

GCISL Full Stack Application

MVP Project Report Draft 2

**Washington State University Granger Cobb Institute of
Senior Living**



GCISL Team

Musa Husseini, Tom Arad, Nathan Bunge

3/2/2023

I. Introduction	4
I.1. Background and Related Work	4
I.2 Project Overview	4
I.3 Client and Stakeholder Identification and Preferences	6
II. Team Members - Bios and Project Roles	6
III. Project Requirements	7
III.1. Use Cases	7
III.2. Functional Requirements	8
III.2.1. Accounts	8
III.2.2. Posting	9
III.2.3. Messaging	9
III.2.4. Analytics	9
III.3 Non-Functional Requirements	10
III.4 System Evolution	10
IV. Solution Approach	11
IV.1 System Overview	11
IV.2 Architecture Design	11
IV.2.1. Overview	11
IV.2.2. Subsystem Decomposition	13
IV.2.2.1 [Widgets]	13
a) Description	13
b) Concepts and Algorithms Generated	13
c) Interface Description	13
IV.2.2.2 [View Models]	14
a) Description	14
b) Concepts and Algorithms Generated	14
c) Interface Description	14
IV.2.2.3 [Controller]	15
a) Description	15
b) Concepts and Algorithms Generated	15
c) Interface Description	15
IV.2.2.4 [Firebase]	16
a) Description	16
b) Concepts and Algorithms Generated	16
c) Interface Description	16
IV.3 Data design	18
IV.4 User Interface Design	19
V. Test Objectives and Schedule	19

V.1. Scope	20
V.2. Testing Strategy	20
V.3. Test Plans	21
V.3.1 Unit Testing	21
V.3.2. Integration Testing	22
V.3.3. System Testing	22
V.3.1. Functional testing:	22
V.3.2. Performance testing:	22
V.3.3. User Acceptance Testing:	22
V.4. Environment Requirements	23
VI. Project and Tools Used	23
VII. Description of Final Prototype	24
VIII. Project Delivery Status	26
IX. Conclusions and Future Work	27
X. Acknowledgements	28
XI. Glossary	28
XII. References	29
XIII. Appendices	30
Appendix A	30
Image 1	30
Image 2	31
Image 3	32
Image 4	32
Image 5	33

I. Introduction

This document will provide a description of the project provided by Granger Cobb Institute for Senior Living (GCISL). The document will keep track of all plans from the beginning to the end on the team's progress on the GCISL project. It will keep track of the project overview, background and related work, and client and stakeholder identification.

The GCISL team has been assigned to make a web application and a mobile application that allows alumni to share their career tracks with other alumni, faculty, and students. Each user will be able to post their job history and chat with other users. The project overview section will go into more detail.

The motivation behind this project was the Granger Cobb Institute for Senior Living is a new institution in the school of hospitality looking for a way to stay connected with alumni in the field. They have data of students who have graduated from the hospitality program in the past eleven years and plan to share this application with them so faculty and alumni can reconnect. The GCISL team will provide the faculty with an application that can help connect the school with alumni.

I.1. Background and Related Work

The Granger Cobb Institute for Senior Living was founded in 2019, with its main goal aiming to provide a better way for senior residents to socialize and be more active. Due to the industry's growing nature the institute is expecting to have a major increase in their program enrollment. As such GCISL is looking for an app which will allow them to track up to date information as well as communicate with their students and alumni. Since the institute is relatively new they do not have any pre existing software applications that could be improved upon or used as a case study for this project. As such most of our learning and resources will come from other applications which implement similar ideas to the ones we have to provide.

Some of our inspiration will derive from successful designs from applications like Facebook, WhatsApp, Discord, and LinkedIn which have key features that our app is aiming to include. They all have the ability to create accounts, post updates, and send messages. As a team, we have past experience working with Flutter and Firebase, so those frameworks would be potential candidates for this project. There is a lot of documentation, pre existing tutorials, and open source projects that we could use as references. These can easily be found on youtube and other online platforms such as github.

I.2 Project Overview

The goal of the project will be to provide a platform that will allow GCISL graduates and current GCISL faculty to stay in touch. This only works when the alumni and faculty are able to

use the application easily, which is why our sponsor has expressed the importance of ease of use for our application. We will aim to make the application as simple as possible, while still satisfying all the objectives outlined by our client.

Since our sponsor wants as many graduates to use the application as possible, she has also requested that we create both a web and mobile application. This will make it as accessible as possible, since users will have the option to use whatever device is easiest for them. This also opens possibilities for mobile notifications for updates and messaging.

We aim to develop the website first using Flutter[8]. This will allow us to build a prototype at high speeds to get feedback from the client. Then, once we have more of the specifics of the sites flushed out, we can very easily port the application to a mobile version using Flutter's multi-platform capabilities.

The first major component of the website that we will need to implement will be creating users and roles. As of now, there will be graduate and faculty roles. Both need to be able to easily create an account using a form, which they can then use whenever they want. Graduates specifically will need a way to easily update their profile, so that faculty can keep track of their career and life status.

Another major component will be posting to the site. Current faculty members will need to be able to post updates about the program, and everyone will need to be able to view them. Not only will this allow graduates to keep up to date with the program, but it will allow faculty members to easily coordinate major events and announcements.

The last feature that our sponsor has highlighted is the ability to message others in the application. Graduates could use the messaging feature to get advice from other graduates and facilities, and current faculty members could use the messaging to coordinate with each other and stay in touch with previous graduates.

Once the website has been created, we will have to find a good domain to host the site on. Our sponsor has suggested using a domain from WSU, which may lower overall costs. Our team will need to communicate with WSU site hosting services to discuss options for hosting the website.

After initial development of the prototype website, we will demo the prototype to our sponsor to get feedback for any changes or additional features that they want to be added. The mobile version of the application will then be addressed. Again, developing the site in Flutter will streamline this process because it allows for multi-platform applications using a single codebase.

The user base for the site will start out small, but would be expected to grow alongside the growth of the Granger program. This makes us lean toward using Firebase[7] as a backend, as it is free to develop and deploy for a small user base, and payments increase as your user base expands. This will need to be discussed further with the client, but it would allow us to develop, demo, and deploy the application for free, and our client would not have to spend any money until the user base is in the hundreds or thousands.

I.3 Client and Stakeholder Identification and Preferences

Our client is Washington State University Granger Cobb Institute of Senior Living and our client will be Darcie Bagott. Mrs. Bagott expects there to be a web and mobile application that can be used by faculty, students, and alumni. She needs there to be a way for users to communicate with each other and post job updates. She also prefers for this application to focus on it being easy to communicate with other users. It was mentioned a lot of their work is based on socializing and communication, so this needs to be a focus when designing the application.

Stakeholders in this project will be the users of the application. The department is relatively new so as of now there are only a few people working in the department, and only Mrs. Bagott is interested in being a part of the application. The needs of our stakeholders will be a functional application that they find user friendly in not only functionality but also interface design.

