name/date logging, user warnings such as No Trespass orders, and basic user info at a glance
- The **Bunk Assignment** tab contains functionality for bunk assignments, including an automatically-updating bunk list, reserved bunk handling, and bunk list editing
- The **Guest Details** tab contains full information about guests, including disciplinary history, storage info, and user notes, and features search functionality for guest names

Usage Guide

Running the HRDC Database Manager

To run the HRDC Database Manager, double-click the .jar file obtained from following the installation guide. In a few moments, the Database Manager will open and be ready for operation.

General Usage

Use the mouse to navigate the Database Manager; tabs can be switched by clicking the large buttons on the side of the screen. When typing in text boxes, the **Enter** key can be used as an alternative to clicking the **submit** button.

Check-in

Check-in handles checking in guests at the front desk; to add a new guest, set the date (it will default to the current date) and type in the guest's name. For ease of use, the **Enter** key can be used to seamlessly add a guest and continue typing the next guest's name with minimal downtime.

Bunk Assignment

Bunk Assignment handles assigning checked-in guests to bunks. The list in this section pulls its information from check-in; next to each guest entry, select the relevant bunk letter (A or B) and then choose the bunk number. Occupied bunks will automatically remove themselves from the dropdown menu; reserved bunks are highlighted in yellow. To add or edit bunk numbers, click the **edit bunks** button. This will open a screen where bunk numbers can be edited, locations can be changed, and new bunks can be created.

Guest Details

Guest Details contains every piece of pertinent information about a guest. Guest info can be accessed by double-clicking their name in the check-in window; alternatively, one can search a guest by name using the field at the top of the Guest Details menu. All information is auto-populated by the database if available; otherwise, editing the values is straightforward depending on each input field type. For checkboxes for item rentals, a date selector appears when each item is checked, allowing for check-in dates to be specified.

HMIS + CW Reporting

This panel contains checking info for reporting to HMIS and CW. As this manager does not have support for these databases, data reporting into them must be completed externally; to support this, this panel allows employees to check off guests which have had their information reported successfully.

When a new guest is checked in, their check-in is added to the list found in the reporting tab. Before entering the guest information into the external databases, be sure to mark whether the guest received either “services only” or laundry for that particular date’s check-in.

The reporting list updates 30 seconds after a guest has both the check boxes for HMIS and CW checked off, indicating that the guests data for that night’s stay has been manually recorded in the external databases by a staff member. Only guests with both database columns checked will be removed. If a checkbox is checked prematurely while doing data entry, the staff member will have 30 seconds to uncheck the box before the entry is permanently removed from the reporting list.

This feature allows staff to keep track of what check-ins are still awaiting entry into HMIS, CW, or both by automatically adding all check-ins, then removing any checked-off guest after a time to ensure that only relevant guests are displayed.

Contributor Guide

This section contains relevant information related to editing and contributing to the HRDC Database Manager's code to suit potentially changing and evolving needs.

Setting up the Development Environment

We recommend that you use the IntelliJ IDE, as it is the most straightforward to operate with our code. If that is not possible or desired, follow the steps below to set up VSCode to handle Java applications:

1. Install the latest release of the Java Development Kit
2. Locate the file path of the installed JDK
3. On the Windows taskbar, right-click the **Windows** icon and select **System**
4. In the **Settings** window, under **Related Settings**, click **Advanced system settings**
5. On the **Advanced** tab, click **Environment Variables**
6. Click **New** in the **System Variables** section to create a new environment variable
7. For **Variable Name**, type *JAVA_HOME*
8. For **Variable Value**, type the path to your JDK installation directory (e.g. C:\Program Files\Java\jdk-14.0.2)
9. Click **OK** to save the JAVA_HOME variable
10. In the same **Environment Variables** window, under the **System Variables** section, find the **Path** variable and select it
11. Click **Edit...** to modify it
12. In the Edit Environment Variable window, click **New** and add %JAVA_HOME%\bin
13. Click **OK** to save the changes

Verifying the JDK Environment Variable

1. Open Command Prompt by typing *cmd* in the Windows search bar and selecting **Command Prompt**
2. Type *echo %JAVA_HOME%* and press **Enter**. You should see the path to your Java installation directory
3. Type *java -version* and press **Enter**. This should display the installed Java version information

Accessing Source Code

The HRDC Database Manager's source code is found in several .java files within *Project_Files/warming_hut/src/main/java/sssp/View*. **MainMenuMockupAlt** contains the main controllers for the UI, along with most of the functionality of the Check-In window. **TableCellListener** controls the functionality of the table within said window; **ButtonColumn** controls the buttons next to each name in the table; **EditGuestPanel** controls the guest panel along the right side of the window; **NewIssuePopup** and **IssueDetailsPopup** control popups associated with said panel; **GuestDetailsPanel** controls the functionality of the Guest Details window.

