Independent Work Fall 2019 Final Report

Princeton Course Planner: A Course Planning Application for Princeton Students on Mobile

Peter Mwesigwa
Adviser: Bob Dondero

0. ABSTRACT

There are a number of course planning applications available to undergraduate Princeton students, but each of them has its strengths and weaknesses. This leads Princeton students to use a combination of different applications at the same time. I wanted to remove this redundancy and create one application with the most popular features that people are looking for when planning their courses. I also wanted to make our application for use on mobile devices – something that currently does not exist. I worked in conjunction with a fellow Princeton student, Vinay Ramesh '20, and together we were able to consolidate three key features into a mobile application for IOS: a course calendar for arranging your classes during the week, a flexible course search algorithm, and a subscription service that notifies you when there are spots available in a class you are interested in. We are about to start beta testing and our application will be available on the IOS App Store in the near future.

1. INTRODUCTION

Course selection is a critical time for the undergraduate student at Princeton. From my own experience along with that of a few of my friends, I found that there are many reasons why

students would select one course over another. Of course, the major reasons that shape a semester's course load pertains to the satisfaction of some academic requirement. It can be a general requirement, such as to fill out the total number of courses for your program, or fulfil a particular distribution area. Sometimes the reasons pertain to your department: it might be a prerequisite for you to take higher-level courses or necessary for a certain track within your department. Beyond this the reasons are different for each and every person. Some people choose classes merely because of location – they are looking for a class in the general vicinity of their other classes, saving them the hassle of hurried walks across campus. Other people are more interested in the time at which the class is held – there are sometimes gaps in one's schedule that they would like to be filled with something productive. Some classes are chosen because of the professor who is teaching them, or from the ratings and reviews from past semesters. Sometimes students are looking for relatively less hectic classes to fill out a course load; for this reason, they might want a class graded on a pass/fail basis. Furthermore, a class can be chosen as part of a new avenue of learning that a student is trying to explore. Course planning is therefore an active process for most students and can start weeks before the semester. A schedule is not usually finalized until the new semester, and even after that, students often add and drop classes during the semester.

To work on the application, I teamed up with Vinay Ramesh, a senior in the Computer Science department with whom I have worked together on a number of projects in the past. Our advisor was Professor Bob Dondero, a lecturer in the Princeton Computer Science Department. Our aim was to create Princeton Course Planner, a mobile application that would address the issues that we saw with the current course planning applications available to students.

Vinay and I felt that our application needed to have four critical features. Firstly, we want users to be able to research information about their different courses. For this we needed an interface where users can query courses in the Princeton database by name, department, course code, professor and a number of other parameters. Secondly, our app needed a weekly calendar where you can visualize the arrangement of your classes for different days during the week. This interface also allows you to create more than one version of a course load for different semesters in order to allow you explore multiple possibilities for your semester. It is also interactive, allowing you to add and remove classes from a schedule directly from the calendar, as well as providing information about the different courses that you would need. Thirdly, this application should allow you to receive notifications on different events, primarily when a class that you are interested in opens up. Last but not least, it should allow you to add these courses directly to TigerHub by transferring them from our application to your course queue.

2. EXISTING COURSE PLANNING INFRASTRUCTURE

2.1. TigerHub

The official platform provided by the University is TigerHub[1]. Established in 2014, TigerHub provides the medium through which students sign up for University courses. You are allowed to add classes to a queue for an upcoming semester and during a certain window during the semester you are allowed to enroll in the courses that are in your queue. TigerHub also has a somewhat useful calendar interface through which you can see your schedule as it would appear during the week. While all students use TigerHub to enroll in courses (after all, it is the only way to do so), because of its limited capabilities beyond that the vast majority of students do their

planning elsewhere and merely transfer their finished schedule into TigerHub as the final step before enrollment.

2.2.Other Applications:

For planning their courses, students use a number of other platforms. Most popular are the trio of Princeton Courses, ReCal and Pounce. Each of these applications is different from the other and has a unique set of features that attract students to use it.

2.2.1. ReCal

ReCal[2] is one of the leading applications that students use in course planning. It is a web application that allows you to add classes to a weekly calendar and is very similar to the course planner on TigerHub.

