Pathways Mobile

Project Solution Approach

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

October 4th, 2022

TABLE OF CONTENTS

I. System Introduction

This section will provide extensive details on the solution approach, Architecture Design, Subsystem Decomposition, Data Design, and User Interface Design. Finally, this document will provide information 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.

II. System Overview

SCALE *Pathways* mobile's functionality and design will be based on the previous iterations of a SCALE *Pathways* desktop application. Previous teams have provided us with a solid foundation regarding Frontend design and Backend systems. We will use a cross-platform mobile architecture, React Native [1], to achieve improved workflow and simultaneous development to both platforms throughout the year.

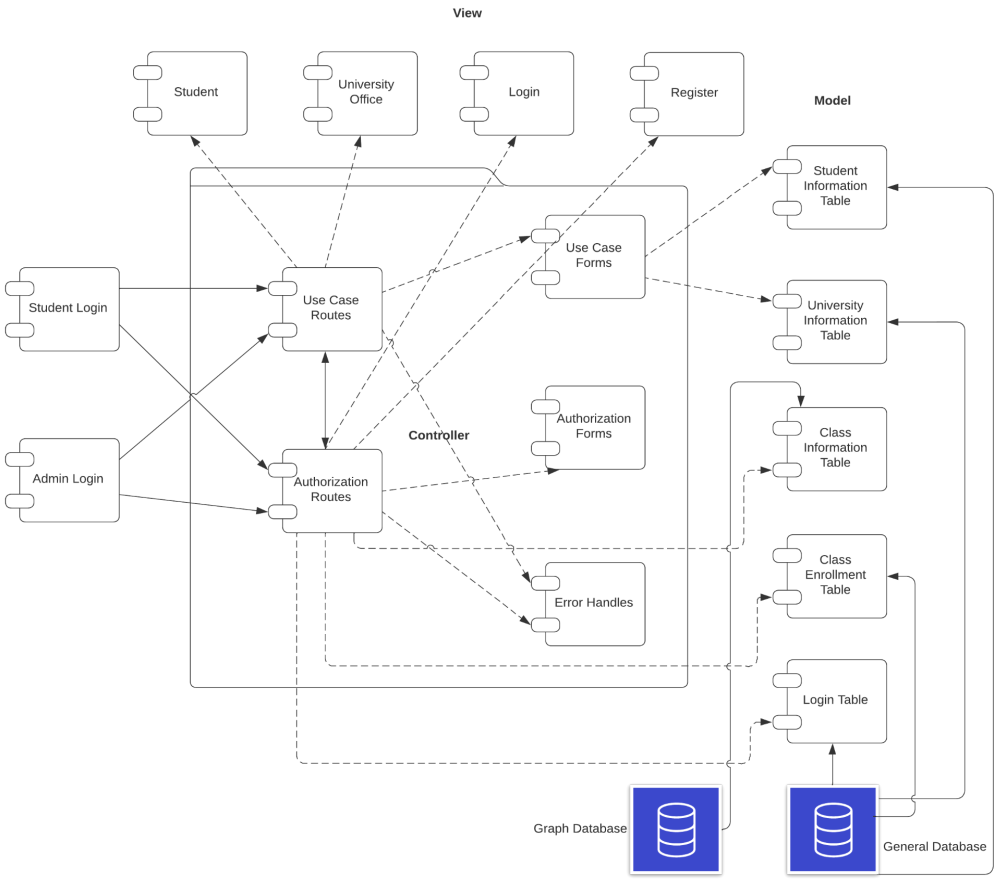
Team LD's project will primarily be focused on functionality for student users on iOS and Android platforms. Admin functionality will be a secondary goal once we have completed the student experience. Backend databases will remain relatively the same as the desktop application. Team LD's goal is to create a seamless iteration of the original *Pathways* application into a mobile format. SCALE *Pathways* Mobile will have all the features from the desktop app, including schedule generation, profile customization, graphing visualization, etc., in a mobile format. Team LD will be focused on the user experience first, making it much easier for students to get an auto-generated schedule as soon as all of their data is processed.

III. Architecture Design

III.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.



III.2. Subsystem Decomposition

I.1.1. 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.

I.1.2. 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 user interface and guarantee user's experience.

