CS97 Final Project Report

You've just committed to a university; you're next four years stuck at some college campus. While people may find it as a great stress reliever to graduate high school and attend college, the reality is that the real pressure has just begun. At a prestigious college such as the University of California, Los-Angeles, not only are students competing and striving for good grades, they have to worry about clubs, and more importantly, the classes they need to take to graduate. As a student myself, I understand that coursework itself is taxing enough; I don't want to have to constantly meet up with counselors, maintain an updated Google doc, and surf through the UCLA coursework website to formulate a four-year plan that will inevitably change throughout my years at college. All this extra work writing down a four-year plan that will have to be altered if certain classes are unavailable is just not worth it. As a computer science student, I decided to create a web application that will alleviate some of the problems stated above.

Our website, BruinPlanner is an interactive application for UCLA students that are in the college of engineering, to help design a 4-year plan based on the available and necessary classes. For too long, the HSEAS website has struggled with presenting information clearly, because of out of date major requirements, old "prospective" 4-year planners, and files all over the place. Instead, if students had an interactive 4-year planner, where they could select their major and visually see what the required classes and their respective prereqs were, this would save a lot of unnecessary time and pain. Our website allows students to create an account that would give them access to a 4-year planner based on a major in which they could organize their classes based on requirements.

BruinPlanner was created using the MERN stack. To specify, my group used MongoDB as our database, NodeJS and ExpressJS for our server, and ReactJS for our client. Outside the standard tech stack, our website utilizes many open-source npm packages/libraries such as passport/passport-local, bcrypt, and react-dnd. Using both passport/passport-local and bcrypt allows the backend to encrypt passwords and store them locally in our MongoDB database. On the other hand, react-dnd is a front-end node package that allows for easy drag and drop of HTML elements. In addition to our project's regular dependencies, we are also utilizing nodemon as a development tool to make updating the backend quicker and easier. In regards to our project's architecture, we have our backend contents located in the top-level directory named BruinPlanner. Inside this folder, there are contents such as app.js (initializes our backend), node_modules directory, package.json file (external packages strictly for backend), .git directory (version control), and a directory called client. The client folder essentially contains our frontend react project. Similar to the top-level directory, it possesses its own set of external imports from package.json and node_modules but for the client-side. In our project, we perform almost all routing on the front-end; however, we use Express's routing to regulate special cases where the users are logging in or have already logged in.

In our application, we provide a variety of features such as login-authentication, autocomplete of UCLA courses based on substring provided by the user, and an interactive drag-and-drop UI. Students that access our website can create a personal account that will store

and save any 4 year plans they create. They can modify their plans using the autocomplete form to generate a list of courses displayed with colorful blocks. To add the desired course to their plan, they can simply drag and drop it into the appropriate quarter specified by the four-year plan grid. In cases where the student wants to remove a course from his/her plan, they can drag the course from the grid to the trash located at the bottom left corner of the screen. When a plan is complete, users can save it or take a screenshot using the top right corner buttons. Finally, an underlying but notable feature is our web application's ability to filter out courses from the autocomplete list. In simpler terms, a course may only appear either in our planner or in our generated list of courses (never both at the same time). Whenever a course is added to our planner, it is removed from the list of general courses from the autocomplete. When it is removed from the planner, it is appended back to the general courses. Upon planner load in, the server-side runs an algorithm to remove courses contained in the loaded planner from our master course list before sending that data to the client-side.

Throughout the creation of this website, I learned a surplus amount of new things. This is primarily because I did about 85-90% of my group's website. To sum up my contributions, I wrote the whole server-side and 80% of the client-side for this project.