However, it is markedly different in a number of significant ways. First of all, whereas the TigerHub planner only has one course load schedule which you can customize, the ReCal calendar allows you to create multiple schedules – each with a different version of your course load for that semester. This allows you to experiment with different combinations of classes to see which one fits you better. Similarly, TigerHub does not allow you to include two classes that occur at the same time. ReCal on the other hand allows this, assuming that you will rectify this collision when the time comes for you to enroll in classes. This again makes the research process in ReCal a bit more flexible as you can add two potential classes that might clash with each other until you decide which one of the two you prefer.

Flexibility in ReCal is further enhanced by the way that you add and remove classes to your schedule. On the side of the calendar is a menu, at the top of which is a search bar. Entering a query in this search bar searches through the Princeton courses database for courses that match

that query, be it in department, course number or title, and returns a list of search results. Clicking on one of these results adds the course to the currently highlighted schedule. The work is not done yet though. Within a course, you have to select specific sections to attend, such as lecture and precept. On adding a course to your schedule, ReCal shows all the different meeting times for all its sections (lectures, precepts and the like). To select a specific section, such as a lecture, that section is clicked on to highlight it in bold. At the same time, all the other sections of the same type (lecture in this case) are removed from the schedule. To change your selection, click on the selected item to deselect it. All the sections that were previously removed should now reappear and a different choice can be made. This functionality is clean and efficient and allows you a lot of flexibility without unnecessarily cluttering the screen. Because of these factors, ReCal provides a "fast, sleek and intuitive user experience", according to their website.

2.2.2. Princeton Courses

Equally popular is Princeton Courses[3], a web application often used in conjunction with ReCal. While ReCal is excellent with arranging your course load during your week, it provides little information about the classes that you search before you select them. This is where Princeton Courses comes in. Like ReCal, you can search courses in the Princeton Database. Princeton Courses, however, has a wider range of queries you can search with. For example, you can search courses up via the professor who is teaching them – something you cannot do on ReCal. Clicking the filter button right at the side of the search bar opens up a panel on the left displaying a number of filters, most notably filtering by department, distribution area that the course falls under(such as QR or LA) and grading requirements of the course (whether it can be taken as a PDF/Audit course). Results can also be sorted by course code, relevance, title or rating, Perhaps the most popular feature on the website is the ratings provided for each course.

Each course has attached to it a rating that it received from the previous semester, as well as comments about the course from people who have taken it in the past. This is a popular feature about Princeton Courses as it allows students insight into what the course is like from the perspective of their fellow students rather than what can be gleaned from the description of the class from the course website.

2.2.3. Pounce

Pounce[4] is not as popular as Princeton Courses and ReCal but is still used by a fraction of the undergraduate population. It is a web application that allows you to subscribe to email and text notifications when you receive courses. You can search up courses by name or course code. Clicking on your desired course in the result brings up all the available sections, and sections that are full are highlighted in orange. Clicking on this allows you to subscribe to receive notifications when the course is open – in other words, to "pounce" on the course. Pounce also has a Twitter account (@PrincetonPounce) through which they alert people about openings in popular classes. Pounce is used by students especially when a course they are researching on ReCal or Princeton Courses is full.

2.3. Where we come in

Each of these applications that we have described has one or two primary features for which it is usually chosen. For this reason, no one application is usually used independently, and most students use TigerHub as well as some combination of applications from ReCal, Princeton Courses and Princeton Pounce. Furthermore, these applications are not inherently designed for mobile. ReCal and Pounce are web applications and are not designed to fit the relatively smaller screens of mobile devices.

Vinay and I decided to tackle these problems with a new course planning application. Firstly, our application is fully mobile, allowing you to do all the planning on your phone. Secondly, our application's functionality centralizes a lot of these features, minimizing the need to consult multiple screens when creating a course schedule for the semester.

3. IMPLEMENTATION

3.1.Initial Planning

Before work could start on the application, Vinay and I had some common planning to do together. The first step was to determine the tasks that would need to be carried out as part of the project. Vinay and I had two core features that we wanted for our application: a functional calendar which you could use to plan a course load, and a robust course search algorithm taking in a number of parameters and returning useful queries. For stretch goals, we planned to add a notification service when spots open up in courses and the ability to transfer a completed schedule in TigerHub. Professor Dondero, Vinay and I collectively agreed that the project would take on a "diamond " shape. Our application would be built on a server to service the mobile application. The application would then branch in the frontend into two main parts: the calendar which I would implement, and the course search side which Vinay would take care of. We determined to allocate additional features later in the semester as work progressed. Our work would convene at the top of the diamond with some common features for the frontend like infrastructure to add the user to the database and also to authenticate the user before they accessed the rest of the application.