Directory Structure

- Project_Files
    - warming_hut
        - src
            - main/java/sssp
                - **Helper: Database helper files**
                - **Model: Database Keys**
                - **View: UI handling**
                    - **components: Database-synced UI components**

Building the App Installer

Prereqs:

In order to build the installer for the app, you need wix3 installed on your local machine.

Process:

Running the 'install' maven lifecycle will build the installer. Once built, the installer can be found under "Project_Files\warming_hut\target\installer".

Configuring:

The installer's build process and configuration can be found in the pom.xml file in jtoolprovider-plugin's configuration.

Design Patterns and Program Structure

There are a few design patterns and helper classes written to ease modification and ensure robustness of the program.

- The “DBSynced*.java” classes in sssp/View/components provide a simple API for synchronizing UI elements with the database.
- The “*Singleton.java” classes in sssp/Helper ensure a single instance of certain key components is always available. Some of these Singleton classes also come with helper methods for easy access to certain functionality.

Backend Structure

This program is designed to connect to a firebase realtime database. This database needs to have a structure that matches the structure read in by the DBConnectorV2 class. It should have a clients section, with a section titled HRDC nested within it containing a section called Tables.

This Tables section is designed to hold multiple tables of varying depths as specified in the code. Depths are denoted as either 1 or 2, A depth 1 has a list of sub entries, and then each sub entry has values, a depth 2 has a list of lists of sub entries, and each of these sub entries has values.

The Depths and expected tables can be found in the DBConnectorV2 class. Adding new tables is entirely possible, but adjustments to the DBConnectorV2 class would need to be made in order to do so. When possible, we recommend making use of the Attributes table to add additional values.

On the matter of data formatting, we strongly advise all tables to have the same fields for all sub entries and lists contained inside them. A notable exception to this is the Attributes table, which is intended to be an asymmetrical place to add one off variables, or additional properties to tack onto other entries.

We also strongly advise against the use of varying depths, or passing records in as values of other records. The best practice is to take the other record’s ID and use that to identify a link. This link can then be expanded in the local code as needed.

On the subject of ID, we strongly advise future developers to make use of the UUIDGenerator class in order to ensure each entry has a fully unique ID to prevent duplicate IDs and potential crashes.

Interacting with the Database should be done through the asyncPush() and asyncPull()

methods. Using the database stored in the DBConnectorV2's database attribute to interact with the data, asyncPull to load it and asyncPush to commit changes.

artifactDatabase and freshDatabase are database copies that see occasional usage but largely serve as part of the push and pull methods. I do not recommend interacting with them unless absolutely necessary.

The push and pull methods work by individually pulling the database, or pulling it, checking if differences are local changes or changes pulled from the remote. Merging local changes with priority and then pushing to remote.

At present, this is not a perfectly efficient approach. We segmented the pull requests by table, and in theory it may be possible to automatically receive smaller updates. These options however we did not explore in depth. If questions of efficiency arise, this is the direction we would point in to mitigate potential problems.

Bug Reporting

To report a bug, create a new issue at our issue tracker; you may need to register a Github account to create an issue. Please include information about what you were doing when the bug occurred and how to reproduce it, if possible.

Reporting from the Database Manager

To report a bug directly from the app, on the check-in menu, click the **Report a Bug** button. This will open our Github page, which is the most efficient way to contact us about issues. Using a registered Github account, click the **New Issue** button to create a new issue entry. Please include information about what you were doing when the bug occurred and how to reproduce it, if possible.

Created by Alexander Maliziola, Beau Baer, Jacob Rivers, Josh Harvey, and Rory Schillo

Section 5: UML:

Figure 1.

