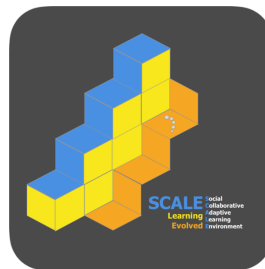


# ***Pathways Mobile***

*Building a Mobile App for Students and Universities from the Ground Up*

SCALE Learning Technologies



## **Mentors:**

Tiffany Reiss, Osman Bakari, Jack Wharton, Jimmy Zheng, Christopher Nathman, Matthew Johnson, Ernest Spicer

## **Team LD:**

Ganeshram Krishnamoorthy, Caleb Lee, Yihui Fang, Wenda Liu

## **TABLE OF CONTENTS**

<b>I. Introduction</b>	<b>4</b>
<b>I.1. Project Introduction</b>	<b>4</b>
<b>I.2. Background and Related Work</b>	<b>4</b>
<b>I.3. Project Overview</b>	<b>5</b>
<b>I.4. Client and Stakeholder Identification and Preferences</b>	<b>6</b>
<b>II. Team Members - Bios and Project Roles</b>	<b>6</b>
<b>III. Project Requirements</b>	<b>7</b>
<b>III.1. Use Cases</b>	<b>7</b>
<b>III.2. Functional Requirements</b>	<b>10</b>
<b>III.2.1. User information</b>	<b>10</b>
<b>III.2.2. User Authentication</b>	<b>10</b>
<b>III.2.3. Data Records</b>	<b>10</b>
<b>III.3. Non-Functional Requirements</b>	<b>10</b>
<b>III.3.1. Easy to Use</b>	<b>10</b>
<b>III.3.2. Reliable</b>	<b>10</b>
<b>III.3.3. Extensible</b>	<b>10</b>
<b>III.3.4. Maintainable</b>	<b>10</b>
<b>III.3.5. Easy To Iterate Upon</b>	<b>10</b>
<b>IV. Solution Approach</b>	<b>11</b>
<b>IV.1.2 Subsystem Decomposition</b>	<b>12</b>
<b>IV.1.2.1 Frontend</b>	<b>12</b>
<b>a) Description</b>	<b>12</b>
<b>b) Concepts and Algorithms Generated</b>	<b>12</b>
<b>c) Interface Description</b>	<b>12</b>
<b>IV.1.2.1 Backend</b>	<b>13</b>
<b>d) Description</b>	<b>13</b>

e) Concepts and Algorithms Generated	13
f) Interface Description	13
VIII. Test Plan	17
VIII.2.3.System Testing	19
VIII.2.3.2.Performance testing:	19
VIII.2.3.3.User Acceptance Testing:	19
IX. Pathways Mobile Prototype Description	20
IX.1. [REACT NATIVE FRONT END]	20
IX.1.1.1. Login Screen	21
IX.1.1.2. Create Account Screen	22
IX.1.1.3. Forgot Password Screen	23
IX.1.1.4. Profile Screen	24
IX.1.1.5. Join University Screen	25
IX.1.1.6. Select University Degree Program Screen	26
IX.1.1.7. Schedule Screen	27
X. Pathways Mobile Prototype Demonstration	28
XI. Future Work	29
XII. Glossary	29
XIII. References	30
XIV. Appendices	32

## I. Introduction

This document serves as a culmination of Team LD's work on the SCALE *Pathways* mobile app. The purpose of this project is to provide students with a way to create and manage their academic schedules throughout their college journey. This document will provide extensive details on the project overview, team member bios, project requirements, and specifications. Finally, this document will provide details on the status of the *Pathways* app prototype. The team will share the current progress on the prototype as well as describe what the prototype looks like.

### I.1. Project Introduction

Students choosing to pursue higher education face many difficult decisions. These include choosing the institution that best fits their needs, selecting the degree(s) they wish to pursue, and picking the individual classes that they will be taking each semester or quarter. These considerations are incredible pain points for students; it has been found that 32.9% of undergraduates are not able to complete their degree programs [1]. The factors that cause these students to drop out include attending an institution that does not match the individual's preferences, switching majors after a change of heart, and having too difficult of a course load. Students are overwhelmed by the incessant number of decisions regarding universities, degrees, and courses that they will have to make during their higher-education journey. SCALE *Pathways* will be a cross-platform mobile application that aims to address these issues. It will streamline the process of selecting a degree from an institution, tracking one's progress through time for their degree, and making adjustments to any hurdles faced. With these solutions, *Pathways* hopes to take a load off the shoulders of students around the country.

### I.2. Background and Related Work

Team LD's primary contribution to *Pathways* will be implementing the mobile version of the existing robust course planning website prototype. It is built with the motivation that choosing the correct path in higher education is a critical issue. In recent times, this issue has been further exacerbated by the pandemic in which students have experienced significant reductions in learning efficacy. This has particularly harmed students who do not have access to adequate financial and social resources. An example of these unfortunate outcomes is the situation at QC, an urban college with a socially vulnerable student population located only three miles from the epicenter of New York City's COVID-19 outbreak in 2020. The pandemic reduced the freshman retention rate by 26% and altered the graduation plans of 30% of all QC students [3].

To make it easier for students to navigate through issues such as these, there are currently numerous apps available that track one's academic schedules. One of the most popular apps in this category is Class Timetable [4]. With its minimalist design, this app allows students to enter the courses they are taking and their accompanying tasks. It can also display its information in a week view. Another similar app is Schooly [5]. It features a clean user interface with Notes, To-Do's, and different themes. Like Class Timetable, Schooly lets the user manually enter their courses and their tasks. The last app we will discuss in this space is Class Schedule Planner & Tasks [6]. This app distinguishes itself in this space with extensive voice assistant shortcuts, schedule sharing, statistics, and many schedule viewing options.

The current solutions on the market lack the ability for learning institutions, such as universities, to enhance the schedule-tracking process for students. The primary purpose of each of the above-mentioned apps is to help students track their academic schedules, and yet, they all do so in a manner that does not allow the entity creating the schedules to participate. Learning institutions like universities would be incentivized to establish this communication link

as it would help attract students to their degree programs. This hole in the market is what SCALE *Pathways* Mobile hopes to fill, offering the unique advantages of being able to select standardized classes, academic schedules that are reactive to changes from both the learning institution and a student, and the ability to compare and contrast different schedules, and more.

### **I.3. Project Overview**

With multiple fields of study in a university, it is intimidating for students to plan an optimal course schedule throughout their years in college. SCALE *Pathways* Mobile is an application that assists current and incoming students with their scheduling based on their major. On a basic level, students will be able to input their university, major, and courses. With this information, the application will automatically generate the optimal schedule for the student throughout the university. SCALE *Pathways* Mobile offers unique advantages in comparison to its current market competitors, such as being able to select standardized classes, academic schedules that are reactive to changes from both the learning institution and a student, and the ability to compare and contrast different schedules.

This schedule created by SCALE *Pathways* Mobile will be modular. Additionally, it can be revised or regenerated based on class availability, changes to graduation requirements, and if a student needs to retake a class. The schedule will be divided into separate terms to clearly show the student which classes to sign up for. Classes will also be categorized and highlighted (i.e., UCORE, Major Class, CAPSTONE, etc.). Classes can be added in bulk via CSV (.csv) files. The schedule can be saved or reverted based on each user. Previously saved schedules can also be accessed, as well as making multiple schedules for each student.

University admins (advisors) will be able to customize different courses as they see fit. Adding details such as course codes, credit hours, descriptions, and availability. Additionally, admins will be able to generate their own pathways to graduation based on specific majors.

This project is based upon an existing web application “SCALE *Pathways*”, however, the mobile version will be built from the ground up. Using the React Mobile framework, the application will be built to support both iOS and Android devices. In previous iterations of the web application, there was an implementation of graphs to build the schedule. We will be iterating on that code and improving the functionality to fit all major requirements. A validation algorithm was also used to check and maintain course dependencies to ensure a correct schedule. We will be iterating on top of the previous version to improve handling and a possible refactor.

SCALE *Pathways* Mobile must be built to be iterated upon, different classes and majors are always being added and removed. Major requirements can change from year to year, so it is essential that the code uses precise software engineering and object-oriented principles to maintain scalability.

An optional objective that Team LD would like to include is automated functionality with university systems. Currently, universities have to manually provide their course information. It would be easier to integrate more universities into SCALE *Pathways* if we implemented a wrapper or other software solution to automate the process by which their course and university information is uploaded onto the app. This objective will be difficult to scale as different universities use different legacy APIs with their scheduling software. However, implementing this feature with WSU would be our primary objective for this year.

Another optional objective that LD would like to implement is “the ability to construct schedules that targets a specialization within a major, such as the AI/ML path for WSU’s

Computer Science B.S. Degree. This could be implemented by using tags, in which the app would create a specific course schedule for the student prioritizing courses with the corresponding tags if they wish to pursue the ML specification.