II. Team Members - Bios and Project Roles

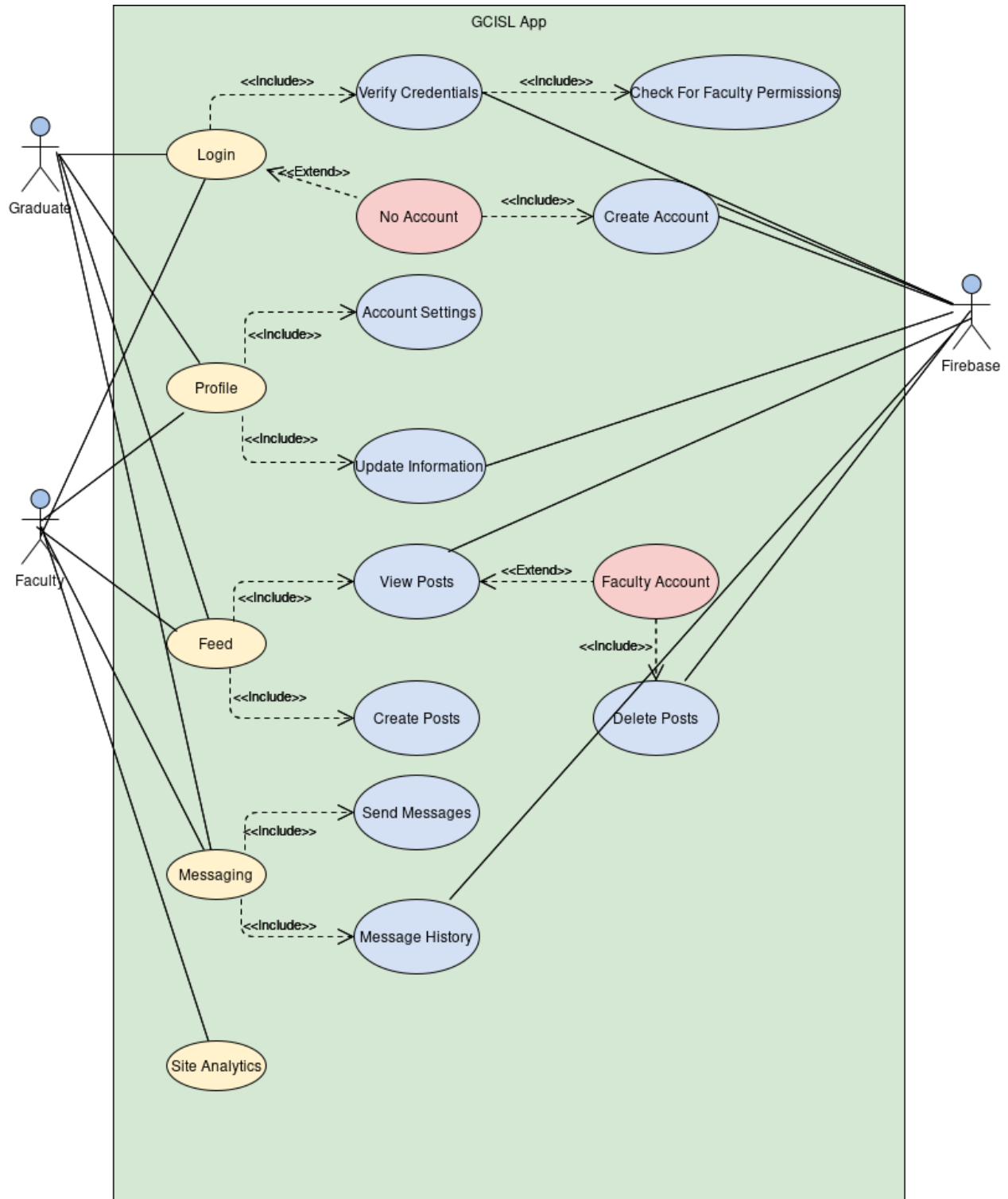
Musa Hussein is a Software Engineering major with interests in web and mobile application development. Musa has prior experience working as a software engineer intern at FotoFetch. His technical skills include proficiency with C/C++, C#, Python, and NodeJS. Musa's responsibilities for this project include team management, client communication, coordination, and doing full stack work for web applications.

Nathan is a Computer Science major with a minor in Math and Music. His prior software experience includes an internship at the Department of Defence developing submarine tracking software. He is very skilled in Python, C++, and Dart. Responsibilities for this project include issue tracking, git management, sprint video editing, and backend development.

Tom Arad is a Computer Science major with a minor in Math and interests in web development, machine learning, and data mining. Tom's prior experience includes an internship with the Snoqualmie Valley Innovation Venter where he helped design a website for SnoValley Jobs using wordpress. In addition Tom has a technical background with C, C++, Flutter, Python and HTML. Tom's responsibilities for this project include: frontend design, coordination with backend developers, and project demos for the client.

III. Project Requirements

III.1. Use Cases



If applicable, provide some major use-cases that illustrate scenarios for using your product. Use cases tell a story about how an end user interacts with the system under a specific set of circumstances. You may illustrate the use-cases with UML diagrams [5].

- As a graduate, you can fill out a form to create an account (Accounts)
- As a graduate, you can choose to receive updates from the program (Accounts)
- As a graduate, you can update your profile so that faculty can know your career status (Accounts)
- As a graduate, you can message other graduates so you can stay in touch with alumni (Messaging)
- As a graduate, you can message current faculty so you can stay in touch with the program (Messaging)
- As a current faculty member, you can fill out a form to register as faculty. (Accounts)
- As a current faculty member, you can send/post updates so that graduates can be up to date on the program (Posting)
- As a current faculty member, you can message past graduates so you can stay in touch with alumni and get their feedback (Messaging)
- As a current faculty member, you can message other faculty members so you can coordinate program events (Messaging)
- As a current faculty member, you can view past graduate profiles so you track their career status (Accounts)
- As a current faculty member, you can view a summary of the graduates' data so that you can make better decisions on the current program (Analytics)

III.2. Functional Requirements

III.2.1. Accounts

Account creation: The system must allow faculty and graduates to create accounts. They must also be able to update their account at any time.[4]

Source: Darcie Bagott - Applies to both current faculty and past graduates

Priority: Level 0: Essential and required functionality

Signing In: The system must allow faculty and graduates to create accounts and sign in. They must also be able to update their account at any time.

Source: Darcie Bagott - Applies to both current faculty and past graduates

Priority: Level 0: Essential and required functionality

[Notification Settings]: The system must allow faculty and graduates to choose their notification settings for their account

Source: Darcie Bagott - Applies to both current faculty and past graduates

Priority: Level 1: Desirable functionality

III.2.2. Posting

[Posting Updates]: The system must allow the faculty and graduates to post updates to the site.

Source: Darcie Bagott - Applies to both current faculty and past graduates

Priority: Level 0: Essential and required functionality

[Posting Announcements]: The system must allow the faculty to post announcements to the site that notify everyone.

Source: Darcie Bagott - Applies to both current faculty and past graduates

Priority: Level 0: Essential and required functionality

III.2.3. Messaging

[Messaging]: The system must allow the faculty and graduates to message each other through the app or website. [10]

Source: Darcie Bagott - Applies to both current faculty and past graduates

Priority: Level 0: Essential and required functionality

III.2.4. Analytics

[Analytics]: The system must allow the faculty to view a summary of data of past graduates. Data includes where they are located, what company they work for, and contact information. Secondary information (not required but preferred) would be their title and years of experience.

Source: Darcie Bagott - Applies to current faculty

Priority: Level 0: Essential and required functionality

III.3 Non-Functional Requirements

Ease of Use:

Creating an account should be as simple and easy as possible. This will help encourage as many people to use the app as possible.