At this point we formally designed the structure of our database, mapping out the tables in the database and the relationships between each table. Each semester (e.g. Fall 2018) would be

stored in the Term table. For each term, we had courses, such as COS 126 that were linked to that term and also to the department offering that course. These were stored in the Course table. Users could craft a potential course load for a specific term. Each of these course loads was stored in the Schedule table and had an association back to the term for which it is being planned as well as the user who owns it. Each course had specific sections such as lectures and precepts that would meet during the week which were stored in the Classes table. For example, a class could be P02 for COS 126, which meets Mondays and Wednesday from 3:00 PM to 4:20 PM. The calendar would then display each class as a series of events, with each event representing each instance the class would meet. Vinay and I decided to store these in the Meetings table. Users can manipulate a schedule by adding and removing courses to them, and also through selecting individual classes within schedules which they want to attend.

3.1.1. Obtaining the necessary APIs.

Before Vinay and I moved further with our project, we wanted to speak with the University administration. First of all, we wanted to inform them about our project and figure out a way to work with them to make the application successful in the long term. On top of that, we needed to get access to information pertaining to the courses available at the university.

We needed a number of APIs in particular. Firstly, we needed access to the list of courses that was actually offered for a semester with information about which department it belonged to, specific information such as the course code, title and description of the course. Similarly, we were interested in accessing the enrollment numbers for each course and how many free spots it had available. Another target was to get an API for obtaining reviews on courses. At the end of a semester, students rate various aspects of each class they have taken and write reviews for them. This information is collated by the University and we were hoping to get access to it. Lastly, we

needed an API that would allow us to send a queue of courses to TigerHub once the user is done planning their schedule in our application. We wanted to remove the redundancy where you plan out a course load in ReCal or Princeton Courses only to have to recreate the same schedule in TigerHub before enrollment. With this write access, you would only have to craft the schedule one time and transfer to TigerHub to find your courses ready in the queue for enrollment.

Vinay and I started off by speaking with higher-ups in the Computer Science department to get direction on who to ask for access to these resources. We managed to get in touch with Scott Karlin, who is a Senior Manager with Computer Facilities in the department. He gave us an endpoint that not only provided information for all courses in semesters for the past 3 years, but also had enrollment and capacity numbers for each of these classes. For help with the remaining two APIs he directed us to the Registrar's Office, the university body directly responsible for academic life at Princeton. We spoke with Jonathan LeBouef, the active Assistant Registrar with the Office of the Registrar of Princeton University, to inquire about the remaining two APIs. Unfortunately, we were unable to get access to the write API to TigerHub as it was a protected resource under University Regulations. Mr. LeBouef also was not aware of any API that provided reviews for courses.

These discussions had a number of implications. Firstly, the feature to transfer course schedules from our application to TigerHub was now off the table. Secondly, we would not be able to use course reviews and evaluations as heuristics in the course search algorithm. Nonetheless, Vinay and I thought that we would still able to continue work as these are not critical to the core operations of the application.

3.1.2. Creating and hosting the backend.

Vinay and I created a backend to store the information about our users and their schedules. This database also stores courses offered by the University that we retrieved using the API from the Registrar's Office. This backend is written in Python and utilizes Flask[5], a server-side framework written in Python, and a SQL database. We decided to host the backend for this database on Heroku and used PostgreSQL to create and maintain the database.

The file structure for the backend is as follows. The root directory contains the *src* folder which contains all the code that we wrote for the backend, as well as a number of other files required in order to host successfully on Heroku. Within the *src* folder there are a number of files. The *templates* folder contains a number of HTML templates used by the application when rendering on a browser. In *app.py,* the Flask app is initialized and configured using settings defined in *config.py*. The database is created from the app using SQLAlchemy[6], a tool for integrating Flask with SQL. We also used the Marshmallow[7] serialization library to convert rows from our database into JSON objects as responses to requests from the application. We also used Flask Mail[8] to create an object for sending emails to the user, for example when verifying their account at registration. The different tables used in the database and their relationships are defined in *database.py.* The API that the server provides to the frontend is put out in the file *routes.py,* in which routes which will be called by the app are mapped to functions that process the requests from the app. Finally, *__init__.py* brings everything together and contains a fully functional app, which is made available to Heroku. This file also runs the app during local testing on the *localhost* IP domain.