#### **I.4. Client and Stakeholder Identification and Preferences**

Our team has identified two clients, which are students and universities. Students are clients since our app aims to help students to build their schedules using the major that they have picked. If students are using the app, Team LD can gather data with their consent and improve the app. Universities can be a client if they choose to cooperate with the team. This is because the app would require full access to a university's API in order for the app to aid the students in creating a schedule for them. The stakeholders are Tiffany, Osman, Jack, Jimmy, Christopher, Matthew, and Ernest. This project requires the team to build an app from the ground up to aid students to build a course schedule based on their major, and the university they are attending.

## **II. Team Members - Bios and Project Roles**

Wenda Liu is a computer science major with an interest in machine learning and web applications. Wenda has prior experience in building front and back end for web applications. Wenda has a wide range of languages he can use like C/C++, C#, Python, and HTML. Wenda is responsible for team management.

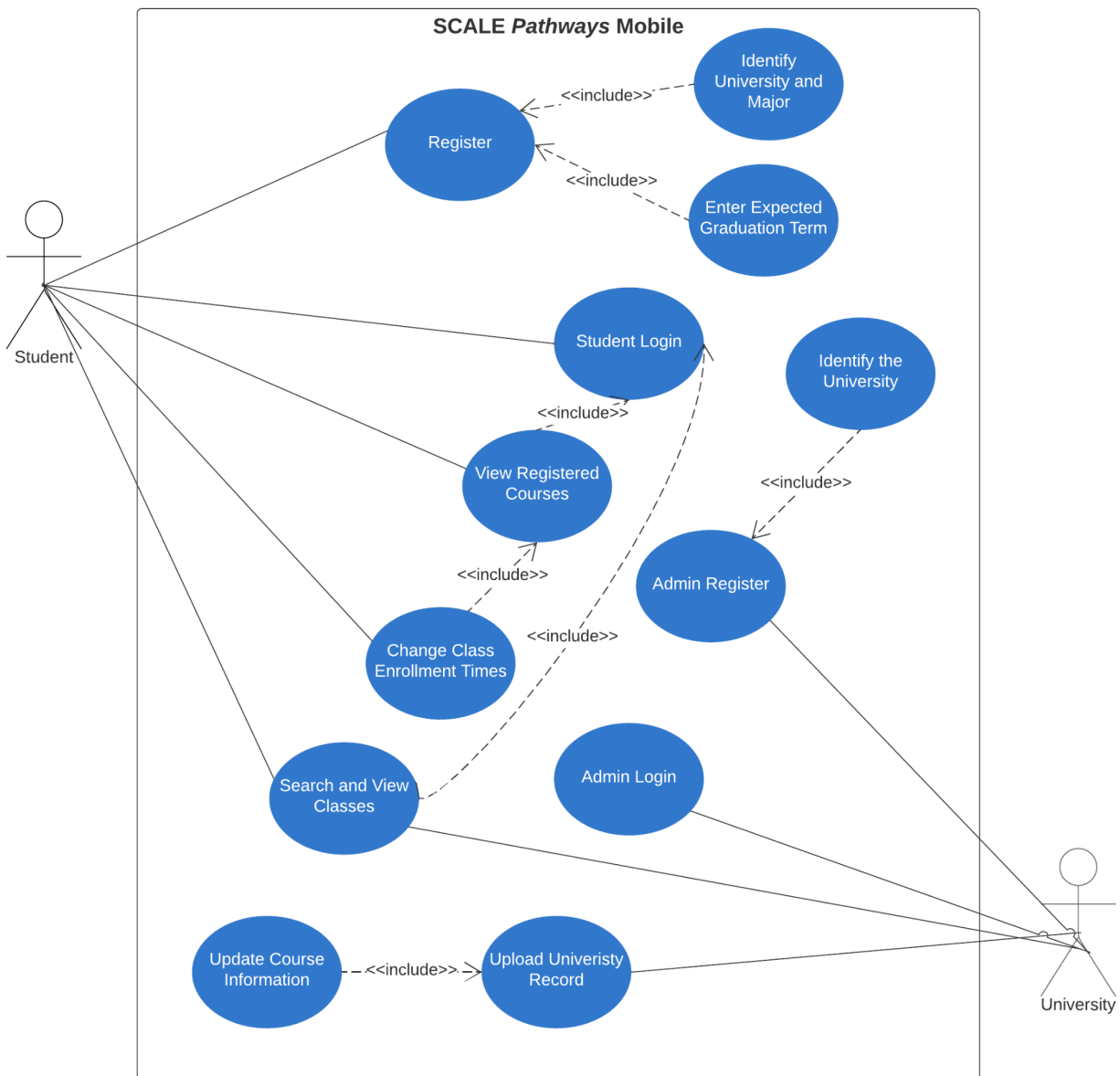
Caleb Lee is a senior computer science major that wants to go into the field of software engineering or cyber security. Caleb has experience with college-level cyber security competitions. Caleb has experience with C/C++, C#, Python, and Haskell.

Yihui Fang is a senior Computer Science student with an interest in software development. Yihui has held teaching assistant positions in Data Structure class, completed a Front-end Engineer, and participated in a deep learning-based vulnerability research program. Yihui has a good understanding of C/C++, Python, JavaScript, and Haskell.

Ganeshram Krishnamoorthy is a senior Computer Science B.S. student. Ganeshram has used JavaScript frameworks such as Ionic Framework to build native cross-platform applications for iOS and Android. Ganeshram's team utilized machine learning to rank second place in WSU's Digital Agathon 2020. Ganeshram is interested in machine learning and artificial intelligence and is proficient in C/C++, Python, and JavaScript. Ganeshram is the team leader.

### III. Project Requirements

#### III.1. Use Cases



Name	Course Schedule
Description	First, a student needs to be able to register and identify their university. Next, they should be able to view and enroll in courses.
Actors	Student
Pre-Conditions	<ul style="list-style-type: none"><li>• The user must be logged in with a registered user account.</li><li>• The user must have identified their university</li></ul>

Flow of Events	<ol style="list-style-type: none"> <li>1. The student selects their major from the list of offered degrees at their registered university.</li> <li>2. The website populates the course table with the template schedule for that degree.</li> <li>3. The student clicks on the course that interest them for more information</li> <li>4. A pop-up box will be displayed containing course details, meeting location, and time.</li> <li>5. The student can manually push back the course they choose to next semester.</li> <li>6. As a result, classes in the following semesters that prerequisite classes would be pushed back too</li> <li>7. The table's credit totals for each semester are updated to reflect the change.</li> </ol>
Related Requirements	<ul style="list-style-type: none"> <li>• The application must be able to populate the class table with a template schedule for every provided degree.</li> <li>• Clicking on a class must direct the user to more information about that class.</li> <li>• The user is able to manually move classes to the future semesters</li> <li>• The application must be able to readjust the schedule to account for changes made by the user</li> <li>• The class table must display the credit load for each semester</li> </ul>
Post-Conditions	<ul style="list-style-type: none"> <li>• The course table must be updated with the correct courses, with no relationship conflicts. For example, students can't take a course with its prerequisite class in the same semester.</li> </ul>
Exceptions	None.

Name	Invalid Class Move
Description	Student attempts to move a class which might affect the graduation date
Actors	Student
Pre-Conditions	<ul style="list-style-type: none"> <li>• The user must have been logged in with a registered user account.</li> <li>• The user must have identified their university</li> <li>• The user must select a degree and have a schedule loaded</li> </ul>
Flow of Events	<ol style="list-style-type: none"> <li>1. The student attempts to move the next semester's class to the previous semester</li> <li>2. The application realizes that there is no way for the student to take that class</li> <li>3. The application reverts the change.</li> <li>4. The application displays an error message informing the student that the move is not possible.</li> </ol>
Related	<ul style="list-style-type: none"> <li>• The user is able to manually move classes between semesters.</li> </ul>



Requirements	<ul style="list-style-type: none"> <li>• The application must be able to readjust the schedule to account for changes made by the user.</li> <li>• The application must be able to recognize invalid schedule changes.</li> </ul>
Post-Conditions	<ul style="list-style-type: none"> <li>• The course table is returned to the state it was in before the attempted change</li> </ul>
Exceptions	None.

Name	Post and Update Course Enrollment
Description	A user with admin permissions for a given university uploads courses and record data for that university.
Actors	University Official
Pre-Conditions	<ul style="list-style-type: none"> <li>• The user must be logged in with a registered admin account</li> <li>• The user account must have admin permissions to at least one university</li> </ul> <p>The user must have already formatted their data into the desired CSV format.</p>
Flow of Events	<ol style="list-style-type: none"> <li>1. The user navigates to the profile page and selects the “Admin” tab.</li> <li>2. The user clicks the “upload university record” button, and selects their preformatted CSV file.</li> <li>3. The application shows a message verifying the success of the operation.</li> <li>4. The course view page is refreshed to reflect the updated courses.</li> </ol>
Related Requirements	<ul style="list-style-type: none"> <li>• Allow the user to import course information from a standard file format:</li> <li>• Provide a sample file to demonstrate the desired format</li> <li>• Allow a university official to upload a CSV file of student academic records</li> </ul>
Post-Conditions	<ul style="list-style-type: none"> <li>• The new data is entered into the database.</li> </ul>
Exceptions	None.