IV. 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:

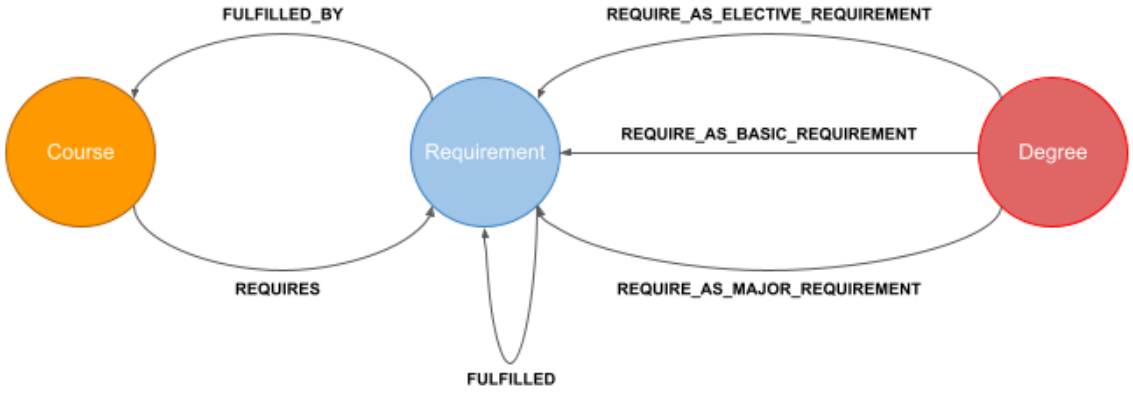
- Letter Grade Table
    - Represents a grade on a specific university's grading scale
        - Enables support multiple grading scales
    - References a university ID
    - Identified by a unique ID
- Course Record Table
    - Represents a record of a student's grade in a past course
    - references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

- Course
  - This will store the course ID, and be how we primarily retrieve class information
  - Attributes: like title, course number, and prefix
- Requirement
  - This represents a prerequisite requirement, degree requirement, or elective requirement relationships
  - Attributes: name
- Degree
  - This represents a specific degree offered by the university, for the purpose of associating courses as required or part of a template
  - Attributes: name, track