There are a few other files used for other tasks within the application. The module that handles CAS authentication is defined in *cas.py*. Here we are adapting code from Professor Dondero's *CASClient.py*[9] which was written to assist students in COS 333. We also wrote a

script in *fill_database.py* that updates the courses in our database with any changes from the Princeton database.

3.1.3. Initializing the Frontend

The frontend of the application is designed using React Native[10]. Designed by Facebook, React Native is a framework for making hybrid mobile applications. That is, it allows you to write code using defined React Classes in JavaScript as opposed to writing native code in IOS or Android. This has a twofold advantage. Firstly, a React Native project can be translated to both IOS and Android, therefore it allows for simultaneous development for the two frameworks. We settled on developing for IOS this semester but in using React Native, we left the door open on the possibility of an Android version in the future. Furthermore, writing code using React takes a shorter time than doing it in Swift/Objective C for IOS and Java/Kotlin for Android. This meant that the project of creating an entire mobile application could now conceivably fit in a semester-long independent work project. Furthermore, Vinay and I have worked extensively with this framework in the past, thus it was a natural choice for us to make when we conceived the idea for this project. The entire React project for this application is contained in the *PrincetonCoursePlanner* folder. Within this folder, all the code that we wrote is in the *App* folder, with the rest being starter code and files provided by React native upon the creation of the project. Within the App folder there are a number of folders. The *redux* folder contains all the code that handles the Redux side of the application. Redux[11] is a library that allows you to store one app state that is accessible from anywhere within the application, thus allowing an app to scale without the logistics of maintaining the same version of information across different components. The *routes* folder contains a file that contains the main navigation tree for the app's screens. The templates folder contains a number of useful templates used to render some

elements that appear within the application. The *utils* folder contains a file with useful constants such as the main URL through which to make requests. The *loginscreens* folder contains the code for the login and registration screens. The *coursesscreens* folder contains the parts for the course search aspect of the application that was worked on by Vinay. The *calendarscreens* folder contains all the code for the course calendar which I implemented

3.2.Implementing the Calendar side of the Application

3.2.1. Planning the calendar interface

With the calendar, I wanted a user to be able to view and edit their course schedules. To manipulate their schedules I planned to design a menu that pops out from the side of the screen allowing the user to change the currently visible schedule. To do this, the user will have to change to the term that contains that schedule and then select that schedule from amongst other schedules in the term. This I thought was a logical and intuitive way of arranging the user's schedules. I also wanted the courses for the current schedule to be shown within the same menu. Each course and its classes would have a specific color, thus differentiating it from other courses in the schedule. Users should be able to change their preferred sections to attend within a course or, if they want to, remove a course entirely and add an alternative.

3.2.2. File Structure:

There are five files in the folder. The code that renders the calendar as seen in the app is in *calendarscreen.js.* In doing so, it calls functions from the file *helper.js* that display the horizontal and vertical lines that partition the calendar and the boxes that represent each class. Important constants for the calendar side are maintained in *constants.js*, such as the height of the calendar in pixels and what time the calendar would start and stop showing for. This allows me to be

modular with the function. In *packer.js* there is a function that allows for classes showing at the same time to display side by side. This allows the user to add classes that clash with each other to the same schedule, thus more flexibility in the process. Each meeting time displayed on the calendar is rendered as a rectangular box using the class in *eventbox.js* where the top of the box is at the start time and the bottom is at the end time of the meeting.

3.2.3. Implementation process and challenges

The first step was designing the appearance of the calendar. I partitioned the screen into five columns for each day of the week and added horizontal lines to mark the hours of the day. The calendar displays only Monday to Friday and times start from 8am to midnight as this is when classes are held during the week. I also wrote a React Native component class that rendered each class in the schedule in the calendar, putting it in the appropriate column for its day and at the appropriate time during that day.