## III.2. Functional Requirements

### III.2.1. User information

**Gathering User Information:** the applications would require each student to use university and major information in order for the application to create and manage the student schedules.

**Source:** Primary stakeholders with SCALE originated this requirement. This requirement is necessary for the application to work.

**Priority:** priority level 0: essential and required

### III.2.2. User Authentication

**Authentication:** the application must use university specific email to authenticate the student.

**Source:** Primary stakeholders with SCALE originated this requirement. This requirement is necessary for users to access their own applications.

**Priority:** Priority level 0:essential and required

### III.2.3. Data Records

**Storing the user data:** The application will store the student data in a database and the user must be able to view and change them.

**Source:** Primary stakeholders with SCALE originated this requirement. This requirement is necessary for the application to view data.

**Priority:** Priority level 0:essential and required

## III.3. Non-Functional Requirements

### III.3.1. Easy to Use

This tool should be intuitive for its targeted demographic of college students. This is necessary for the continued use and proliferation of the SCALE *Pathways* mobile application.

### III.3.2. Reliable

The application must be free of bugs and issues that prevent the application from functioning nominally.

### III.3.3. Extensible

It must be simple for universities to integrate their APIs with SCALE *Pathways* mobile as this capability is integral to the application's functionality.

### III.3.4. Maintainable

SCALE *Pathways* mobile must be built with robust design principles so that it is simple to maintain. Without this requirement, users will encounter negative downstream effects such as slow-to-respond pages and loss of data, leading them to cease their app usage.

### III.3.5. Easy To Iterate Upon

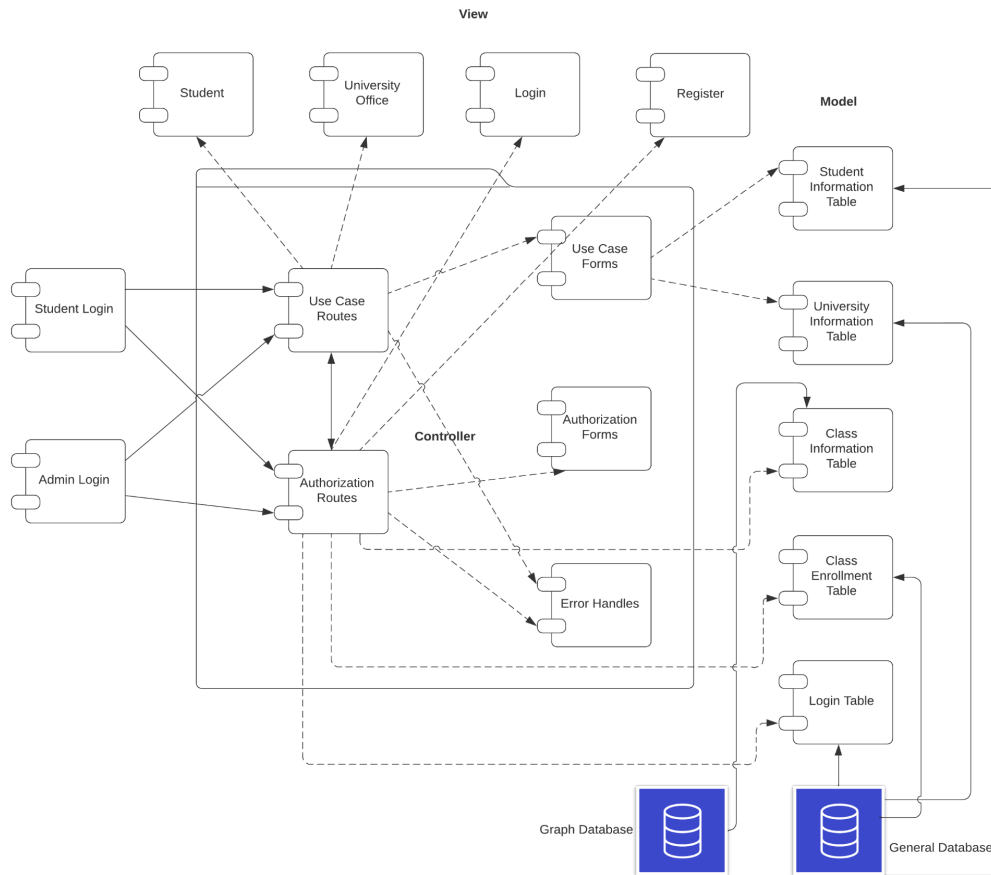
In order to address user feedback and implement new features, SCALE *Pathways* mobile must be built in a manner that makes it easy to iterate upon.

## IV. Solution Approach

### IV.1. Architecture Design

#### IV.1.1 Overview

Team LD has decided to use a cross-platform application architecture for this application [2]. This allows SCALE *Pathways* Mobile to reach iOS and Android markets with a single, unified codebase. This design decision has benefits and tradeoffs. As stated, the primary use of this approach only has to maintain a single version of the app rather than two distinct versions of the app for iOS and Android. The primary tradeoff is poor performance in high-performance and graphics-heavy applications, neither of which apply to SCALE *Pathways* Mobile. Thus, it is the most suitable choice for us. The mobile platform would contain a Frontend for user interaction and a Backend for requesting services. The team has included a diagram detailing how we would develop our application based on this diagram. As for this, we would provide a general idea of our diagram and go into more detail of its subsystem decomposition. The Frontend will assist with the initial decision that a user makes, which is to select whether they login or create a Student account or a University Admin account. After the user has selected an account type the Frontend will communicate with the Backend via Use Case Routes or Authorization Routes. This component is important to determine the type of account the user has and what kind of access the user will be granted while using the application. Starting with the Use Case Routes, if the user is logged in as a Student but has no data within their account, they only have access to their student profile and a general landing page. While if the user is logged in with a University Admin account with no data, they only have access to their university office profile and University information table which is a general landing page. The Frontend would change dynamically depending on the information that the user provides. For the Authorization Routes, if the user is a student and they have data on their schedule, they would have access to a new landing page that would benefit them more. For example, when the Student joins a University, they would be greeted with a page displaying the University's degree programs rather than the general landing page they were shown previously. If this user joins a degree program, the Authorization Routes would detect this change and present them with a new landing page that shows their Class Enrollment. While on the University Admin side, if the account contains data with the user's identity and the correct permissions have been granted, the user would have access to a Class Information page and be able to perform changes. More details on these object components can be found in the Subsystem Decomposition section.



## IV.1.2 Subsystem Decomposition

### IV.1.2.1 Frontend

#### a) Description

The Frontend is responsible for giving the user a way to interact with our API. The Frontend is mobile-based which requires JavaScript and JSX to make each mobile page dynamic. The mobile page styling will be built upon JSX.

#### b) Concepts and Algorithms Generated

The Frontend will be written in JavaScript or TypeScript, which is a superset of JavaScript, but rendered with native code through React Native. In this framework, rendering logic is coupled with UI logic, and they are handled through components. This is done because this simplifies the Frontend design process [3]. React Native has wrapped many existing elements on Android and iOS into components, and any further elements that we create can be wrapped as well. No business logic of SCALE *Pathways* Mobile will be implemented on the Frontend, as the Frontend runs on each user's personal device.

#### c) Interface Description

The Frontend interface is running through API calls to the Backend. The API will use GET and POST to receive and display the data.

Services Provided:

### 1. User Interface

The user interface will allow the user to enter their information into the Frontend where then the Frontend will communicate with the Backend.

Services Required:

1. **Backend**

The Backend will be used to make the Frontend a dynamic application. All application data displayed to the user through the Frontend will be stored in the Backend.

**IV.1.2.1 Backend**

**d) Description**

Since our project exists within the SCALE *Pathways* application, we will continue to use Python and the Flask web framework to develop the Backend. However, to increase productivity in the long term, we will be utilizing the SQLAlchemy [4] and py2neo [5] object relational/graph mappers. Utilizing these tools will reduce the amount of code required to translate between native Python objects and data within our databases. We will also request access to WSU's API, which would give us an extensive list of classes, help identify class prerequisites, generate class schedules, and more.

**e) Concepts and Algorithms Generated**

The mobile app application will only serve API requests, the database will only handle user registration information, generate student's four years course plan and validate schedule changes.

**f) Interface Description**

The Backend integrates with all parts of the application, including the Frontend, MySQL database and Neo4j graph database.

