Spring 2019

TutorUP

Alexa Baldwin
baldwina18@up.edu

Danh Nguyen
nguyenda18@up.edu

Elias Paraiso
paraiso18@up.edu

tutor up

FINAL REPORT • APRIL 23, 2019

Team Members
Alexa Baldwin, Danh Nguyen, Elias Paraiso

Faculty Advisor
Tammy VanDeGrift, PhD

Industry Advisor
Casey Sigelmann, Precoa

Client
Tau Beta Pi Oregon Gamma Chapter

Introduction

TutorUP is a peer tutoring network for University of Portland. The purpose of the network is to make academic support more accessible for students who find themselves needing more assistance than on-campus resources and professors can provide. The goal of TutorUP is to provide tutoring for a wider breadth of subjects and to make it easier for students to find and connect with tutors. In order to use TutorUP, the user must have access to a valid University of Portland email address. Once logged in, users can either create a tutor profile or search for a tutor. Tutors create profiles that include majors and minors, a brief bio, general availability and the courses he or she can tutor. Tutees can search for a tutor by major, course, or name.

Throughout the implementation phase, the team achieved several substantial milestones. The first milestone was installing the application locally and getting the architecture set up in a secure and reliable fashion. This phase included implementing the Express API, integrating the React frontend with the Express API, implementing user authentication with JSON Web Tokens using the Passport module and integrating Redux for state management with React. The next milestone was building out the tutoring profile and styling the application. We built the form components using the Material styling from MaterialUI. This phase also included modifying styling to include custom colors and layouts. The final phase of implementation was creating the administration layer and finalizing the search page. This layer allows the application to have control over users and subjects. The search page allows users to search, filter and sort through the existing tutor profiles.

The team faced several challenges in developing the application. Initially the team wanted to create the application using the university's systems. The goal was to have TutorUP behind UP's authentication and to have some course and student information pulled from Banner. However, security is a top concern for Information Services, so they were hesitant to allow access to these resources. Another challenge the team faced was what stack technologies we wanted to use. We initially created a MEAN stack application with MySQL, Express, Angular and Node.js. This stack posed several problems as we were unable to get the project running locally for all team members. We were also having difficulties getting the application frontend and backend components integrated. As a result, our team switched to a MERN stack with MongoDB, Express, React and Node. While this switch was helpful as it fixed the aforementioned issues, it also created conflicts within the team and added a learning curve to the project while we became comfortable with the React framework. The final major challenge our team faced was feature evaluation. Initially we had envisioned features such as ratings and in-app messaging. However, when we originally pitched these ideas, we failed to consider how difficult these would be to monitor for harmful language.

Overall our team is proud of the application we created and the challenges we overcame to get there. We were surprised with how efficient our meetings were this final semester and how easy it was to learn React. We believe we have created a stable application that will perform well, and scale as needed. While we struggled with teamwork and communication at the beginning, we found a way to appreciate each other's skills regarding the construction of our application.

The remainder of this document will discuss in more detail how our product works and the technical outcomes of the project. Specifically, it will give an overview of the frontend and backend organization and a description of the various use cases for the application.

Technical Outcomes

Being a user-driven web application, TutorUP is best described through the roles and actions of its users. An overview of the users' functions has to be given before going through the architecture and structure. There are three defined types of the users for the application: tutees, tutors, and admins. Their abilities are shown in Figure 1.

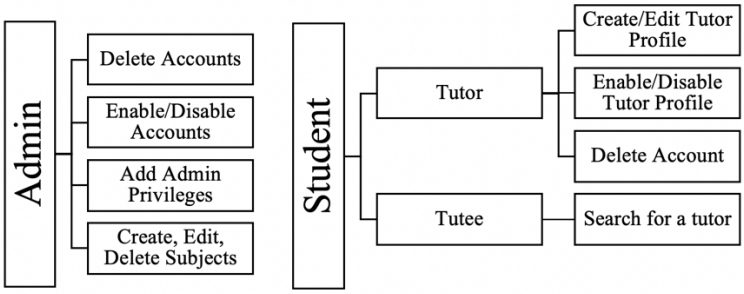


Figure 1: TutorUP User Types

The components related to the architecture of the app are noted in the diagram below (Figure 2). The app is based on the MERN stack architecture – MongoDB, Express, React, and Node.js are the elements. MongoDB is the database that persists data for the application. Express is a lightweight web application and API framework for Node.js. Node.js is a JavaScript runtime environment built on Chrome V8's JavaScript engine that allows developers to run JavaScript code server-side instead of only in the browser like a decade ago. The application is hosted in the cloud by Heroku. Additional information can be found in the References + Glossary section of the report. As for the components of the product, they can be split into two categories: components related to the REST API backend and components related to the React frontend.