It was at this point that I encountered one of the most challenging parts of my work on the calendar. Because users were allowed to add classes that clashed with each other, I had to find a way of displaying both of these classes at the same time. I decided that the best way would be to have any clashing events rendering side by side within a day column with each event taking up its full height according to the times at which it started and ended, but allowing any other events that clashed with it to display next to it without intersecting. I tried a number of algorithms without success but eventually achieved this by remodeling an algorithm to address a similar issue in a public GitHub repository called *react-native-events-calendar*[12]. Here, the owner had implemented a React Native library for creating a calendar to which you could add daily events. Coming up with this solution initially set me back, but I was able to get back on track.

To link the calendar in the mobile application to the server, I created a route that returns all the schedules that a user had in the database. For the application, these schedules had to be grouped according to the semester they were in. More so, classes had to be grouped by course and section. To do this I wrote a function that pre-processes the schedules before they were returned to group them by semester. It also grouped the classes within the schedule by course and precept and returned a tree-like object that moved from terms to schedules to courses and then classes. This top-down hierarchy is critical to how the calendar operates in manipulating classes within schedules. This object is stored as raw data in the frontend and referenced every time the user makes a selection.

The next step was to implement the selection of specific classes within a course by a user. To do this I adapted a similar model as is used in ReCal. When a course is added for the first time to one's schedule. I add all the classes that correspond to that schedule. Thereafter, to select a class, the user taps on it and that class is highlighted while all the other classes in the course of the same type disappear. Tapping the selected class brings back the classes that were taken away, allowing the user to make another selection. After some planning, I decided that the best way would be to remove all the classes that the user didn't want from the schedule when they made a selection and put them back when the user changed their mind. To do this I wrote two routes on the backend which would each take a schedule and a class: one of which would remove all the classes of the same type and course as the submitted class from a schedule and another that would put all the classes in the database of the same type and courses back in.

I also wrote a number of other functions on the backend pertaining to the calendar. I implemented functionality for changing the name of a schedule or deleting a schedule. I also

added code that logs the user out of the application as well as code that deletes the user's account from the database.

3.3.A side note: The Challenge of implementing CAS

Our application is limited to Princeton Students. As such our authentication process involves ensuring that the user is a member of the Princeton community. Vinay and I therefore had to integrate CAS authentication with the university into our application, which is a protocol through which the user authenticates themselves as a member of the university before accessing any privileged information. Our initial plan was to put the authentication as part of every single request that was made to the backend. We planned it such that if the user is not authenticated, a modal would pop up and prompt them to enter their credentials. However, we soon realized that this was not feasible. This is because, unlike on web where the browser stores a session cookie, a different cookie – so to speak – would be created with every single request that we sent for mobile, Therefore, for one request could not carry on for subsequent requests.

We came up with kind of a workaround, where we used a basic web app for authentication. This authentication happens once when a user is registering for the first time. Before they can create their account, they are directed to a web application that authenticates them with CAS. Once successful, the user can return to our application and log in with their new credentials. We think that while this is acceptable, it would not be an ideal solution in all situations. Something for future improvements would be to be able to integrate CAS authentication for results made from non-browser applications. Such an innovation would be useful not only to us, but to any similar applications trying to use CAS authentication.

4. USE CASES

4.1.1. Installing the application on IOS

On IOS the app will be available on the App Store in the near future. To download it off the App Store, merely search for the application called “Princeton Course Planner” and the application will be in the search results. Clicking on the result will take you to the application’s page on the App Store, which will have a button called “Get” near the top. Click this to start the download of the application. You will be prompted to verify that you want this download to proceed, and might have to enter your Apple ID and password or perform verification using Face ID or Touch ID. Once the download is complete, you can open the application using the “Open” button or exit the App Store and find the application amongst your phone apps.

The application is currently in the beta testing phase and is therefore not available on the App Store yet. Chances are, therefore, that you will be downloading the application using TestFlight. To start this process, download the TestFlight app off the App Store. Secondly, you will need an invitation in order to download the app beta. This invitation is sent from Apple via email. If you do not have an invitation, email me at mwesigwa@princeton.edu so that we can rectify the situation. In this invitation there should be a link. Clicking on this link will take you to TestFlight and if you follow the prompts, you should have the app downloaded on your phone. Click the “Open” button near the top of the screen and TestFlight will install the beta on your device.