Scalability:

At first, the system only needed to handle a small number of users. However, as the institute grows, the systems will need to handle a large number of traffic to accommodate for the increased number of graduates and faculty.

WSU Integration:

If possible, the system should be integrated into a WSU domain. This will save costs on custom domains, and may make it easier to access for users.

Maintainability:

This software will be maintained by other developers in the future. The software needs to be well maintained and easy for any new developers to understand and pick up

Compatibility:

Since the product is expected to be a web and mobile application, the software needs to be able to run smoothly on different browsers, operating systems, and devices

III.4 System Evolution

As time passes software evolution will be required for this project in order for it to stay stable. Since this project is going to be developed from scratch, there will be room for improvement in the future. One major part of our system that will be needing upgrading will be the server as traffic increases. As for now since it is a new application, we anticipate low traffic, hence we will only be needing a low priced package to host. If the application grows in size and traffic, so will the server.

Another part of our system that may need to change in the future could be the front end and back end. As technology quickly evolves our system will have to also. As for now we are using Flutter and Firebase to work with. It will help create a good baseline deliverable quickly, but if this application grows in size the system will need to be updated.

The client also mentioned who after we are done developing the application that they will hire someone to maintain the software. It will be important that we leave our application well documented and open ended so as we pass on this project to other developers they can continue adding and improving the existing code base.

IV. Solution Approach

IV.1 System Overview

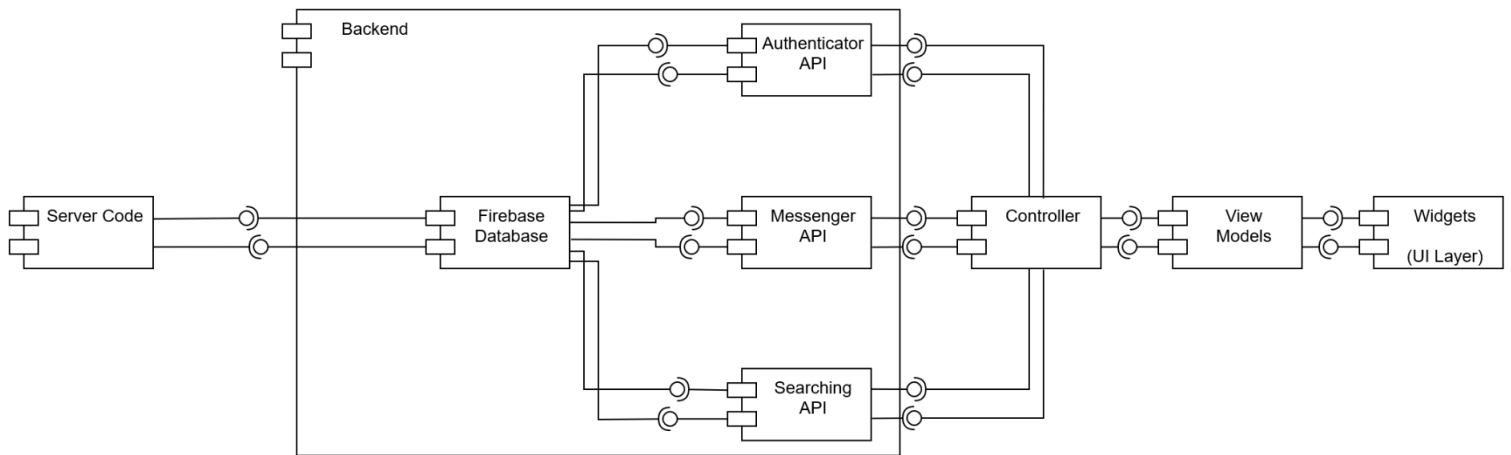
The GCISL program must have a robust and scalable architecture in order for the app to be functional and long lasting. It is a web based application, so we must define what is going on in the Front-End, the Back-End, and any APIs we will be using. We are mainly using Flutter for the UI and Front-End, and Firebase as the Back-End and APIs. The following sections will define what type of system architecture we used to organize these and why.[11]

IV.2 Architecture Design

IV.2.1. Overview

We decided to use a Client-Server architecture for the GCISL program. This is because it is a web based application, so users will need to be able to view updated data that is being sent from a server. As stated before, we opted to use Flutter and Firebase as our main frameworks for our codebase, which consequently led us to create the architecture as shown in the following diagram. On the far right is the UI layer, which contains all the widgets we will define. These are all the things that the user will see on their page and interact with. The widgets themselves are only tasked with displaying information in the most well looking fashion and initiating callbacks only when necessary. Next to that is the View model, which holds state info and updates any incoming data when it is changed. The view models can be updated when users puts in an input and sends those to the controller. In the reverse setting, it takes the updates from the controller and sends them to the widgets to be displayed to the user. The controller is in charge of communicating with the different APIs to request the necessary information that the view model needs. All the APIs are part of the firebase backend. Currently, we are using the Authenticator, Messenger, and Searching API's that firebase offers [4]. Each of

these aim to satisfy a different requirement, and are explained more in depth in the subsystem decomposition section. The three API's then access the Firebase Database as necessary to retrieve or add information. Finally, the Firebase Database is dictated by the defined server code. This will tell the database how to organize the data efficiently and effectively.



IV.2.2. Subsystem Decomposition

IV.2.2.1 [Widgets]

a) Description

The widgets are responsible for all the functions that the user will be interacting with directly. This is typically considered the front end of the project. [12]

b) Concepts and Algorithms Generated

The model will be creating prototype forms that can be called and used in the application. Typically each class will be a form and will have multiple variables where the user input is stored.

c) Interface Description

Services Provided:

1. **Service name:** Registration View
Service provided to: User
Description: This will provide a user interface when the user is making an account
2. **Service name:** Login View
Service provided to: User
Description: This will provide a user interface when the user is attempting to log in
3. **Service name:** Post View
Service provided to: User
Description: This view will provide a view of all the job postings from users on the platform.
4. **Service name:** Messaging View
Service provided to: User
Description: This view will provide a user interface when users are messaging between each other.

Services Required:

1. **Service Name:** View Model
Service Provided From: View subsystem

Needs updated information from the view model to display the most accurate data.

IV.2.2.2 [View Models]

a) Description

The view model is responsible for defining essential components of the application. It will provide the forms that users will fill out for logging in, registration, posting job updates, or sending messages.

b) Concepts and Algorithms Generated

This is where a lot of the html, css, and javascript will be kept. It will create the layout of the application and control what the user sees.

c) Interface Description

Services Provided:

1. **Service name:** Faculty model
Service provided to: Widge, Controller
Description: The faculty model will define a specific type of user. This type of user will have basic attributes such as name, email, profession, etc. But this type of role will allow faculty to have access to specific data analytics on other users.
2. **Service name:** Alumni model
Service provided to: Widge, Controller
Description: This model is similar to the faculty model. The key difference is this type of user will be considered a “standard” user and will not have access to data analytics
3. **Service name:** Posting model
Service provided to: Widge, Controller
Description: The posting model is responsible for creating a form for posting messages, such as job postings. It will include attributes such as title, description, start date, end date, etc...
4. **Service name:** Messaging model
Service provided to: Widge, Controller
Description: This model will be similar to the posting model. The key difference with the messaging model is it is only intended for two users to interact with. There will only be a few attributes associated with it also, such as, recipient, title, and body of message.