Services Provided:

The business rules of how the user interacts with the data stored in each database are contained in the database subsystem.

Services Required:

1. **Neo4j Database**

Neo4j will be used to generate the graph database, it will show the dependencies for each degree plan offered by the university office.

2. **Frontend**

The mobile-based Frontend will consider the user interface and guarantee the user's experience.

**IV.2. Data Design**

The previous Pathways team, called Team Artifact, had implemented a SQL database designed to store university data and user data. They also implemented a graph database to store information about classes and the relationships they have with one another. Our team will be continuing to utilize this database design, with both the SQL database and the graph database. Our team will be including one new schema to the Artifacts SQL database. It is called the User Data schema and will contain the user's email, name, major, and bio.

**Course Schema Updates:**

- Instruction Team Table

- Represents a university semester, quarter, or term
  - References a university ID
  - Identified by a unique ID
- Course Offering Table
  - Represents a specific offering of a general course
  - References an instruction term ID and a course ID
  - Identified by a unique ID

#### **Degree Schema Updates:**

- University Degree Table
  - Represents a specific degree earnable at a university (BS or BA)
  - References a university ID, major ID, and a course ID
  - Identified by a unique ID

#### **Schedule Schema Updates:**

- Schedule Table
  - Represents a schedule saved by a user
  - References a user ID
  - Identified by a unique ID
- Schedule Slot Table
  - Represents a course offering that is saved to a schedule
  - References a schedule ID and a course offering ID
  - Identified by a unique ID

#### **Student Record Schema Updates:**

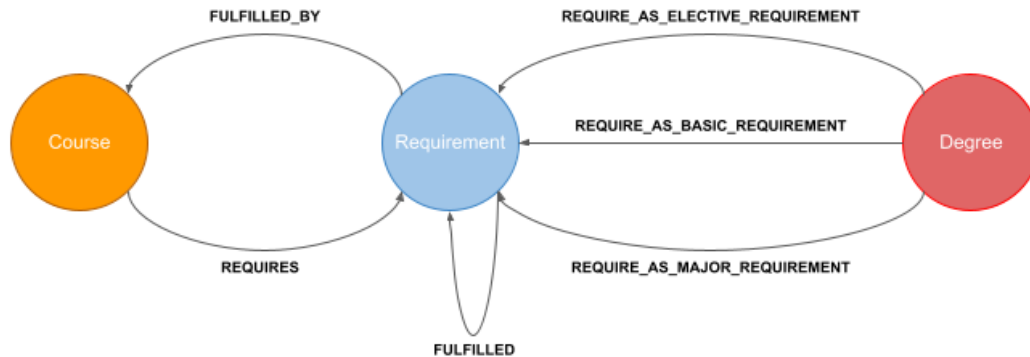
- Letter Grade Table
  - Represents a grade on a specific university's grading scale
    - Enables support multiple grading scales
  - References a university ID
  - Identified by a unique ID
- Course Record Table
  - Represents a record of a student's grade in a past course
  - references a course offering ID, a user ID, and a letter grade ID
  - Identified by the unique course offering ID and user ID pairing

#### **User Data Schema Updates:**

- Email
  - Represents an email saved by a user
- Username
  - Represents a name saved by the user
- AccountType
  - Represents the user's type of account
- Bio
  - Represents a bio saved by the user
- Majors
  - Represents majors saved by the user

#### **The Neo4j Graph Database**

A graph database consists of nodes and directed relationships, each of which has a label to determine the “type”, and attributes to store information. While Neo4j does not strictly enforce a schema, having a consistent structure for our data makes it much easier to retrieve the information we need.



This schema consists of three different types of nodes and six different types of relationships that they can have between them. With these relationships, we will be able to perform all the required logic to generate schedules, and auto-adjust schedules based on the department of courses.

## Node Types

### V. Course

V.1. This will store the course ID, and be how we primarily retrieve class information

V.2. Attributes: like title, course number, and prefix

### VI. Requirement

VI.1. This represents a prerequisite requirement, degree requirement, or elective requirement relationships

VI.2. Attributes: name

### VII. Degree

VII.1. This represents a specific degree offered by the university, for the purpose of associating courses as required or part of a template

VII.2. Attributes: name, track

## IV.2. User Interface Design

Team LD has created a Figma prototype for the application that the team envisions. Figma allows the team to create a semi-functional application prototype which can be interacted with in a similar manner to a real application. In the Figma prototype, we have created a loading page when a user has clicked on the application. Next, after the loading is done, it will bring the user to a login page where they could either enter their profile information to log in, or create a new account. If the user does not have an account, they can create a Student account or a University Admin account. For account creation, a screen is displayed which allows the user to enter their information. Let us first presume the user creates a Student account. The Student account will be greeted with a landing page upon signing in. The content of the landing page depends upon the user's state. If this is the first time the user has signed in, it is likely that the user is not already enrolled in a university. Thus, the landing page will display universities in a list view, which the Student user will be able to interact with. They will be able to find more information on the university by clicking on its UI element, such as the cost of attendance and a description. Once a Student user has joined a university, the landing page will show the degree

programs available in their university in a list view. By interacting with a degree's UI element, the user will be shown information such as a course outline. Once the student selects a degree program, the landing page will display the user's schedule. Each iteration of the landing page and all other pages will contain an options icon in the top left corner. When clicked, this will open a pop-up providing the user with quick, easy-to-access links to each page in the mobile application such as Schedule, Degrees, Universities, Classes, and Account Settings. Another thing the user would be able to interact with is their profile page, where the user can update their personal information like name, email, and major. Now, let us presume that the user creates a University account. Upon sign-in, the user will be shown a landing page that will allow the user to perform Degree Management actions. The screen will display a list of the degrees that the University Admin user is in charge of. From this screen, they can search for degrees or classes. When they select one of these degrees, the page will then display a list of classes that belong to this degree. When a class is selected, the user is then able to edit its details. The user can also add more classes or degrees. Since this is the first stage of the prototype there are areas the team can refine and change after some testing with actual usage from users. Below are some brief descriptions for our current screens, which are displayed in the **Appendices** section.

#### *Figure 1*

From left to right, we have displayed the loading screen, the sign-in page, and the menu. The loading screen will be shown immediately after the user enters the app while content is being processed. The sign-in page will be displayed if the user has not already logged on with an account. After the user has signed in, it will no longer display that page and instead display their landing page, unless the user has signed out. Finally, the menu UI element will be shown when the user clicks on the three horizontal bar icons in the top left after signing in. It will allow the user to navigate to different screens.

#### *Figure 2*

These are the landing pages for account creation, account editing and university selection. Once the user completes filling out all their information they will be taken directly to the university landing page to start the process of generating their schedule. Any edits that the user needs to make to their account can be made on the second landing page and is accessed through the menu icon (indicated by the three lines on the top left of each landing page).

#### *Figure 3*

The first image displayed is what the University account Sign-up page could look like. It asks the user to enter the following fields: the user's first name, last name, university email, password, and their university. The next image displays the University Admin's account settings. Here, they will be able to update their information should they choose to. The final image displays how a University Admin user would be able to filter classes and edit them.

#### *Figure 4*

From left to right, the screens shown are the reset password page, university details expansion, and degree selection landing page. The first screen demonstrates how the user will be able to reset their password, and this page will only be accessible through the login page or in the account editing option. The next screen shows what the app will look like when a Student user selects a certain university. The details will include: average tuition/cost, biography of university, location, and other important information that each university would like to add. After



the Student user selects a university, a user will be able to select a certain degree that is offered at the university. The degrees are depicted in a list view. This is what is shown in the last screen.

*Figure 5*

From left to right, we have the Degree Management screen for University Admin accounts, the Course Management screen for University Admin accounts, and a screen that allows Student users to inspect classes in their degree. The first screen allows University Admin users to view and select any of the degrees that they manage. The next screen displays what the University Admin user sees when they select a degree to manage. They are shown a list of classes that must be completed in that degree program, which they can select to edit. The final screen displays So whenever students enroll or switch classes, the program will use the degree management list Admin added for reference. Course management is for admin to update all the prerequisite classes. Both students and admin can view and access the courses list for each major.

*Figure 6*

The first screen shows the landing page a Student user will see when they have selected a university and enrolled in a degree program. The Student user can see and analyze their auto generated schedule, as well as make changes to it. When a user taps on one of the classes, the bubble will expand to show more details about the class. This is what is shown in the second screen.

## VIII. Test Plan

### VIII.1. Testing Strategy

For the testing strategy, Team LD will take a Behavioral Driven Development (BDD) model to test SCALE Pathways [1-2]. The first priority of Team LD is to create a working mobile app that mirrors the web application. Testing cycles will be based around objectives that Team LD would like to accomplish. Developers will write code to create an initial implementation of new features and then write test code to ensure that each feature is passing. Extraneous cases will be written but focus will be on realistic cases to meet the time constraint of each sprint. If an extraneous test case cannot be completed but will affect the timeline of the project, it will be put on the back burner and will be returned to if there is a major need.