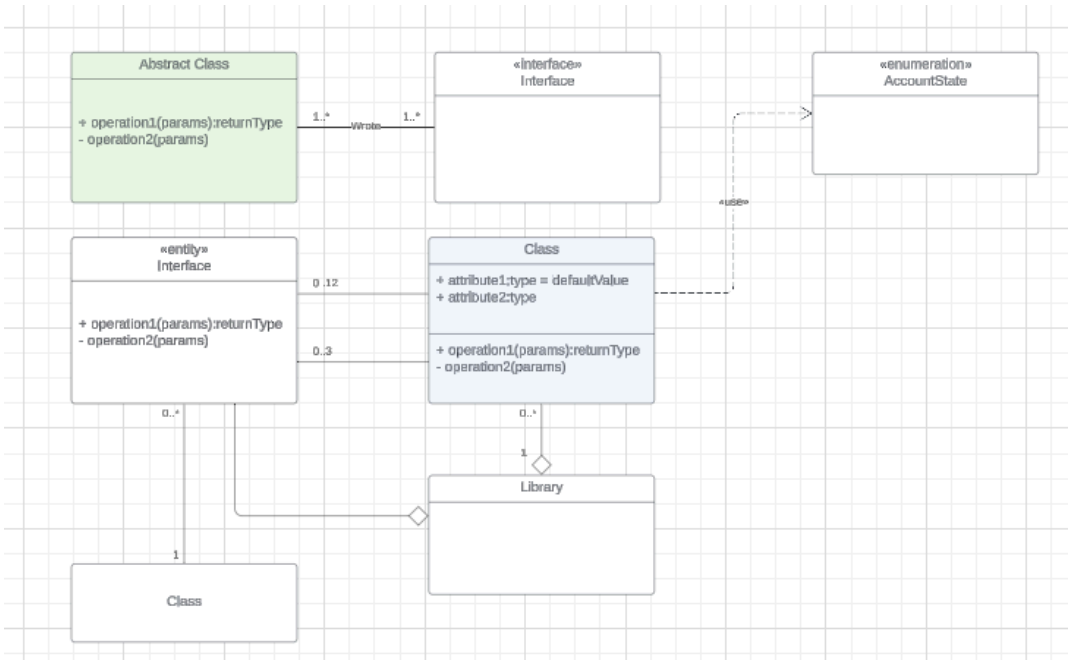


Figure 2.

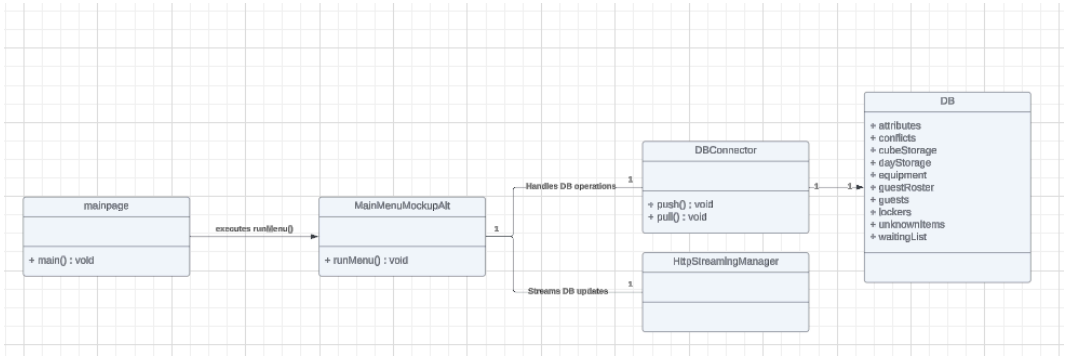


Figure 3.