4.1.2. Signing up as a user for the application

Once the app has been opened, the user is met with the login screen. As this is the user’s first time with the application, they will have to sign up. On pressing the signup button, they are prompted to a screen that requests their NetID. It will then send an email to their Princeton email, with a link to sign up. When they open this email and access this link, they are taken to a

webpage where they set their new password. If all goes well, they receive an alert that they have been successfully registered. After the user is signed up, they can now navigate to the Login page. To access their account, they simply input their Princeton NetID and their new password in order to verify their information.

4.1.3. Adding courses to a schedule.

Let's say that you are logged into the application and you don't have any schedules. The current semester should be "Spring 2020" and the schedule name at the top of the application is "None". Let's say you want to add COS 217 to your spring schedule. In order to do this, click on the "Courses" tab on the application in order to be taken to the search side of the application. Here type in the query "COS" and select the term "Spring 2020". The application will return all the courses for that term that match the current search query. You can also add a number of filters to your search to reduce the number of results: Click on the menu button on the left of the search bar to display the filters, For the departments filter select "COS" and for the course levels filter "2xx". This should show you COS classes at the 200 level. Clicking on the result for COS 217 takes you to a screen showing the details of the course. You'll see the course name, title, description and professors teaching the course for that semester. You can add the course to your schedule by clicking the Add to Schedule button at the top of the screen. This will open up a menu. Click on "Create a new schedule!" When prompted for the new schedule name enter "Schedule 1". A message should then show up alerting you whether the course has been successful or not. Clicking on the calendar tab on the tab navigator should take you back to the calendar where "Schedule 1" is displayed and COS 217 has been added.

4.1.4. Adding a new schedule from the calendar side

It is also possible to create a new schedule directly from the calendar itself. Clicking on the options menu button at the top left of the calendar screen opens the side menu for the calendar. Check that the selected term is Spring 2020, otherwise click on “Change Term” to change the active term. Below the name of the current schedule click “Change Schedule” and a list of all the available schedules should show up. At the bottom of the list is a button with a + sign titled “New Schedule”. When prompted to enter a schedule name enter “Schedule 2”. You should now have two schedules for Sprint 2020

4.1.5. Adding and removing sections and courses from the schedule.

Open the side menu and select “Spring 2020” as a term and “Schedule 1” as your schedule. You should have once course visible in the calendar: COS 217. There is only one lecture available, and everyone attends this by default. However, there are a number of precepts that you can choose from. Clicking on one of the precepts selects that precept. It removes all the other available sections front the calendar and highlights that section by showing it with a dark bar above the name of the course. To cancel your selection, tap the section and the removed sections should reappear, with all of them faded in the background. Let’s now add another course to the application, say PHY 104. Open the side menu and click on “Add course” right above the label for COS 217. In the course search side look up “PHY 104” making sure that the selected term is Sprint 2020. Once the search results have loaded, select the appropriate result. On the next screen, select “Add to Schedule” and select “Schedule 1” as the schedule to be added to. Once this is successful, click on the “Calendar” button on the tab at the bottom of the screen. You should be taken back “Schedule 1” in the calendar, which now has COS 217 and PHY 104 in it. All the classes for PHY 104 are displayed in a different color than from COS 217. For PHY 104,

you have to select a lecture, a precept and a lab. After doing that, say you now want to remove COS 217 from your schedule. Opening the options menu, below the name of the current semester and schedule, there is a list of courses that are in this current schedule. Each course in the schedule has the same color in this list as in the calendar. Locating the label for COS 217 in the list, you see that there is an "X" next to it. Clicking on that X allows you to remove the course.

5. TESTING AND EVALUATION

Even before Vinay and I started coding, we tested the idea amongst people on our circles to determine how the application would be received by people. Through this kind of informal interview process, we received a lot of feedback about our application. For example, we found that a lot of people were interested in filtering their classes by the classes' start and end times. We therefore added this feature to our application.