Below are the steps of the **testing life cycle** Team LD will use. The timeline will be a 2-week pseudo sprint cycle for development and testing. Team LD will use a continuous integration (CI) model to ensure that the application is working properly every time new features are pushed. At the moment, SCALE Pathways mobile is not in a sufficient enough state to implement continuous development (CD) testing.

- a. **Meet with clients to gather requirements:** Team LD will use the bi-weekly meetings to obtain the initial requirements by our sponsors.
- b. **Developer writes code to create an initial implementation:** Team will split up requirements and implement them separately. Team members will constantly stay connected to ensure collaboration. Push separate branches based on requirements to the remote repository.

- c. **Developers/Testers create test cases based on client requirements:** Developers will write code for test cases. Each developer will contribute test cases for each other to cover for missing use cases.
- d. **Developer runs tests on code:** The test programs will be executed, and the developers will assess if the code needs more implementation.
- e. **If testing fails, developers rewrite code to fix bugs and retest:** Go back to step b, if there is a roll over on a feature, the team will bring it up to the sponsor, in that case go back to **step a** to reassess.
- f. **If testing passes, push and tag changes to remote repository:** Pushing each developer changes to an individual branch in the remote repo and tag their commit they want to merge for the time life.
- g. **Once all new features have been pushed to remote repositories, merge branches together:** Needs to be approved by the majority of the members of the development team (3/4) team members.
- h. **Once the app is in a suitable beta state, test it with users:** Deploy the mobile app to a variable tester for feedback. Send out google forms to gather information and feedback.
- i. **Repeat steps a-i with new requirements/feedback from tested users.** Gather all user feedback and deliver it to the client. Create new requirements with the client based on the feedback.

Team LD wants to focus on collaboration throughout the testing process. Though we are going to have individual tasks that each developer will work on, each member of the group will take a look at each other's code to make sure they are implementing it properly. This goes for test cases as well. Having each member create test cases, Team LD can properly cover for missing test cases ensuring more optimal testing.

## VIII.2.Test Plans

### VIII.2.1.Unit Testing

For unit testing, we will follow rigorous guidelines. Firstly, we will ensure that all crucial functions have around three unit tests. The first unit test will consider a normal case. The second unit test will consider an edge case. The final unit test will consider an exception case. There may be more than one test case each of these, and it may also be that an edge case or an exception case cannot be tested because they are inappropriate for the function. By crucial functions, we are referring to complex functions that perform key functionality that is of utmost importance to the app's operation. We will also implement test functions for non-crucial functions, however, it cannot be guaranteed that all such functions will have testing methods as it would be unnecessary. An especially trivial function will definitely not be unit tested as this would be a waste of time and resources. Team LD will discuss amongst themselves in order to accurately determine which functions are crucial and require testing, and which functions do not need testing.

### VIII.2.2.Integration Testing

In a mobile application, a feature may contain multiple modules that must communicate with each other. Integration testing will test the contract between functional modules and check that the modules interact as planned [5]. We will also test whether React Native successfully communicates with two different operating systems: Android and iOS. React components are responsible for rendering the app and the user interacts directly with the components. The core parts of testing React Native components can be separated into two parts:

- Interaction: make sure that the component works properly when interacting with the user (e.g., when the user clicks a button)
- Rendering: make sure that the React component is displayed correctly (e.g., how the button looks on the page and where it is positioned)

### VIII.2.3.System Testing

System testing is a type of black box testing in which all components are tested together. The app is seen as a single system to identify faults that violate the overall requirements and specifications.

#### VIII.2.3.1.Functional testing:

In order to perform the functional testing for SCALE *Pathways* Mobile, we will be following the guidelines discussed in our Requirements and Specifications section. More specifically, we will test the following:

- **User Information correctly displayed:** In the Profile screen, as well as elsewhere, the user's information should be displayed concisely and accurately.
- **User Authentication is successful:** Users should be able to make accounts, sign-in, and sign-out.
- **Data is Recorded correctly:** Any states that are modified by the user should be accurately updated in the frontend and the backend.
- **UI is easy to navigate and use:** The user interface should be pleasing to look at, and it should be simple for the user to interact with it.
- **Application is reliable:** The application should not crash or perform unexpectedly.
- **Application can be extended to include other APIs:** It should be possible to integrate other University APIs.
- **Application can be updated easily:** The users will be greeted with new features as the application is built in a manner that allows for easy updates.

#### VIII.2.3.2.Performance testing:

At the moment, Team LD does not have a fully-functioning application where we can conduct a survey on the performance of the application. In the future, the team plans to distribute the application to students and have to give the team feedback on a scale of 1-10 on the performance of the application.

#### VIII.2.3.3.User Acceptance Testing:

For acceptance testing the team will meet with the client every two weeks throughout the project to give them updates on development progress and development and demonstrate any new work. This demo with the SCALE team has given us feedback on how they envision users would use the application, which is helpful to the team since it is quick feedback on how they believe users would get confused about some parts of the application. In the future when the application is fully working the team plans to release the application to select a few students and have them use the application and give us feedback so we can improve the user ability of the application.

### VIII.3.Environment Requirements

Specify both the necessary and desired properties of the test environment. The specification should contain the physical characteristics of the facilities, including the hardware, communications and system software, the mode of usage (for example, stand-alone), and any other software or supplies needed to support the test. Identify special test tools needed.

## IX. Pathways Mobile Prototype Description

### IX.1. [REACT NATIVE FRONT END]

#### IX.1.1. Screen Layout and Functionality

Using REACT Native, Team LD used Flexbox, a layout model built into CSS. Using three main containers: Screen, content and navigation bar. Development of new screens can be implemented easily. Figure 1.0 shows how team LD was building screens before and after utilizing Flexbox to its fullest potential.

Prior, team LD had only one View component that housed every interactable object in the screen. This would lead to issues when implementing a navigation bar as the navbar would be placed relative to the last component of the screen, not the bottom. It was also very confusing in the code where we would have to place a component to get it on a certain part of the screen. Additionally, components were margined using pixel values, this would lead to issues when using SCALE Pathways mobile for different sized devices.

The NEW part of figure 1.1 is the solution we implemented to fix this issue. The purple box is a fixed size view container that is 100 percent of screen size. The *screen container* is using flex to distribute the two inner orange containers. The top container is the *component container* that is used to hold all buttons, titles, etc. The *component container* can also utilize flex to add more components in a customizable way.

The bottom container is the *navbar container* this is fixed to a certain size and is solely used to hold the navigation bar. You can see in figure 2.0: There is no container surrounding the “HamburgerMenu” component. That is because this last bottom container is implemented in the menu code itself. This will assure that the team will be able to just implement a navbar at the end of the screen to ensure that it will be set at the bottom.

Using Flexbox also allowed us to manipulate margins using a percent value of a certain container. Meaning that component placement will be consistent throughout different sized devices.

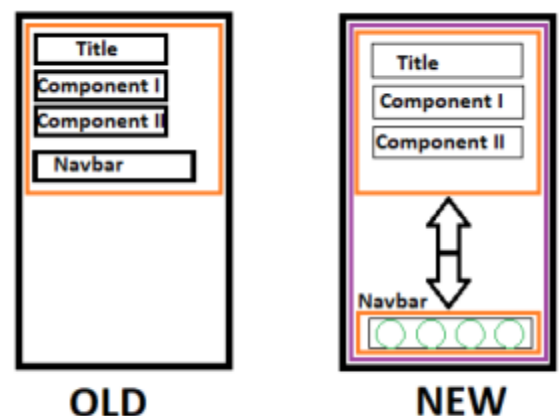


Figure 1.1: Before and After Flexbox

```
return (  
  <View ScreenContainer style={styles.outterContainer}>  
    <View ComponentContainer style={styles.contentContainer}>  
      <Search_Bar></Search_Bar>  
      <Text style={styles.title}>Schedule</Text>  
      <Flatlist style={styles.flat_list}>  
        data={terms_and_classes}  
        renderItem = {onwTerm}</Flatlist>  
      </View>  
      <HamburgerMenu></HamburgerMenu>  
    </View>  
  )  
)
```

Figure 1.1: Code for an example screen

1.1.1 Outer Screen Container

1.1.2 Component Container

1.1.3 Hamburger Menu / Navbar (Container in Implemented Code)

#### IX.1.1.1. Login Screen

This screen will be shown to users who have not logged in. It prompts the user to enter their email and a strong password. If these fields are valid, the application will authenticate the user and display the appropriate landing page.



Figure 2.0

#### IX.1.1.2. Create Account Screen

This screen will be accessible through the login screen and is intended for users who do not have an account. They will be prompted to enter their name, email, address, phone number, and a strong password. Then, the user can log into the app through the Login Screen.