Services Required:

Service Name: Controller

Service Provided From: the controller subsystem

IV.2.2.3 [Controller]

a) Description

The controller is responsible for defining how a user will add or complete a task. It acts as a connection between the view and database. It takes user input and decides what to do with it, making calls to the backend as necessary.

b) Concepts and Algorithms Generated

The controller will be in charge of managing the routes and forms that a user may go through when using the application. It will handle GET, POST, and DELETE methods as well as security for certain landing pages. It will take in the data received from the user, process it, and send the results to the model.

c) Interface Description

Services Provided:

1. **Service name:** Register (GET, POST)
Service provided to: View Model, Backend - Authenticator
Description: It will request data such as name, job, job title, years of experience, and any other relevant information. Once the required information is received it will add the new user into the database.
2. **Service name:** Login (GET, POST)
Service provided to: View Model, Backend - Authenticator
Description: It will request data from the user asking for login credentials, if there is a match in the database they'll be redirected to a success page, if not they will be prompted to login again.
3. **Service name:** Logout (GET)
Service provided to: View Model, Backend - Authenticator
Description: If the user is logged in they will be logged out and sent back to a login screen.
4. **Service name:** Post Job (POST)
Service provided to: View Model, Backend - Database
Description: It will take the information a user inputs for a job posting, create a database entry for it, and post it to the server.
5. **Service name:** Send Message (POST)
Service provided to: Model, View, Database - Messaging

Description: This will allow users to send messages to other users.

Services Required:

Service Name: Messaging Data

Service Provided From: Message API

Service Name: Authenticating User Information

Service Provided From: Authenticator API

Service Name: Searching tool

Service Provided From: Searching API

IV.2.2.4 [Firebase]

a) Description

The database will be the place where all information is stored. Information can vary from user information, postings, messages, and analytics.[7]

b) Concepts and Algorithms Generated

The database will be created to store information in different tables. Each table can have relationships with each to share information. In order to call this information there will need to be queries called upon tables and each of its columns to retrieve specific information.

c) Interface Description

Services Provided:

1. **Service name:** User information table
Service provided to: Controller subsystem
Description: This table will hold all relevant user information such as names, login credentials, and career information
2. **Service name:** Post table
Service provided to: Controller subsystem
Description: This table will contain information on each post made. Columns of this table can include but are not limited to, title, body, and comments.
3. **Service name:** Messaging table
Service provided to: Controller subsystem
Description: This table will keep track of all the messages sent between users. It will allow there to be message history.

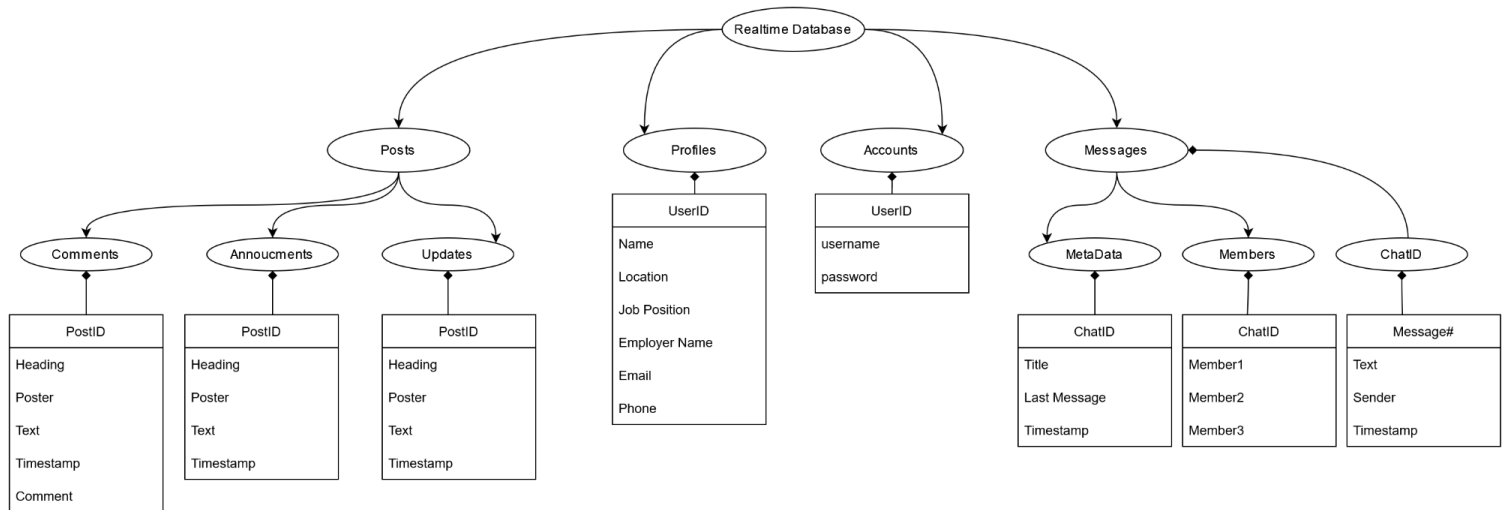
4. **Service name:** Analytic Table
Service provided to: Controller, Faculty Users
Description: This table will store specific data that can only be seen by faculty users.
5. **Service name:** Message API
Service provided to: Controller and Database
Description: This API will handle messaging between users and notifications.
6. **Service name:** Authenticator API
Service provided to: Controller and Database
Description: This API will handle user credentials when logging into their accounts.
7. **Service name:** Searching API
Service provided to: Controller and Database
Description: This API will handle search queries when getting data from the database.

Services Required:

1. **Service Name:** Controller
Service Provided From: Controller subsystem
2. **Service Name:** Model
Service Provided From: Model subsystem

IV.3 Data design

We will be utilizing Firebase as our main source of API extensions and database, which means we will store our data in Firebase's default data structure format. Firebase stores data in realtime using a JSON tree [6]. As such, each of our data objects will be stored as a node in the JSON tree.



Posts: Posts will either be stored under announcements or updates. This makes searching by one or the other categories much faster than having them altogether. For example, If they were classified in one branch, then every single post would be retrieved if you requested to see only announcements. However, if they are separated, then only the announcement branch would be searched. For the same reason, comments are separated into a different branch. Comments are only displayed when you select to view them, so we don't need to retrieve them from the database every time we are viewing the main feed page.

Profiles: Profiles contain all data for analytics that are necessary to track. So far this is only Name, Work Location, JobPosition, Employer Name, Contact Email and Phone. Depending on which categories are searched more frequently than others, this may be reorganized in the future to optimize queries.

Accounts: Accounts only contain the information necessary for users to login. This is to optimize sign up/sign in times for the application.

Messaging: Contains all the data necessary to allow users to message each other and view old messages. MetaData contains the minimum data needed to display all the chat preview windows. This is separated so that the preview windows can all be loaded without having to load in all of the messages in the whole database. Members contain the members that are a part of each chat session, and ChatID contains the actual text for every message in a session.