The main task on the server-side included

- Configuring Express and Mongoose (for MongoDB)
- Writing all GET, POST, and DELETE request routes
    - Creating a new planner
    - Modifying a planner
    - Getting account information (username, planners, etc)
    - Receiving data from the form on the contact page
- Initializing passport-authentication for logging in
- Pulling the UCLA DevX 2016-2020 course list (http://api.ucladevx.com/), filtering it for unique classes, and writing those classes into a new JSON file named master_courseList.json

On the other hand, my task on the client-side included

- Creating the front-end routes
- Designing/Developing the home page (Home.js, Home.css)
- Designing/Developing the log-in pages (Login.js, Login.css)
- Designing/Developing the sign-up page (Signup.js, Signup.css)
- Designing/Developing the account dashboard page (Account.js, Account.css)
- Designing/Developing the planner modification page (Planner.js, Planner.css)

From my contributions, you will notice that I created the foundation of my group's web application and a majority of its features. As of the time this report is being written, my group members were responsible for filling up the text on the homepage, creating the about and contact page, adding a screenshotting button for the planner's page, and writing the README.md. While

this may sound like a total bash on my group members, it is not. I am not annoyed or complaining about the lack of contribution from my group members. Upon several times my members and I video-chatted, I noticed they were hardworking students and just had trouble starting up the project due to inexperience in web development. I assisted them to the best of my ability by explaining the syntax of the code, the purpose of each file, and how to clone/setup the project. Given that we are virtual, this task proved to be a lot more difficult than expected, but at the end of the day, we were able to create a functional website that I am proud of. Once again, this is simply a response to my contributions to the project.

As mentioned above, the group's main challenge was trying to set up and start coding the website. In regards to my struggles, learning the syntax and functionality of different libraries and how to optimize the speed of sending a list of all 2016-2020 unique classes to the front end were the biggest challenges. Before starting this project, I was a pure front-end developer. I had never touched the backend and had no idea how to connect the React frontend to a NodeJS and ExpressJS backend. After countless StackOverflow searches and video tutorials, I figured out the general idea of transferring information from client to server (vice-versa). However, throughout my implementation, I ran into many problems with async, await, and promises. Despite watching many video tutorials explaining how they work,
I was still very confused about the execution order with these statements. My temporary solution to this problem was to avoid using async functions and when a line executes after the lines following it, I would wrap that line in an async function and call it with await. Moving on to my second biggest struggle, I wanted a fast way to send the list of UCLA courses from the backend to the frontend. My initial approach was to fetch course info from the UCLA DevX Courses API, find all unique course IDs and store the IDs along with names in an array of objects, filter that array by removing courses that are already present in a planner, and then send that array to the frontend. This approach was undoubtedly very tedious and unnecessary. As I coded other components of the website, I realized that I could instead store the unique classes in a file called master_courseList.json to avoid having to fetch and sort from the DevX API. I created a data mining helper function that performed the normal fetch, sort for unique class IDs, and then used the fs package to write my unique array into a .json file. Doing this, I was able to significantly reduce load time and avoid directly depending on the DevX API for my application to run. In the end, my optimal solution to this problem was to create an array variable that would take the value of master_courseList.json, filter it for classes existing in the planner, and then send it to the frontend.

Throughout the creation of this project, I tried my best to apply the idea of scalability and modularity in case of future additions to this project. For example, I have separated parts of the backend code into files where I can further expand the functionality. Having said this, I am very happy with the way I tacked this project. I didn't just rush to code a finished project; instead, I considered potential advantages and disadvantages with the project architecture, algorithm methods, and more. The primary reason I considered these aspects of BruinPlanner is because I have many plans for it in the future! I would like for our website to preload 4-year plans of a

certain major upon creating them, provide a checklist for students to indicate whether they have created a planner that fulfills all requirements to graduate, and support all majors outside of the college of engineering. These additions are ultimately the most helpful yet most time-consuming part of the project because the approach to these features would require a UCLA requirements/4-year plan API (which could not be found online) or extensive data-mining. As a result of this, we were unable to include these features in our project submission. On the other hand, some minor changes I would like to implement are making the website responsive (beautiful on smaller screens like phones), giving users the ability to search courses by ID, hooking up the contact page to my email, and more. Most of the minor changes are additions that could be added fairly quickly but lack the importance that the major changes possess. Finally, after I implement the numerous changes stated above, I want to publish BruinPlanner by hosting it in the cloud. The ending goal would be to make it available as a tool for the students of UCLA!