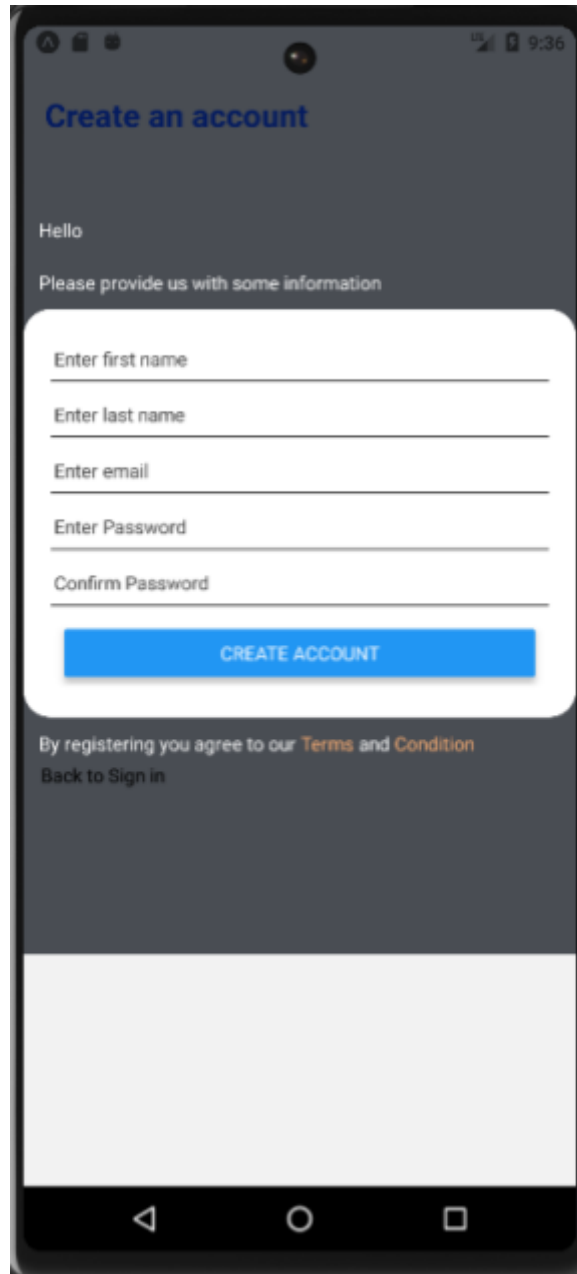
A mobile app prototype of the 'Create an account' screen. The screen has a dark gray background. At the top, the title 'Create an account' is displayed in blue. Below the title, the text 'Hello' and 'Please provide us with some information' are shown. A white rounded rectangle contains five input fields: 'Enter first name', 'Enter last name', 'Enter email', 'Enter Password', and 'Confirm Password'. Below these fields is a blue button labeled 'CREATE ACCOUNT'. At the bottom of the white rectangle, there is a line of text: 'By registering you agree to our Terms and Condition' with 'Terms and Condition' in orange, and a link 'Back to Sign in' below it. The bottom of the screen shows a white navigation bar with three icons: a back arrow, a circle, and a square. The status bar at the very top shows the time as 9:36 and some signal icons.

Figure 2.1

### IX.1.1.3. Forgot Password Screen

If the user has forgotten their password, they will be able to receive a recovery email through this screen. The user will provide their email address in order for this to occur.

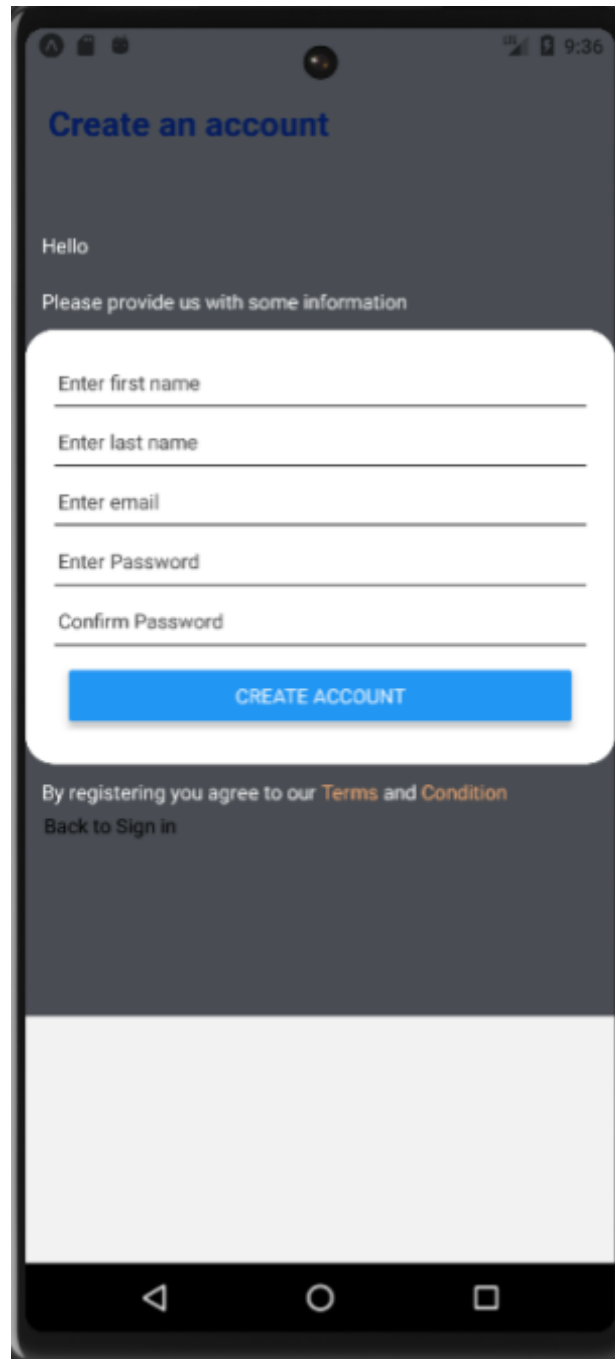
A mobile application screen titled "Create an account" in blue text. Below the title, it says "Hello" and "Please provide us with some information". The form contains five input fields: "Enter first name", "Enter last name", "Enter email", "Enter Password", and "Confirm Password". A blue button labeled "CREATE ACCOUNT" is positioned below the fields. At the bottom of the form, there is a line of text: "By registering you agree to our [Terms](#) and [Condition](#)" followed by a link "Back to Sign in". The screen is framed by a dark grey border, and the bottom shows a standard Android navigation bar with back, home, and recent apps icons. The status bar at the top shows the time as 9:36 and various system icons.

Figure 2.2

#### IX.1.1.4. Profile Screen

This screen will display the profile information of the user. It will show their profile picture, name, email, university, and major.



Figure 2.3



#### IX.1.1.5. Join University Screen

This screen is displayed to a new user. It displays a list of universities that the user will be able to select. After selection, the app will display the Select University Degree Program screen.



Figure 2.4

#### IX.1.1.6. Select University Degree Program Screen

This screen is displayed to users who have just selected a university. It allows the user to now select a degree offered by the University that they have selected. After selection, the user will see the Schedule Screen.



Figure 2.5

#### IX.1.1.7. Schedule Screen

In this screen, the user will be able to see the schedule for each term for their degree. They will be able to interact with the classes and move them around, and the schedule will automatically update.