IV.4 User Interface Design

The GCISL team has created a partial UI design for the GCISL web application. [Appendix A](#) contains images of this design. Upon launching the application will display [Image 1](#), the login/sign up page. This page will allow users to create and sign into their account (Satisfies Login use case). Following that the application will display the “Home” page as can be seen in [Image 2](#) it contains a feed of posts made by the other users and allows creating new posts (Feed use case). [Image 3](#) displays the “Personal Information” page” this page will allow the users to provide and edit their current information in accordance with the data the client is interested in (Profile use case). Once the client enters their information it will only be updated if they click the “save” button at the bottom of the page. This is a barebone design of what the page will look based on the client’s interests and as such it is likely to change in the future to be more visually appealing. [Image 4](#) displays the “Messages” page, this page allows users to view their message history with other users as well as send new messages (Messaging use case). This page also has a lookup feature for users to find specific people they want to contact. The design for this page was inspired by a community design on figma.^[2] The last image ([Image 5](#)) displays the “Analytics” page (Site Analytics use case). This page will be visible only to those with staff permissions and will contain all the personal information provided by the users. Currently the information is displayed in a table, but further discussion with the client about data visualization could lead to changes. In addition every single page will have a side menu as can be seen in all the images. Through this menu they will be able to go between the different pages of this application described above.

Some features that appear in the use case diagram do not appear in this early design. The home page should allow users with staff permissions to delete posts. This feature will be added later once the UI team gets more time to expand on the design. Our use case diagram also shows that users should be able to alter their account settings. Since these settings are not yet clear to us we have not been able to create a design which implements this option, hence this will be also added to the UI at a later point.

V. Test Objectives and Schedule

Software testing will be crucial in the development of the Cobb Connect application. It will help the team identify any issues present in the app and as such outline what needs to be fixed so that the application meets the expected standards and quality. In order to achieve this goal the Cobb Connect team will use existing testing tools provided by flutter and firebase as well as tools made by independent developers which were made available to the public such as the firebase fakes.

When handling the front end flutter code the team will use the flutter_test tools to conduct unit testing as well as widget testing. The flutter_test package provided by the flutter SDK allows the user to add test files written in dart which check for logical issues as well as

render widgets to the screen. While the flutter_test package is handy when testing individual components of the app, it is also important to test how all of these components interact together. For the front end integration testing the team will be using the integration_test package which is also provided by the flutter SDK. This package allows the user to search for components as well as verify that everything that should be present in the app is indeed there. Similarly to the flutter_test package the testing files for integration testing are written in dart.

Testing the back end of the application is a bit more complicated. It is difficult to use the Firebase libraries to run unit tests since they require an actual device or emulator. A good replacement the team has found for this problem is the Firebase fake/mock packages. These packages implement the API of specific firebase libraries and simulate their behavior. Finally for back end integration testing the team will use the Firebase emulator which uses the cloud firestore as well as other core firebase resources.

The deliverables provided by the team will include our gitlab repository containing our application code, test files, and a README file which will provide all of the necessary information regarding the repository. In addition the team will provide documentation detailing the tests and application in a separate document. We will be using the agile process as our testing strategy. Tests will be written by the developers and implemented at the same time as the code for the application. Milestones: code, unit tests for the code, integration tests for the code, widget tests for the code, integration between flutter and firebase, demo, and user testing the code. These milestones may repeat if there is a need for updates or other additions to the application which require the beginning of a new cycle.

V.1. Scope

This document discusses our plans for testing the Cobb Connect Application. The scope of this document covers the general testing guidelines which the team will follow, as well as the resources and frameworks we will be using to test our code. While this document provides a breakdown of the testing strategy it does not discuss individual test cases as well as the people who will be responsible for writing them.

V.2. Testing Strategy

For our testing strategy, our team will create tests for all aspects of the system. Everything from the front-end to the back-end will have some level of testing applied to it. This will include testing all landing pages, views, and functionality. To achieve these goals, these are the steps to be taken for our testing life cycle.

Our team will use the CI/CD (Continuous Integration/Continuous Delivery) testing strategy. This process will allow us developers to create and deliver product features in a timely manner. It will introduce automation into the stages of our app development.

1. **Write Test Cases:** All developers will be responsible for writing tests for the code they create. This will be required for all subsystems. Ensuring each developer creates tests for their own work will help keep our system under control.
2. **Run Test Cases:** Each developer will run the tests they have created. It is up to the developer to determine if their test cases have enough coverage and are concise.
3. **Make changes according to test results:** Depending on the result of the test case, the developer will need to take different actions. If the test fails, they need to locate the source of the bug, revise and fix the bug, and rerun tests until they pass. If all tests pass the developer may move onto the next step.
4. **Push code to remote, and CI runs commit through pipelines:** The CI/CD testing will happen once a developer pushes their code to the repository. The pipeline will run the newly committed code and check to make sure there are no conflicting issues. If the pipeline fails then the developer will need to redo the previous steps. If they pass the developer can move onto the next step.
5. **Make merge request to main:** In this step the developer will make a merge request to main to have their code a part of the main codebase. At this stage it is relatively safe to make this request due to the amount of testing done beforehand.
6. **Request must be approved by another developer:** As a rule of thumb all developers should have their code reviewed by at least one other developer on the team before merging to main. This will ensure that the code being proposed to be merged is being seen from a different perspective and not through automation.
7. **When the branch is merged into main, it will be deployed by the CD:** In this step after the new code is merged into main, it will be deployed by the CI/CD.

If a bug does get through the process described above there are certain steps to be done to fix this bug. The first step is to create an issue in the repository describing the bug. The issue should be assigned to the person who wrote the code, not the person who found the code. This is done so that the person who is the most familiar with the code can trace down the source of the bug. The developer responsible for the bug must not only fix the bug but also add test cases to catch any occurrences of the bug in the future.

V.3. Test Plans

V.3.1 Unit Testing

When creating unit tests the goal will be to isolate each function and give it inputs to test against the returned variable. Each unit test should have at least two to three tests. Each unit must test for a lower bound and upper bound (if applicable), an error check, and the expected result. The breakdown goes as so. Each class will be given a test suite. Each function within that class will be tested in their own individual unit. Within each unit the rules as described above will be used for assertions. Along with each test suite, there will be a setup function, which will set up all the required variables needed to test, and a tear down function to deallocate any memory used. [2]

V.3.2. Integration Testing

Using integration tests for our system will be difficult to do effectively. Our team will develop call graphs to help organize the way our system works within itself. We will create top-down call graphs for classes and create cross class graphs where needed. To integrate a class, we will follow the call graph created. We will begin with creating mocks and stubbing each function within the class. We then will go through one function at a time, un-stubbing it, and then verifying that the assert passes. The integration test for a class will be considered complete once all functions are un-stubbed and all assertions are passed. [1]

The Table for Unit Testing can be found in the **Appendix X.2.1.**

V.3.3. System Testing

The following types of tests will be performed to test the system:

V.3.1. Functional testing:

Our functional testing will cover each of our four main requirements: Accounts, Posting, Messaging, and Analytics. Each of these were defined in the Requirements and Specification section, which goes into more detail for what needs to be done in each requirement. Testing for accounts will involve both testing account creation and signing in. Multiple fake and real email accounts will be sent for verification, and the site must correctly verify the emails and passwords. Testing for posting will involve adding, deleting, and editing posts. The tests will need to verify that the main page is updated each time. Message testing will be very trivial, it must verify that a message sent to a user is received. Lastly, Analytics will need to be tested with fake user data to determine if the site is calculating correctly. [3]

The Table for Functional Testing can be found in the **Appendix X.2.2.**

V.3.2. Performance testing:

Performance testing will not be as straightforward as function testing, as the non-functional requirements are not quantitative in nature. This means that most of the non-functional requirements will be more up to developer discretion. After a prototype site is created, each of the non-functional requirements will be examined and the developers will have to determine if they have met the requirements. Any requirements that are not met will require code refactorization.