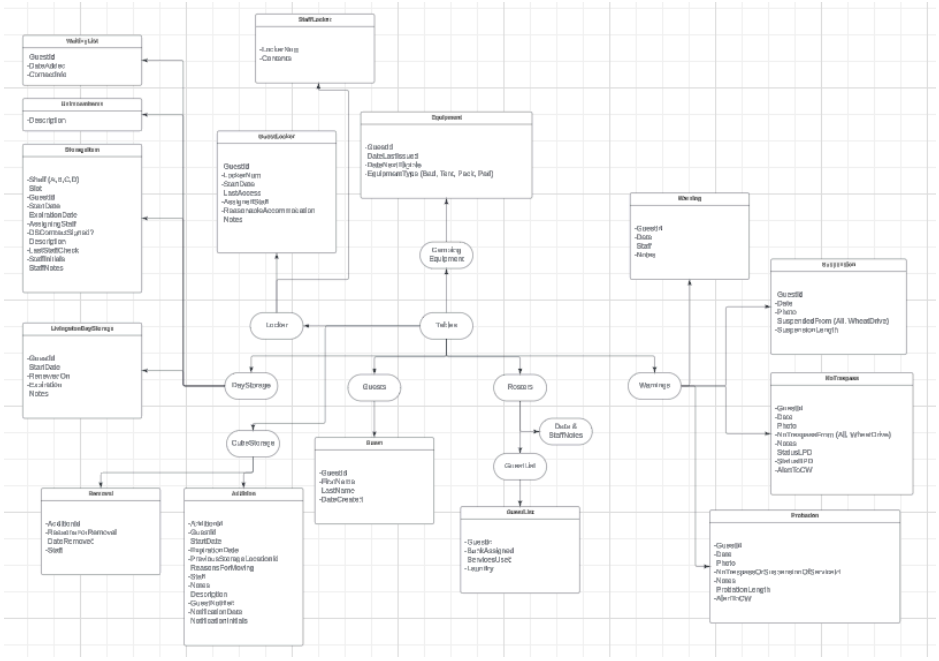


Figure 4. (Due to its size, in order to make it legible I had to split up this diagram by section. I attempted to preserve the overall appearance and connections)

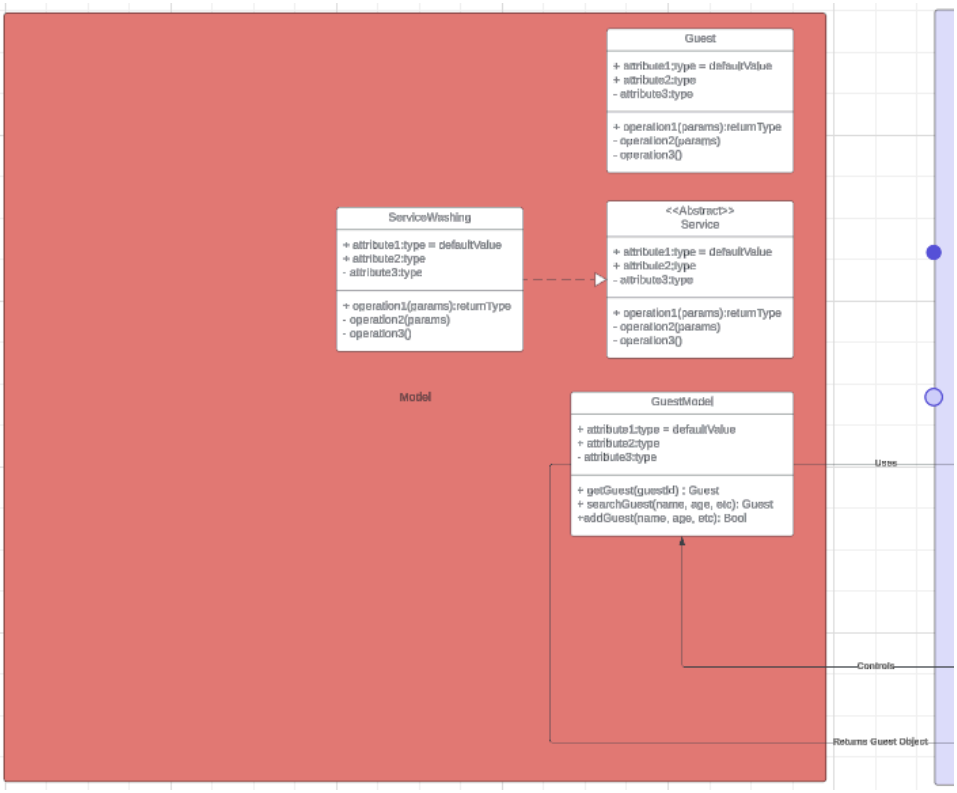


Figure 4 Cont.