Furthermore, we were both was able to build a version of the application on an IOS simulator as I coded the application. We could also build a version of the application by installing it from our computer to a physical iPhone or iPad using a cable. This was useful for testing the visual appearance of screens in the application and ensuring that they would render as described. I especially used it for testing the calendar and making sure that all the different elements of the screen were displayed in the right place. The IOS simulator also allowed me to simulate the screens of different iPhones such as the iPhone 8, iPhone 8+, iPhone 11, iPhone 11 Pro and iPhone 11 Pro Max. This allowed me to adjust the dimensions of the components of the screen to ensure a uniform experience across all these devices. I also used Redux frequently during debugging. Since our application state was managed by Redux, we could observe its contents at

various points of time using Redux Logger[13], a tool that prints the contents of the state to the console every time that they change. This allowed me to test the different features of the calendar and ensure that everything changed as expected.

In preparation for the beta testing Vinay and I needed a way to get the application installed on a large number of devices through a relatively easy install process. For this we turned to TestFlight. TestFlight is an application designed by Apple to facilitate the safe and easy installation of beta applications on one's device. A tester is sent an invitation link via email, through which they are directed on how to install the application. To prepare our application for upload on TestFlight, we created an account on App Store Connect. Doing this would enable us to send the application directly to people's phones and would also facilitate pushing the application to the app store in the future.

In the meantime, I was able to demonstrate prototypes of the application to Professor Dondero during our meetings with him, as well as a few of our friends and acquaintances on campus. I received feedback on a number of features on the application. Vinay and I haven't been able to do full-stage beta testing yet as we have only just received authorization from the app store. We plan to do this over the next few weeks, especially in the period leading up to course registration at the start of the spring semester. This will allow us to fix any bugs that we might have missed during development so far, as well as adjust the application's features according to user feedback.

6. CONCLUSION

All in all, the application was a success. Vinay and I implemented all the main features that we were aiming for. While we were unable to get some of the additional features such as course

reviews and being able to write to TigerHub, we believe that we still got a more than satisfactory outcome in the end. We are currently about to start beta testing for our application, and we will continue to make bug fixes and aesthetic changes as we continue to get feedback from our users. Once we feel that both us and our users are satisfied with the end product, we plan to release the application on the App store in the near future.

7. ACKNOWLEDGEMENTS

I would like to thank Vinay Ramesh for working with me on this application, as well as Professor Bob Dondero for advising our project and providing user feedback on our application. Furthermore, I am grateful to Scott Karlin with the Computer Science department, Dean Jill Dolan and Jonathan LeBouef with Office of the Registrar of Princeton University for meeting with us and providing invaluable supporting the project. Lastly, I would like to thank the Computer Science Department for funding this project.

8. SOURCE CODE

The source code for this application is on GitHub. The React Native project for the frontend is housed at is at https://github.com/vr2amesh/princeton-mobile-frontend and the Flask project for the backend at https://github.com/vr2amesh/princeton-mobile-backend.

9. HONOR CODE

This paper represents my own work in accordance with university regulations.

/s/ Peter Mwesigwa 01/06/2020

10. REFERENCES:

[1] "TigerHub." Princeton University. [Online]. Available: https://registrar.princeton.edu/tigerhub

[2] "ReCal." Princeton USG TigerApps. [Online]. Available: https://recal.io/landing

[3] "Princeton Courses." Princeton USG TigerApps. [Online]. Available: https://www.princetoncourses.com/

[4] "Pounce." Princeton USG TigerApps. [Online]. Available: https://pounce.tigerapps.org/

[5] "Flask." The Pallets Projects. [Online]. Available: https://www.palletsprojects.com/p/flask/

[6] "Flask-SQLAlchemy" The Pallets Projects. BSD-3 License. [Online]. Available: https://github.com/pallets/flask-sqlalchemy

[7] "MarshMallow." MIT License. [Online]. Available: https://github.com/pallets/flask-sqlalchemy

[8] "Flask Mail" Matt Wright. BSD-3 License. [Online]. Available: https://github.com/mattupstate/flask-mail

[9] "PennyFlaskCAS". Robert Dondero, Jr. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall19/cos333/lectures/16websecurity/PennyFlaskCas.tar

[10] "React Native." Facebook. [Online]. Available: https://facebook.github.io/react-native/

[11] "Redux." Dan Abramov. [Online]. MIT License. Available: https://github.com/reduxjs/redux

[12] React Native Events Calendar. Duy Lượng. ISC License. [Online]. Available: https://github.com/duyluonglc/react-native-events-calendar

[13] Redux Logger. Eugene Rodionov, MIT License [Online]. Available: https://github.com/LogRocket/redux-logger