V.3.3. User Acceptance Testing:

User acceptance testing will occur multiple times throughout the app development stages. The team will provide a series of activities which will demonstrate the app functionality based on the requirements and specifications as well as what the team managed to get done.

These activities will most focus heavily on recent additions to the application. After the activities are performed the customer will be free to explore the application in whatever way they see fit and prove additional comments. Any major bugs or problems found will be fixed by the team and included in the activities of the next demonstration.

V.4. Environment Requirements

Unit testing will be performed using dart's testing package [11.] This has no hardware requirements, but all the installation steps outlined in the project readme will need to be performed in order to perform the unit tests. Integration tests will also be performed using dart's testing package. Functional tests will be done with both dart tests scripts, as well as outside black box testing. This means that the software requirements are the same as before; all the installation steps outlined in the readme are required to run the tests.

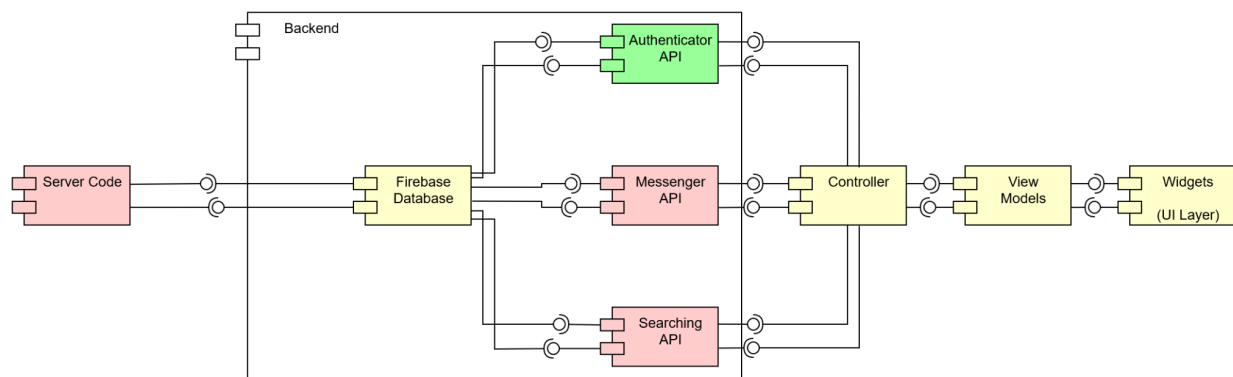
Performance testing will be done all by developer discretion, as there is no quantitative way to test the requirements. Therefore there are no hardware or software requirements, other than just accessing the website itself. This is the same for user acceptance testing. They will only need a device that has a web browser to access the website.

VI. Project and Tools Used

Tools/Libraries/Frameworks	Description
FireBase Packages	These firebase packages allow us to call and store information into our database

Languages Used in Project	
Dart	C++
C	HTML

VII. Description of Final Prototype



The above diagram shows the architecture for Cobb Connect. The green components show the parts that are completed, the yellow show ones that are currently in the process of being developed. Finally, the red components represent parts that still need to be implemented. Each of the components will be expanded upon in detail in the following sections. All the frontend for the pages fall under the View and Widget components, meaning that they are still under development. The backend has been created and connected, but not all the pages have been created yet, meaning not all the backend has been finished yet.

VII.1 Widgets

VII.1.1. Functions and Interfaces Implemented

The Cobb Connect team has implemented UI functionality for some of the more important aspects of the app so far. These widgets include: The sign in page, the register page, the appbar, the home page, and the analytics page.

The sign in page as well as the registering page design is relatively final, since we do not expect there to be a need to improve or change the way it looks. The appbar is also complete for now. We have added buttons as well as text and icons which correlate to the page which the buttons direct the user to.

The home page currently only has a very basic layout which contains a text form and a scrollable list of all the text entered into the form. Once the team connects the textform to the backhand the front end will be expanded upon. Similarly, the analytics page is currently at a very early point. It currently only displays a basic map using the Google maps app. More progress on this page as well as the other pages which lack widgets is expected in the next sprint.

VII.1.2. Preliminary Tests

So far the team has only demonstrated the app to our client by having one of the developers walk through the current design and features. The client has approved of our basic visual themes. As the app solidifies and gets closer to completion we will conduct more testing in which the client will be using the app. At the start they will be more hands-on, but as we go we will slowly transition into free exploration and testing by the client. As such the UI will undergo changes when a request from the client is made or when new content is added.

VII.2 View Models

VII.2.1. Functions and Interfaces Implemented

Currently the only main view model components that are relevant are the sign in page, register page, and the appbar. The sign in page asks the user for their login information and verifies it by looking for a corresponding entry in the user database, it also allows new users who do not have an account to navigate to the register page where they can create one. The register page asks for the user information and then inserts it into the database so that the account can be verified when they attempt to log in. Once inside the app the app bar allows the user to navigate between the different pages of the application by clicking the corresponding buttons. As the app progresses there will be more relevant components, such as the form which collects the user information.

VII.2.2. Preliminary Tests

When testing these components we conducted full app testing as well widget tests. For the sign in page we created some temporary accounts and verified that they could be authenticated, while other accounts with information that was not present in the system were unable to.

To test the registration page we created new accounts by providing the information into the text forms in the page and checked to see whether they would then be authenticated when attempting to log in with the same information later.

Finally to test the app bar we checked that each button fulfilled its functionality, that is navigate to the correct page.

Most of these tests were through interactions with the UI to make sure everything worked as expected.

VII.3 Controller

VII.3.1. Functions and Interfaces Implemented

So far controllers have been added for the user sign in page, sign up page and posting. They track the user input for the forms on each page so that they can send the text information to the authenticator. Future controllers still need to be added for the messaging page, profile

page, settings page, and any other pages that are made that need to communicate with the backend.

VII.3.2. Preliminary Tests

So far we have registered multiple accounts with various real and fake emails to test if the controller sends the information correctly to the authenticator. This has been successful, as you are now able to create accounts and sign in on the website. Further testing will be implemented as each new controller is added. Also, solidified unit tests will be created to avoid manually running tests ourselves.

VII.4 Firebase Backend

VII.4.1. Functions and Interfaces Implemented

We have completed the first big step in setting up the firebase backend, which was creating and connecting firebase to the flutter application. This means when we run our application, it connects to a firebase project that is hosted by google. We can then run any of the extensions we want by just adding them to our firebase project and including the necessary libraries in the flutter code. This has been done with the authenticator, as you can now create accounts and sign in, and the accounts are all saved in the firebase project. Future work for the firebase project will consist of adding the messaging api and searching api so that those pages can communicate with the corresponding backend components.

VII.4.2. Preliminary Tests

The tests so far have been creating new accounts and signing in with those accounts. The sign in authenticator has been tested with real and fake emails, as well as correct and incorrect passwords. It has passed all the tests. The sign up component has also been tested by creating new accounts with good and bad passwords, and then testing to see if those accounts were saved in the firebase project. It has passed all the tests. Future tests will be made for the messaging and searching API's.