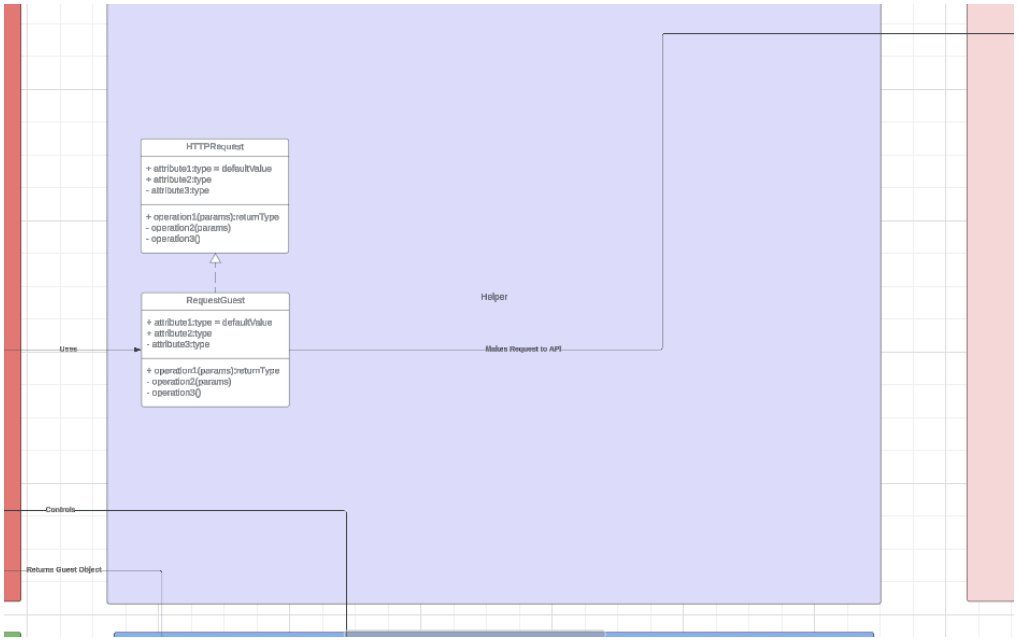


Figure 4 Cont.

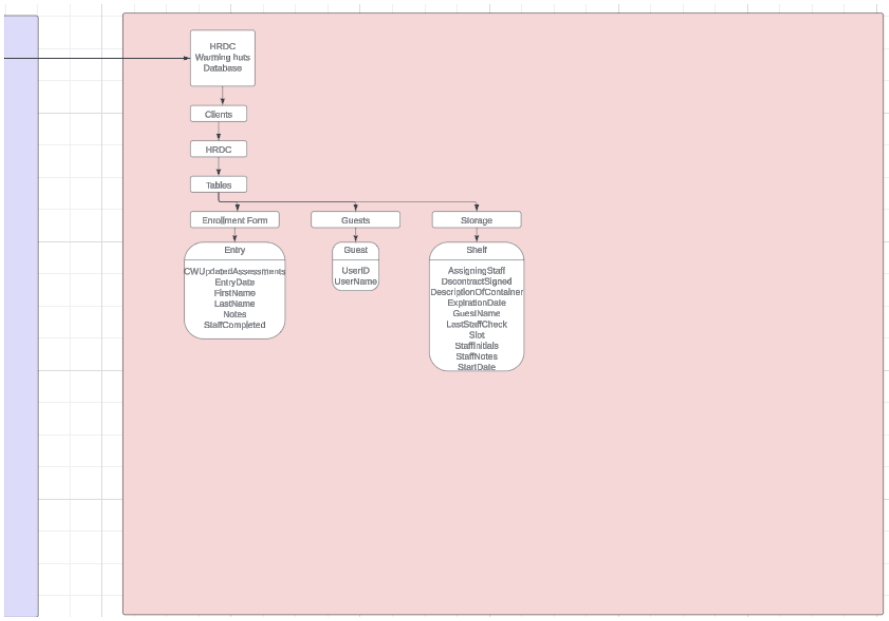


Figure 4 Cont.