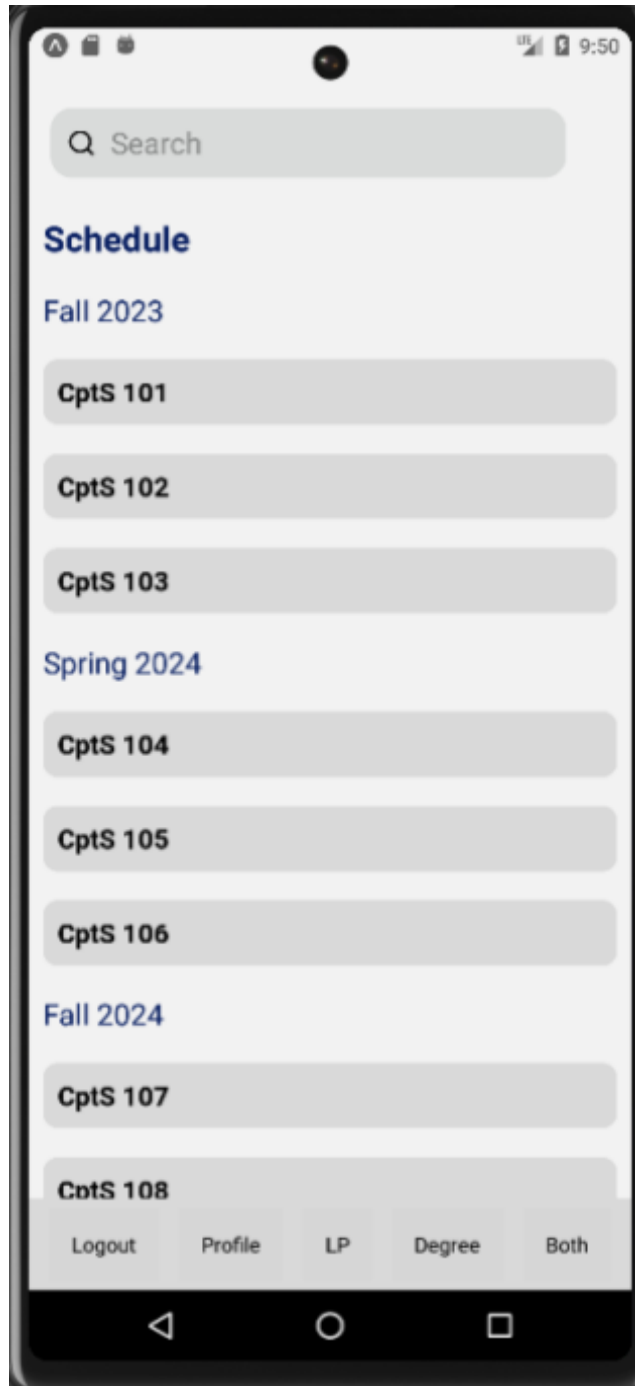
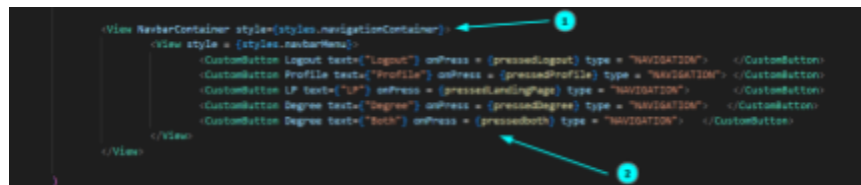


Figure 2.6

### IX.1.1.8. Navbar

The navigation bar was the main way that users will use to navigate the app. Currently, the navbar utilizes text to show off each button's functionality. Icon implementation will be done in the future. Figure 8.1 shows off the implementation of the navbar. It utilizes a flex view container with *custom buttons* as the components. The navbar utilizes the navigation component to switch screens with every button press. Figure 8.2 demonstrates what the navbar looks like.



```
<View NavbarContainer style={styles.navigationContainer}>
  <View style = {styles.navbarMenu}>
    <CustomButton Logout text=("Logout") onPress = (pressedLogout) type = "NAVIGATION" /> </CustomButton>
    <CustomButton Profile text=("Profile") onPress = (pressedProfile) type = "NAVIGATION" /> </CustomButton>
    <CustomButton LP text=("LP") onPress = (pressedLandingPage) type = "NAVIGATION" /> </CustomButton>
    <CustomButton Degree text=("Degree") onPress = (pressedDegree) type = "NAVIGATION" /> </CustomButton>
    <CustomButton Degree text=("Both") onPress = (pressedBoth) type = "NAVIGATION" /> </CustomButton>
  </View>
</View>
```

Figure 8.1: Navbar Code  
8.1.1 Flex Nav Container  
8.1.2 Buttons in Container



Figure 8.2

## X. Pathways Mobile Prototype Demonstration

Some key features are authentication, an interactable navigation bar, and populatable pressable Flatlists that are ready for backend content. At the moment the team has designed an early version of the login scheme where the user has to enter an actual email to log in. Our mentor gave us feedback on what service they want us to implement into the authentication. The team will also demo the navigation bar which allows users to travel between each screen easier. The team mentor had a concern that the navigation bar is not user-friendly since it would make sense for the team but people using it outside of the engineering perspective might not understand it. The last thing the team will demo to the mentor is how we used a stacked library to keep screens in memory where users can have an easier time loading between screens and navigating between them. The mentor liked the idea of stacked screens. The team has gathered many details from the mentors and plans to improve the application in the next sprint. First, the team plans to improve the navigation bar with icons to show users where this will take them. The team will allow the use of the Firebase to authenticate users. Another important feature the

team plans to improve is the screen where the displays of content will be consistent between mobile devices.

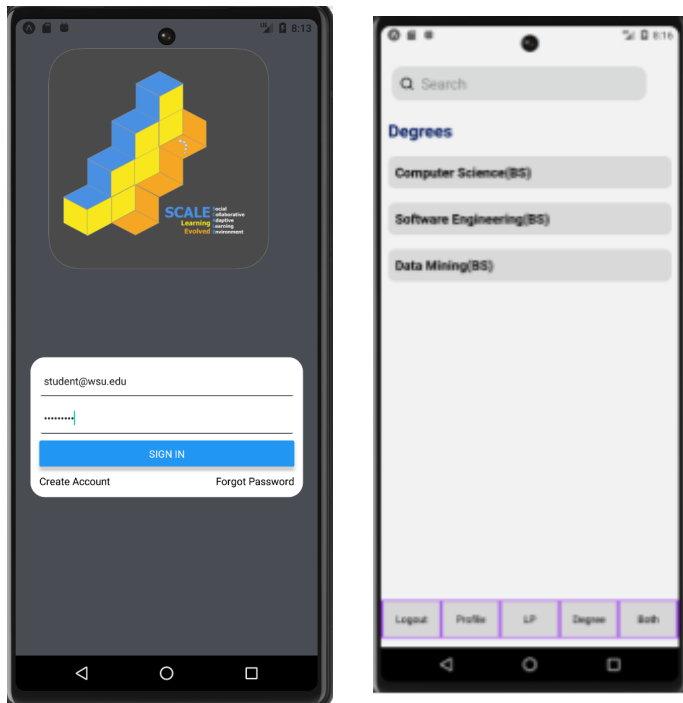


Figure 3.0

## XI.Future Work

One big feature the team will be working on for the second semester is the backend. The team will create a database to store user content and use them to display user information. Another thing the team will try is to implement their graph dependencies of classes onto the mobile version of the application.

## XII. Glossary

**AI:** Artificial Intelligence

**API:** Application Programming Interface

**BA:** Bachelor of Arts

**BDD:** Behavior Driven Development

**BS:** Bachelor of Science

**API:** Application Programming Interface

**CSV:** Comma Separated Values

**iOS:** iPhone Operating System

**JSX:** JavaScript XML

**ML:** Machine Learning

**QC:** Queens College

**Team LD:** Team Lockdown

**UCORE:** University Common Requirements

**UI:** User Interface

**XML:** Extensible Markup Language

### **XIII. References**

- [1] "Introduction · REACT NATIVE," *React Native RSS*, 06-Sep-2022. [Online]. Available: <https://reactnative.dev/docs/getting-started>. [Accessed: 05-Oct-2022].
- [2] R. Ranjan, "The Mobile App Architecture Guide for 2022," *Insights - Web and Mobile Development Services and Solutions*, 03-Oct-2022. [Online]. Available: <https://www.netsolutions.com/insights/mobile-app-architecture-guide/#cross-platform-application-architecture>. [Accessed: 06-Oct-2022].
- [3] *Introducing JSX*. React. (n.d.). Retrieved October 5, 2022, from <https://reactjs.org/docs/introducing-jsx.html>
- [4] "The Python SQL Toolkit and Object Relational Mapper," *SQLAlchemy*. [Online]. Available: <https://www.sqlalchemy.org/>. [Accessed: 05-Oct-2022].
- [5] "The py2neo handbook," *The Py2neo Handbook - py2neo 2021.1*. [Online]. Available: <https://py2neo.org/2021.1/>. [Accessed: 07-Oct-2022].
- [6] M. Hanson and F. Checked, "College dropout rate [2022]: By year + demographics," Education Data Initiative, 21-Jul-2022. [Online]. Available: <https://educationdata.org/college-dropout-rates#:~:text=College%20dropout%20rates%20indicate%20that,up%20to%2040%25%20drop%20out>. [Accessed: 21-Sep-2022].
- [7] M. Kuhfeld, J. Soland, K. Lewis, and E. Morton, "The pandemic has had devastating impacts on learning. what will it take to help students catch up?," *Brookings*, 03-Mar-2022. [Online]. Available: <https://www.brookings.edu/blog/brown-center-chalkboard/2022/03/03/the-pandemic-has-had-devastating-impacts-on-learning-what-will-it-take-to-help-students-catch-up/>. [Accessed: 30-Sep-2022].
- [8] N. Rodríguez-Planas, "Hitting where it hurts most: Covid-19 and low-income urban college students," *Economics of education review*, 28-Jan-2022. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8797148/#:~:text=Furthermore%2C%20the%20pandemic%20reduced%20freshman,expected%20household%20income%20of%2064%25>. [Accessed: 30-Sep-2022].
- [9] C. T. LLC, "Class timetable - schedule app," *App Store*, 30-Mar-2011. [Online]. Available: <https://apps.apple.com/us/app/class-timetable-schedule-app/id425121147?platform=iphone>. [Accessed: 30-Sep-2022].