VII.5 Server Code

VII.5.1. Functions and Interfaces Implemented

No specific server code has been created yet, as none has been required for the authenticator. This will be needed however for hosting, and likely for setting up database rules. This will all be addressed in future work, once the database is set up for profiles, and once the site has started its hosting.

VII.5.2. Preliminary Tests

Since there has been no server code yet, there have been no preliminary tests.

VIII. Project Delivery Status

The project will be delivered Tuesday, May 2, 2023. While the code and other deliverables will be submitted by April 28th, the few days in between will be spent finalizing our documents and any other final tasks. We plan to demonstrate our final project over a Zoom call to our client, as we have been doing throughout these past few months. We will be sure to leave the project in a good state where another team can pick up where we left off and continue to grow the project.

Included in our deliverables will be the following:

- Code repository
- Github Issues
- README
- Mockup Diagrams
- User Manual

Before submitting our final iteration of the project, all documents, source code, diagrams, and issues can be found in our Github repository. In order to properly run our project it is important to closely follow the instructions provided in the README file. Being able to follow these directions will allow any new user to set up the environment on their computer.

IX. Conclusions and Future Work

IX.1. Limitations and Recommendations

The application is in a usable state, but there are a few limitations in features. As of right now there is no messaging implemented yet. This feature will be implemented in a future sprint, but as of right now there are more urgent and important features to implement.

The analytics and mapping is in progress and is expected to be finished next sprint. There are a few issues with the database sending the needed information to the system, so once that is resolved location will be available on the map. As of right now there is a generic google map. To solve this we are currently investigating what may be causing an issue in our database and trying to track down the root of the problem.

The front end is always evolving and changing. As of right now it is in a vulnerable spot where we keep making UI changes. So there may be some inconsistencies in some of the pages on the website. We plan to have a solid front end UI completed in the upcoming sprints, and having this finalized will create a cleaner looking project.

IX.2..1 Future Work

For upcoming sprints there will be some major changes coming to the project. Since most of the prior sprints were spent planning and designing our application, the focus for future sprints will be implementation. We currently have a baseline application that has the screens/pages we will need and a login/registration system. The upcoming sprints will include adding messaging, posting, account edits, and data analytics. The messaging will allow for users to privately communicate with other users in the system. Posting will allow users to make public posts for anyone to view on the feed. These posts will typically be job or life updates that are meant to be shared as public information. Account edits will allow users to make personal edits to their accounts. As of right now you cannot view your account to make changes to it. There will be a user screen where users can add personal information, profile pictures, and change anything they want about their account. Data analytics will be something for users to view where other users are in the world. Admin users will be able to view more descriptive

details, the details will be later provided by our client. By May we expect to have a fully functional application that meets the clients needs and can be used by anyone.

X. Acknowledgements

We thank Granger Cobb Institute for Senior Living for providing a great project. Special thanks to our client Darcie Bagott for meeting with us and discussing new features and implementations for this project.

XI. Glossary

Flutter: An open source user interface development kit

Firebase: A hosting service for applications

Figma: A web application used for interface design

GCISL: Granger Cobb Institute of Senior Living

Front End: The part of the application the user will be seeing and interacting with

Back End: The part of the application that runs the logic and processing

Database: The part of the application that stores all data

Full Stack Application: An application that has front end, back end, and a database working together

UML Use Case Diagram - A visual representation of a system along with its main actors, roles, actions, artifacts, or classes.

Main Actors - A human person or another external system which interacts with the system being modeled

Operating System - A software that provides basic functionality for a computer.

Web Domain - The name of the website.

API: Stands for application programming interface. It is a type of software interface that offers a service to other pieces of software.

SDK: stands for software development kit, is it a is a set of software-building tools for a specific platform

XII. References

- [1] T. Hamilton, "Integration testing: What is, types with example," *Guru99*, 27-Aug-2022. [Online]. Available: <https://www.guru99.com/integration-testing.html>. [Accessed: 28-Oct-2022].
- [2] T. Hamilton, "Unit testing tutorial – what is, Types & Test example," *Guru99*, 27-Aug-2022. [Online]. Available: <https://www.guru99.com/unit-testing-guide.html>. [Accessed: 28-Oct-2022].
- [3] "Test: Dart package," *Dart packages*, 26-Oct-2022. [Online]. Available: <https://pub.dev/packages/test>. [Accessed: 28-Oct-2022].
- [4] "Functional vs Non Functional Requirements." GeeksforGeeks, April 29, 2020. <https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/>.
- [5] "Microsoft." Microsoft Support. Accessed September 28, 2022. <https://support.microsoft.com/en-us/office/create-a-uml-class-diagram-de6be927-8a7b-4a79-ae63-90da8f1a8a6b>.
- [6] Offutt, Jeff. Overview of software maintenance and Evolution. Accessed September 28, 2022. <https://cs.gmu.edu/~offutt/classes/437/maintessays/maintEvolutionOverview.html>.
- [7] Google. (n.d.). *Firebase documentation*. Google. Retrieved September 20, 2022, from <https://firebase.google.com/docs>
- [8] Flutter documentation. Flutter. (n.d.). Retrieved September 20, 2022, from <https://docs.flutter.dev/>
- [9] (Dutoit, 2010), 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice Hall, 2010.
- [10] "Apple Messages Template". Figma Community. <https://www.figma.com/community/file/1118503083462368752> (October 4, 2022)
- [11] "Structure Your Database" Firebase Documentaion <https://firebase.google.com/docs/database/web/structure-data> (September 15, 2022)
- [12] Firebase Extensions, Google (October 1, 2022) <https://firebase.google.com/products/extensions>

XIII.I Appendices

Appendix A

Image 1

Sign In

Sign in to stay connected.





Email

Password

☐ Remember me? [Forgot Password](#)

[Sign in](#)

or sign in with other accounts?

Don't have an account? [Click here to sign up.](#)