V. 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.

VI. Glossary

BA: Bachelor of Arts

BS: Bachelor of Science

API: Application Programming Interface

JSX: JavaScript XML

XML: Extensible Markup Language

UI: User Interface

VII. 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].

VIII. Appendices

Figure 1

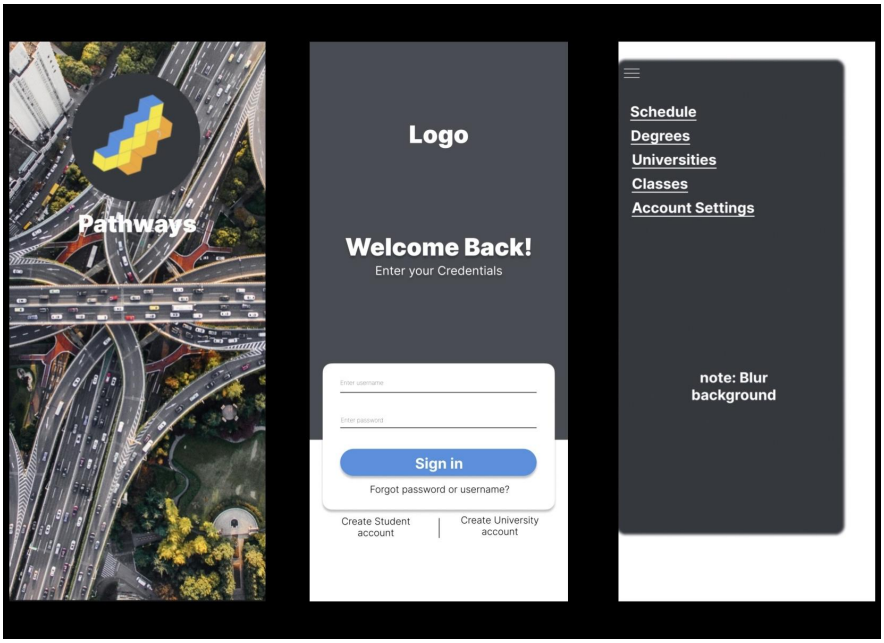


Figure 2

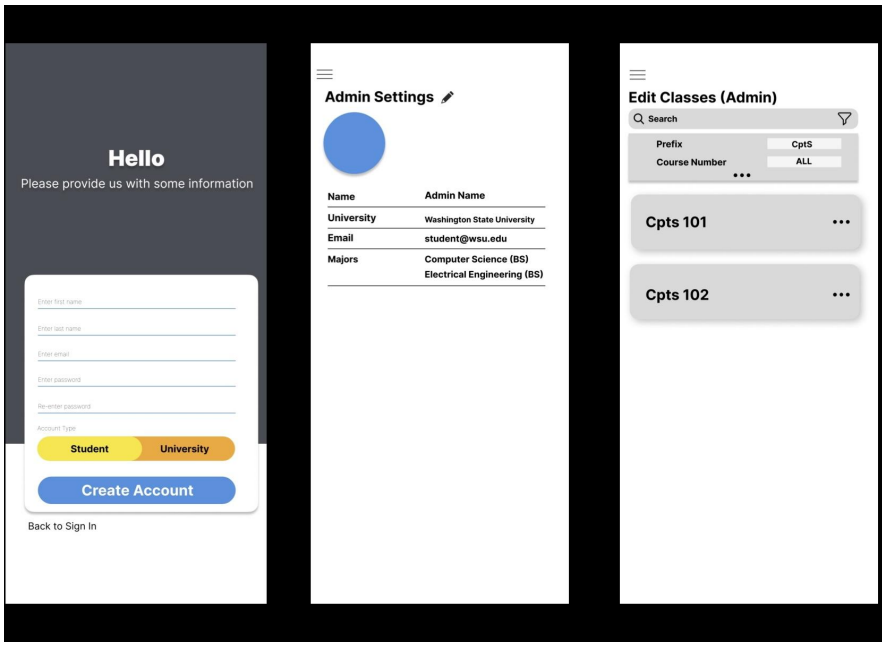


Figure 3

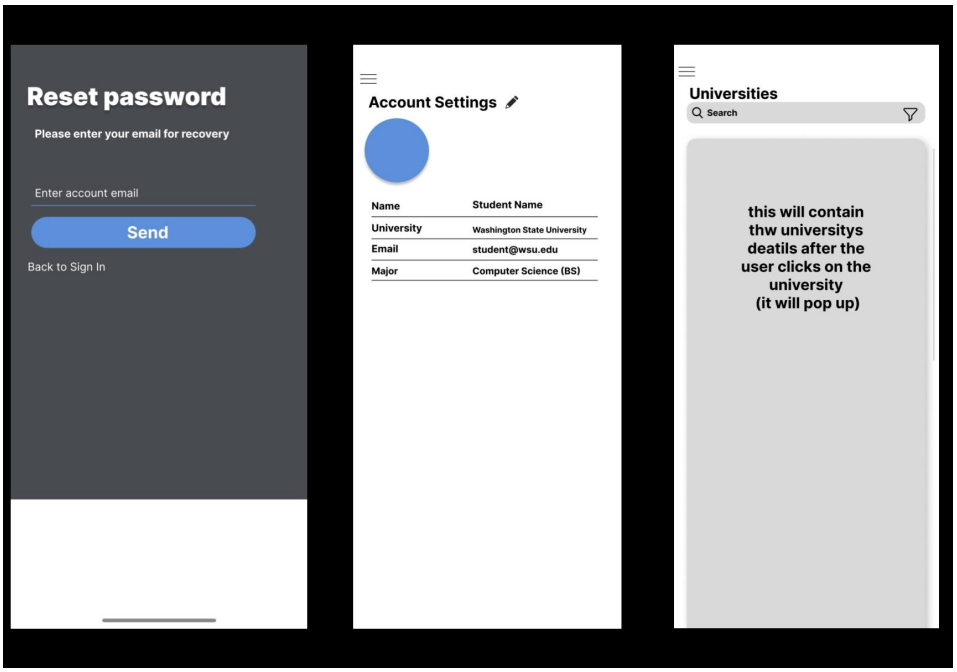


Figure 4

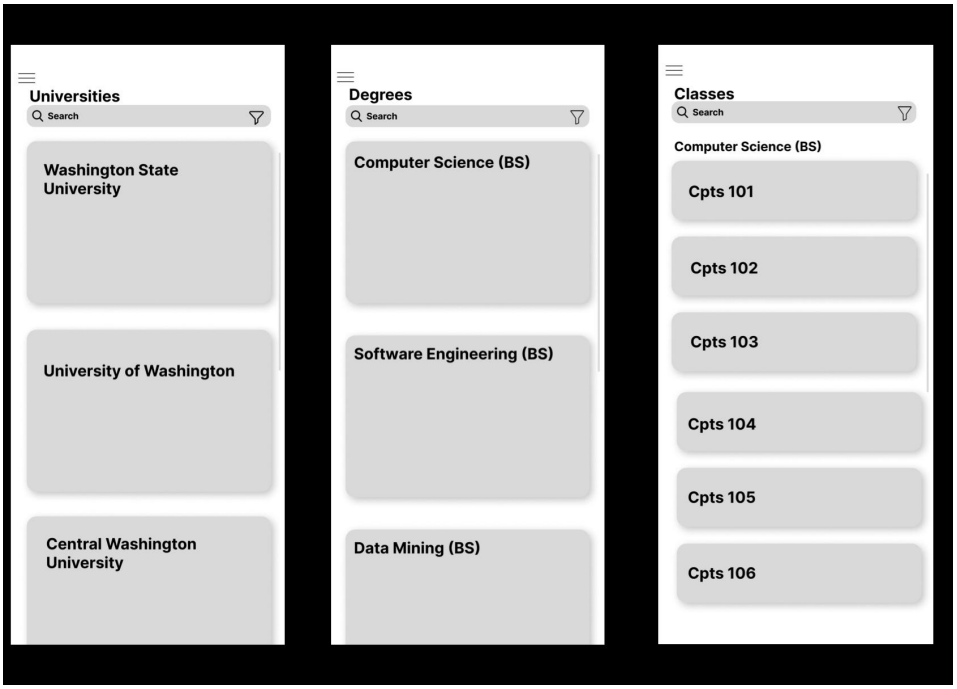


Figure 5

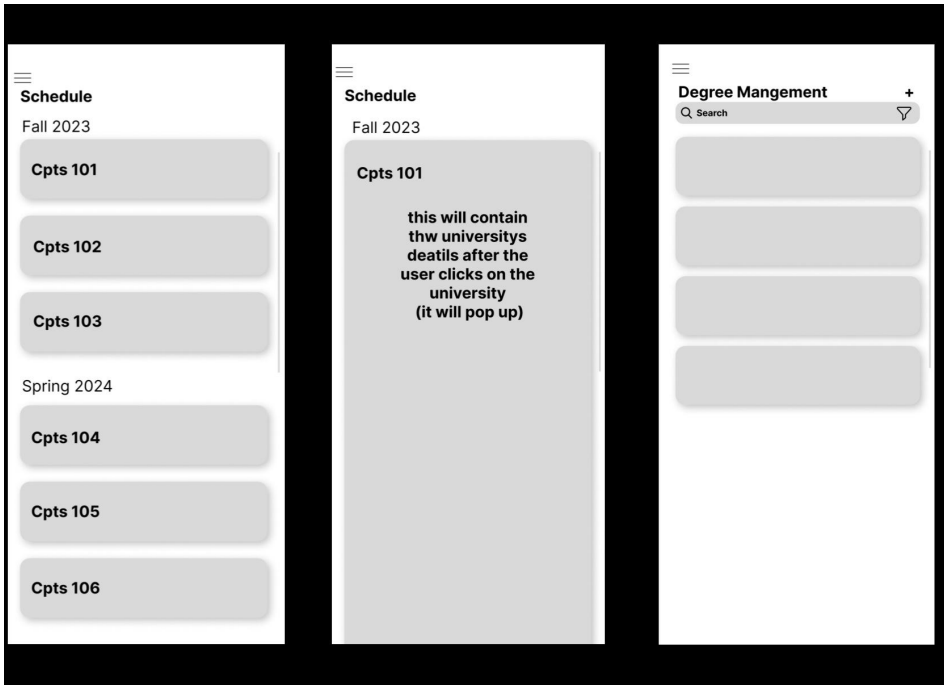


Figure 6