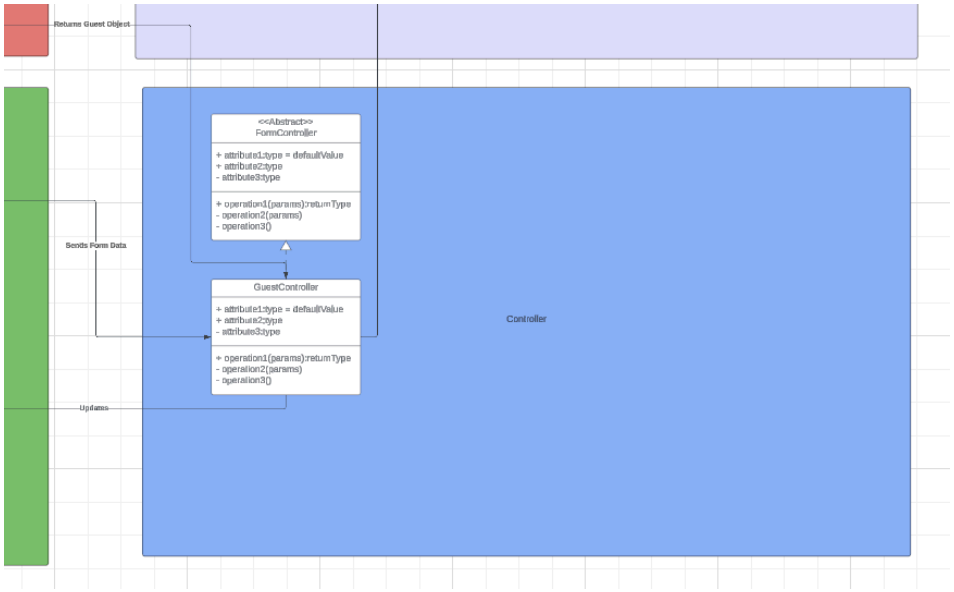


Figure 4 Cont.

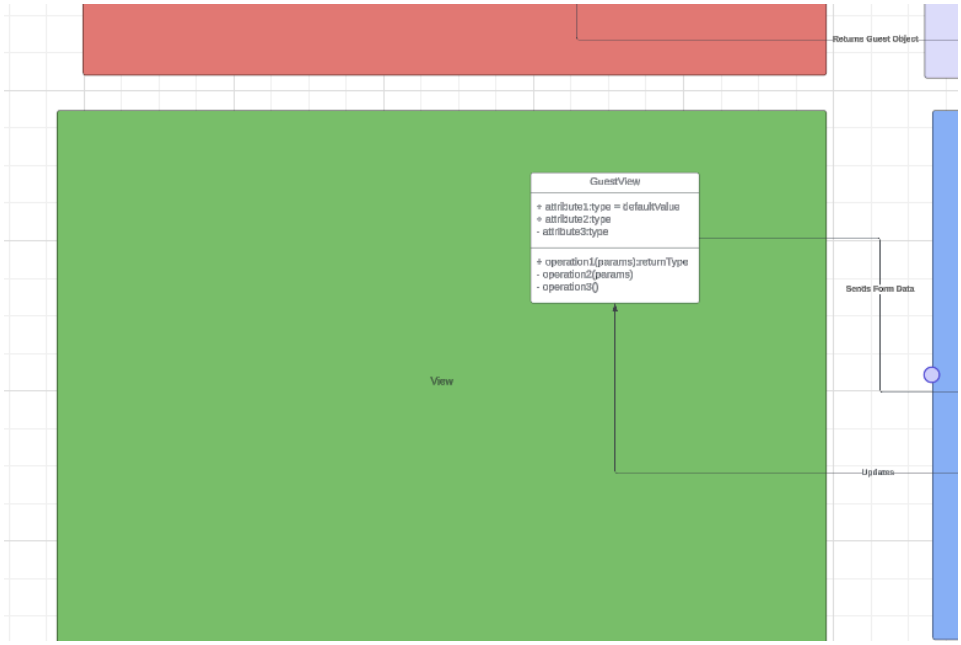
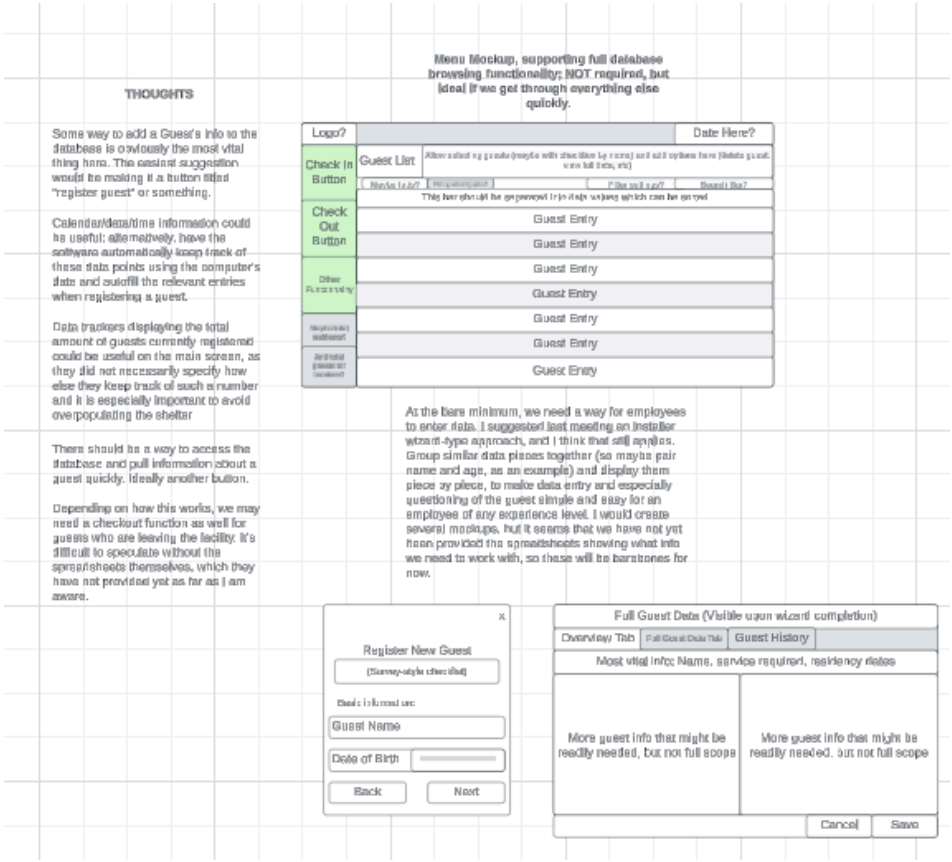