Image 2

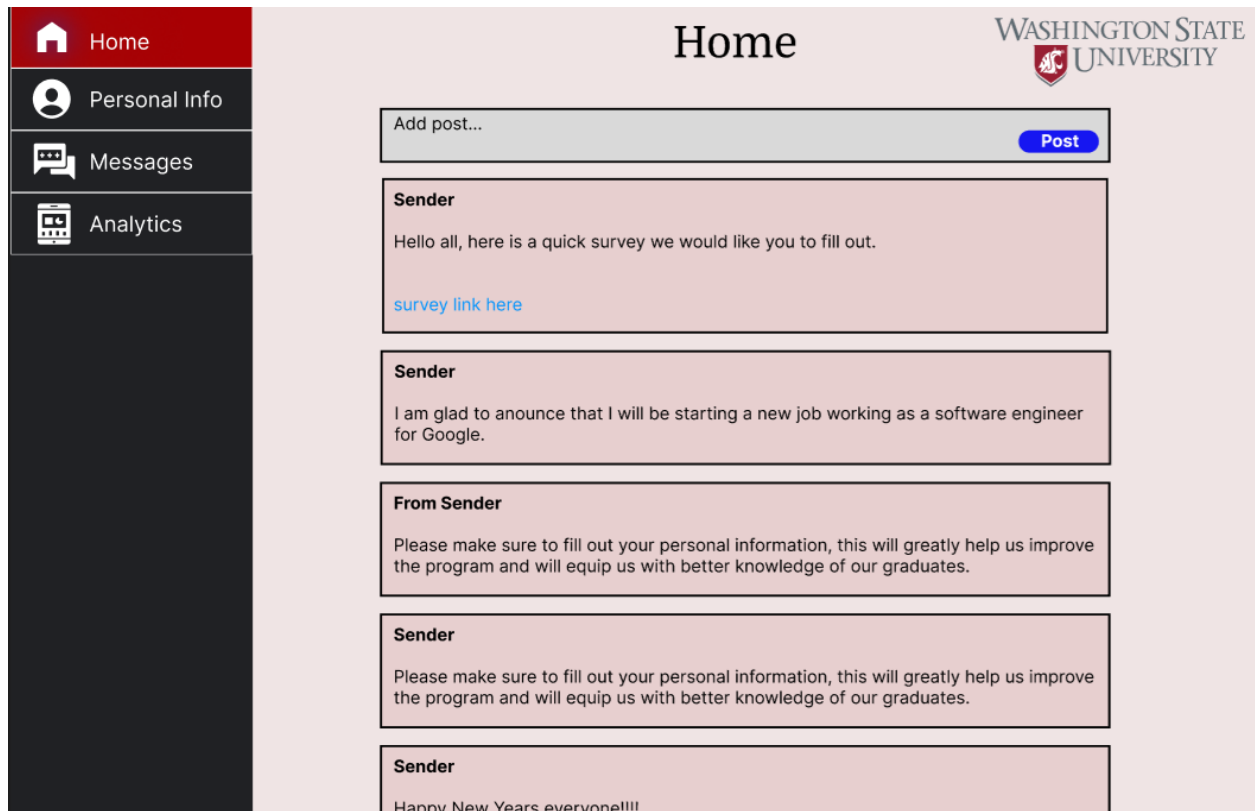


Image 3

Home
 Personal Info
 Messages
 Analytics

Profile Information

Name (first and last)

Employer's Name

Location (City, State)

Job Title

Employment Date

Email

Phone Number

Save

Image 4

Home
 Personal Info
 Messages
 Analytics

Let's get lunch. How about pizza?

Let's do it! I'm in a meeting until noon.

That's perfect. There's a new place on Main St I've been wanting to check out. I hear their hawaiian pizza is off the hook!

I don't know why people are so anti pineapple pizza. I kind of like it.

Messages

- Stephen Yustiono** 9:36 AM >
 Nice. I don't know why people get all worked up about hawaiian pizza. I ...
- Erin Steed** 9:28 AM >
 (Sad fact: you cannot search for a gif of the word "gif", just gives you gifs.)
- Daisy Tinsley** 9:20 AM >
 Maybe email isn't the best form of communication.
- Zach Friedman** 9:00 AM >
 Tabs make way more sense than spaces. Convince me I'm wrong. LOL.
- Kyle & Aaron** 8:58 AM >
 That's what I'm talking about!
- Dee McRobie** 8:35 AM >
 There's no way you'll be able to jump your motorcycle over that bus.
- Gary Butcher** 8:32 AM >
 Nathan is a hater, you can totes make that jump. Do it. Do it.
- Kyle Norman Olson** 8:02 AM >
 #letsgetcoffee
- Dee & Daisy** 7:45 AM >
 Good morning! I've got some great news, we got the grant!!!

Image 5

[illegible]

XIII.2 Testing

XIII.2.1 Unit Testing

Class	Test Type	Input	Output (Expected)	Output (Actual)	Test Status Result	Description
Account.dart Register	Normal	newEmail@gmail.com 123456	1	1	Pass	Successful register. (Requires account to not already exist)
	Boundary	123456	-2	-2	Pass	Invalid email
	Boundary	newEmail@gmail.com	-4	-4	Pass	Password too weak
	Exception	test@gmail.com 123456	-1	-1	Pass	Email already in use (Requires account to exist)
	Exception	nouser@gmail.com 123456	-2	-2	Pass	Invalid email
	Exception	newemail@gmail.com 123	-4	-4	Pass	Password too weak
Account.dart Signin	Normal	test@gmail.com 123456	1	1	Pass	Requires account to exists
	Boundary	“” 123456	4	4	Pass	No user with “ ”
	Exception	test@gmail.com 12345	5	5	Pass	WWrong Password
	Exception	nouser@gmail.com 123456	4	4	Pass	No user with that email. Requires that users not existing
	Exception	noaccount@gmail.com 123456	2	2	Pass	no account with that email

XIII.2.2 Functional Testing

Test	Input	Output (Expected)	Output (Actual)	Test Status Result	Requirements
Account Creation	test@gmail.com 123456 123456	Error: Email already exists	Error: Email already exists	Pass	test@gmail.com is already in database
	newtest@gmail.com 123456 123456	Account created	Account created	Pass	newtest@gmail.com is NOT in database
	Other.com 123456 123456	Error: invalid email	Error: invalid email	Pass	
	test@gmail.com	Error: no password	Error: no password	Pass	
	test@gmail.com 123456 12345	Error: Passwords do not match	Error: Passwords do not match	Pass	
	test@gmail.com 123 123	Error: Password too short	Error: Password too short	Pass	
	test@gmail.com 123 12	Error: Passwords do not match	Error: Passwords do not match	Pass	
	test@gmail.com	Error: No Passwords	Error: No Passwords	Pass	
Account Sign-in	test@gmail.com 123456	Successful Sign in	Successful Sign in	Pass	
	test@gmail.com 12345	Error: Wrong Password	Error: Wrong Password	Pass	
	test@gmail.com	Error: No password	Error: No password	Pass	
	123456	Error: No email	Error: No email	Pass	
	noaccount@gmail.com 123456	Error: no account with that email	Error: no account with that email	Pass	

Notification Settings	Still needs to be implemented				
Posting Announcements	Add post	Post saved	Post saved, but lost on refresh	Failed	None
	Delete Post	Post Deleted	Cannot test, no delete button	Failed	At least one post exists
	Delete another person's post as user	Post cannot be deleted	Cannot test, no delete button	Failed	
	Delete another person's post as admin	Post deleted	Cannot test, no delete button	Failed	
	Edit Post to add "edit"	Post is updated with edit	Cannot test, no edit button	Failed	At least one post exists
Messaging	Send Message "test" to test@gmail.com	Message sent to user	Cannot test, no message page	Failed	Must be signed in
	View Messages from test@gmail.com	Message "test" is viewable	Cannot test, no message page	Failed	Must be signed in as test@gmail.com
	Send Message ""	Nothing sent	Cannot test, no message page	Failed	Must be signed in as test@gmail.com
	View message history from test@gmail.com	Full history of tests viewable	Cannot test, no message page	Failed	
View Analytics	View analytics page	Full list of names viewable	Full list of names viewable	Pass	
	Select "Nathan Bunge"	Map shows Nahtan Bunge's location	Map shows Nahtan Bunge's location	Pass	"Nathan Bunge" must be in database
		Shows Company	Shows Company	Pass	

		Shows contact information	Shows contact information	Pass	
		Shows title and experience	Shows title and experience	Pass	
	Select user on map in China	"Tom Arad" is shown	"Tom Arad" is shown	Pass	"Tom Arad" must be in database with "China" as location
		Shows Company	Info not shown	Fail	
		Shows contact information	Info not shown	Fail	
		Shows title and experience	Info not shown	Fail	