- [10] A. Inc., "Schooly: School Planner," *App Store*, 25-Feb-2020. [Online]. Available: <https://apps.apple.com/us/app/schooly-school-planner/id1500232555>. [Accessed: 30-Sep-2022].
- [11] D. Kravcov, "Class schedule planner & tasks," *App Store*, 13-Oct-2017. [Online]. Available: <https://apps.apple.com/us/app/class-schedule-planner-tasks/id1278473923?platform=iphone>. [Accessed: 30-Sep-2022].

## XIV. Appendices

Figure 1

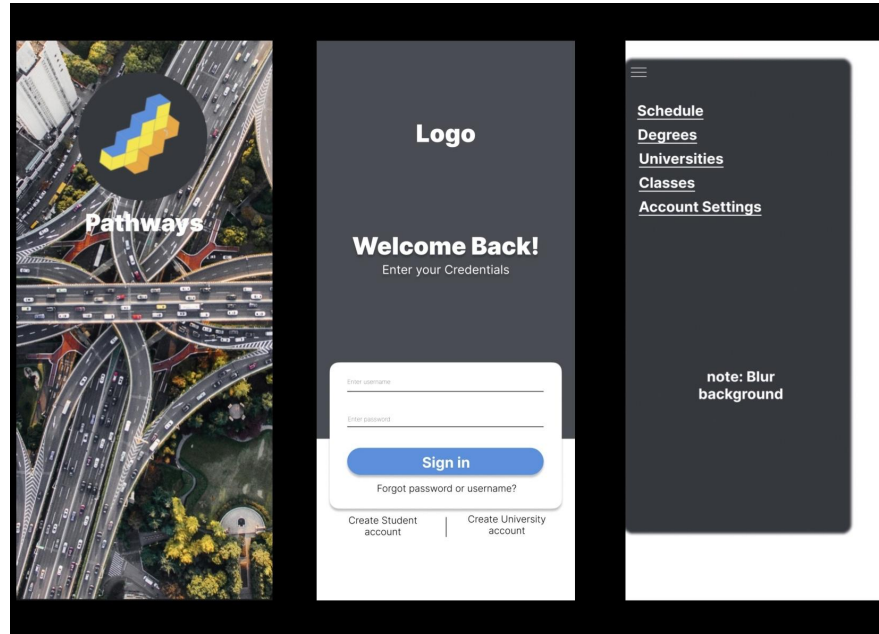


Figure 2

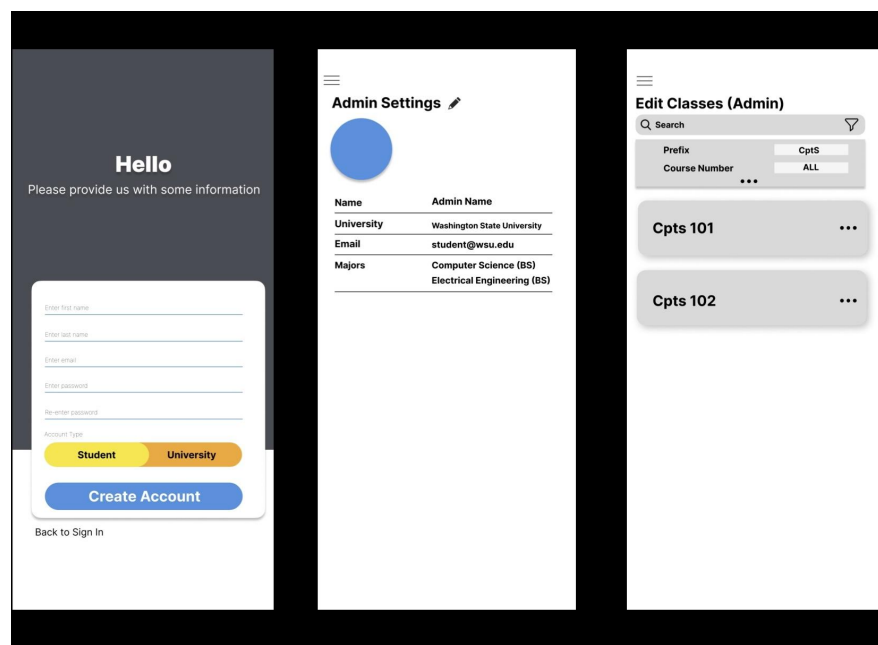




Figure 3

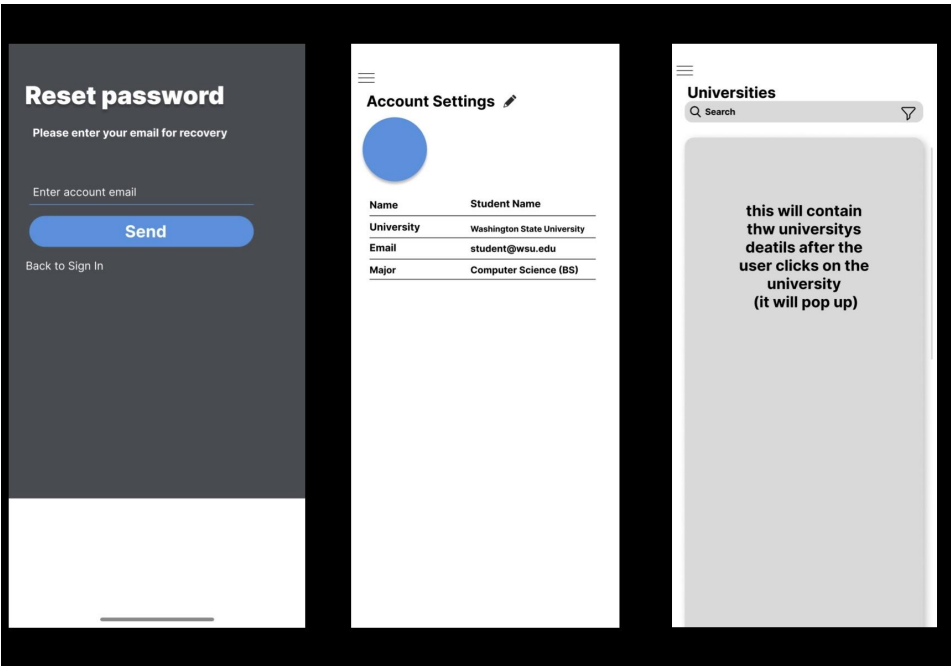


Figure 4

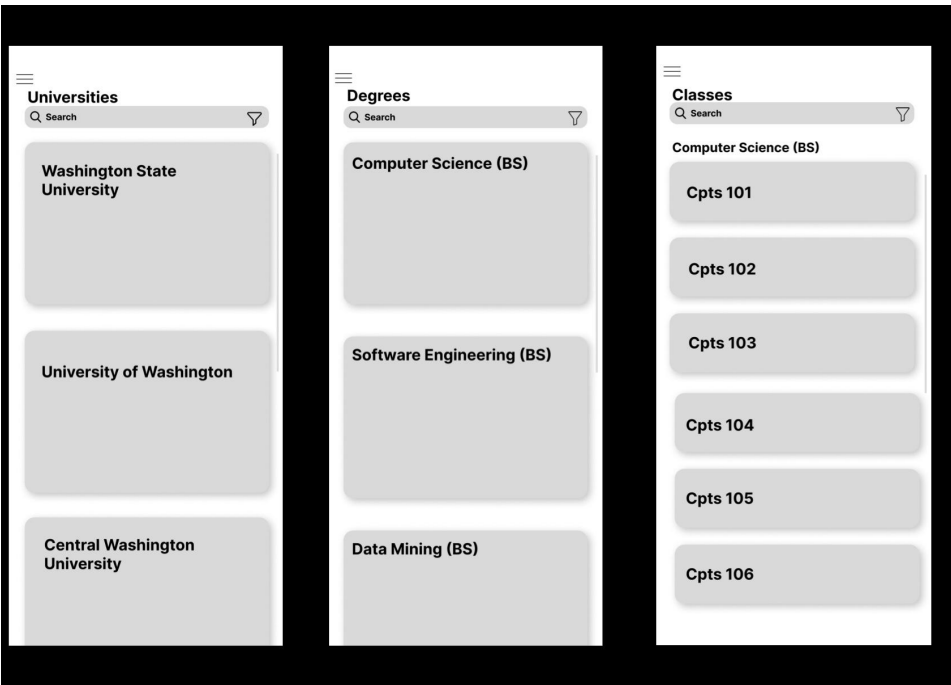


Figure 5



Figure 6