Figure 5.



Section 6: Design Trade-offs:

Java can be used effectively to create a desktop application, but it cannot be used to develop web applications. Using Java locked us entirely out of web based models and into a desktop application. This was a tradeoff as developing a desktop application's visuals was significantly harder than developing a web application using HTML and css.

The drawbacks were harder visual development, but the benefits we decided were more substantial. We believed a desktop application would be more secure and allow us to handle sensitive information more safely. Additionally a desktop application would not require a web hosting server, only firebase for the data storage, this would make handoff to the client after class easier.

A final major advantage to using Java is that comparatively to web based scripting languages, our group felt more familiar with performing complicated project development in Java.

Section 7: Software Development Life Cycle Model:

We completed this software using the Agile Development life cycle model for our project. Agile development has an emphasis on iterative development, breaking items down into smaller steps, creating a backlog and working in 2 week long sprints to meet specific targets.

We found this aspect of the agile development model to be helpful, as it let us organize things, and gave us concrete goals that were easy to lock in on and focus down. The major issue with it was that based on our timeframe, things did feel a bit rushed, and we were working with a burndown chart that we struggled to meet. Sprint based breakdowns did help with the quality of each product, and we got rudimentary releases out on time, but having to shift focus so rapidly did often break up larger goals along unfavorable schedules leading to us falling behind on some requirements due to them taking more than the expected sprints.

Flexibility is another key focus of agile development, and to us it was both a blessing and a bane. We had several initial ideas we had to drop to more closely align with the targets our client wanted to hit, and being able to pivot was highly advantageous. It was somewhat detrimental however as certain client requirements changed over the course of the class and as a result, the understanding that we could pivot necessitated doing so. While this did result in a better product for the client, it caused severe problems for our expected progress, and slowed us significantly.

Collaboration is a vital element in agile development, and on the whole I would say this was a strict benefit to our team. Being able to actively communicate with our client enabled us to drop elements that were less critical and focus on what they really wanted. Within our team, sharing updates and proposing solutions at and beyond the Standups we had at the startups at the beginning of class enabled us to rapidly deal with most of the implementation problems that arose throughout the project.

Team driven work is a major benefit to agile development, and it is one we certainly took advantage of. Being able to shift the scrum leader and decide on implementation in a small team enabled us to make this solution in the timeframe we did. Working with an externally provided schema, or answering closely to a higher authority would have drastically slowed development. Being able to Ax unnecessary elements, change plans and re-assign work on the fly enabled us to just get to work and produce results without any bureaucratic nonsense slowing us down waiting for approval and submitting requests.