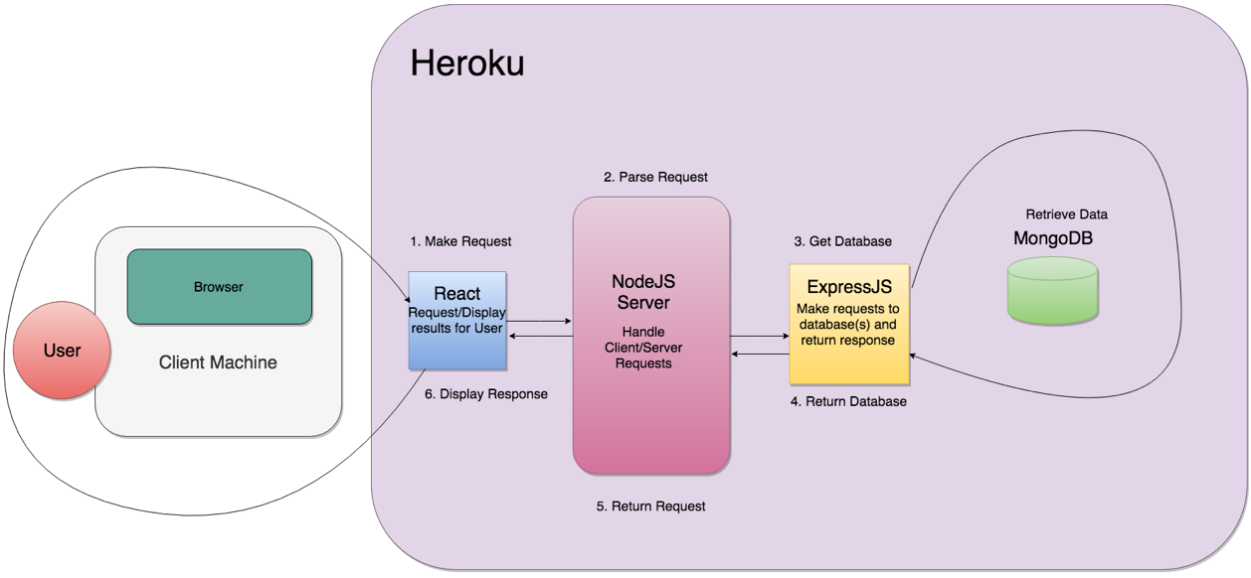


Figure 2: TutorUP Framework Interaction

Part I. Application User Interface + Frontend Components

The following section demonstrates the application's state of flow, with images giving context for the descriptions of each page's functionality. It follows the perspective of a new user registering, creating a tutor profile, and gaining admin privileges.

The website is currently accessible with the URL: https://www.tutorpilots.com. The user is greeted with a homepage that has a random background image. The new user can register in two ways: by clicking the hamburger menu icon and selecting the 'Register' link or by simply clicking the 'Click Here to Get Started!' button on the homepage.

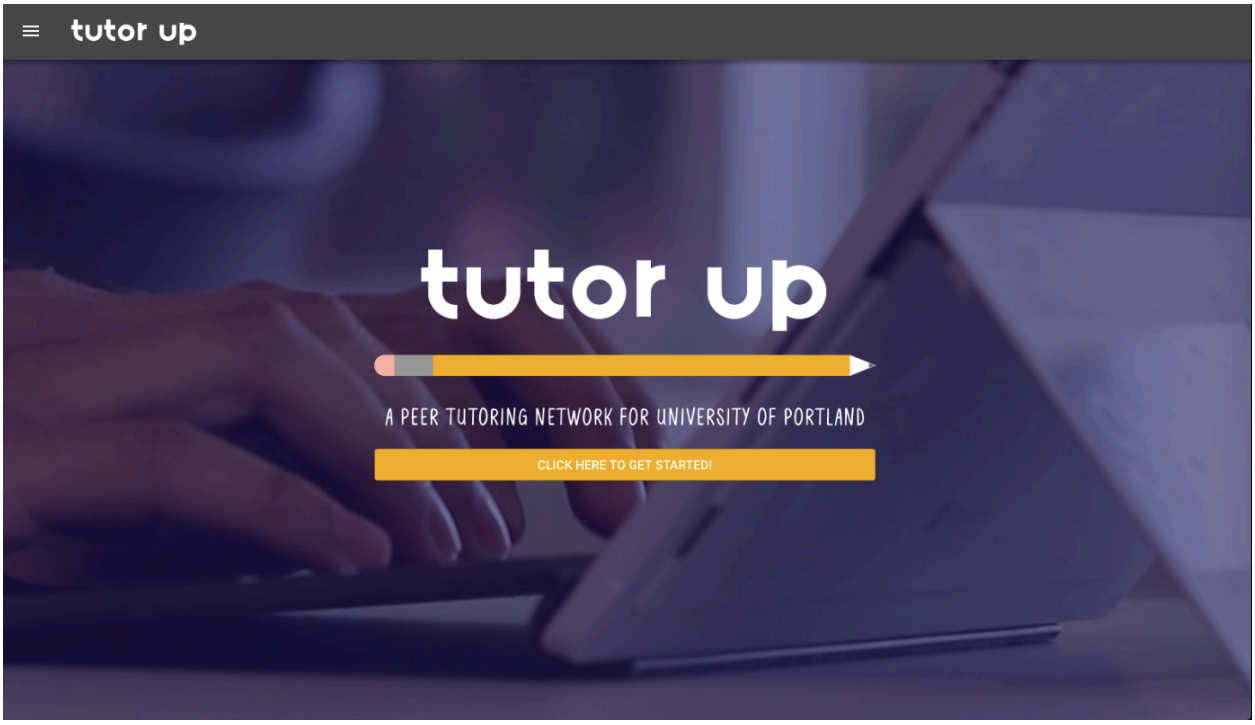


Figure 3: Home Page (with random background images at each revisit)

The register and login components are the only public-facing pages for the application, hiding UP students' profile information and search functionality from non-UP users. To register, the new user needs to provide their first name and last name, their UP username (UP email without '@up.edu') and a password that fits the login criteria (Figure 4.1). The user retypes their password in the 'Confirm Password' field because they will not be able to change their password after registering. A dialog will pop up alerting them to check their email and click the confirmation link to be able to log in (Figure 4.2). Once the newly registered user clicks on the confirmation link in their email (Figure 5), they will be able to sign in to the application. Their confirmed property is set to true after clicking the confirmation link.

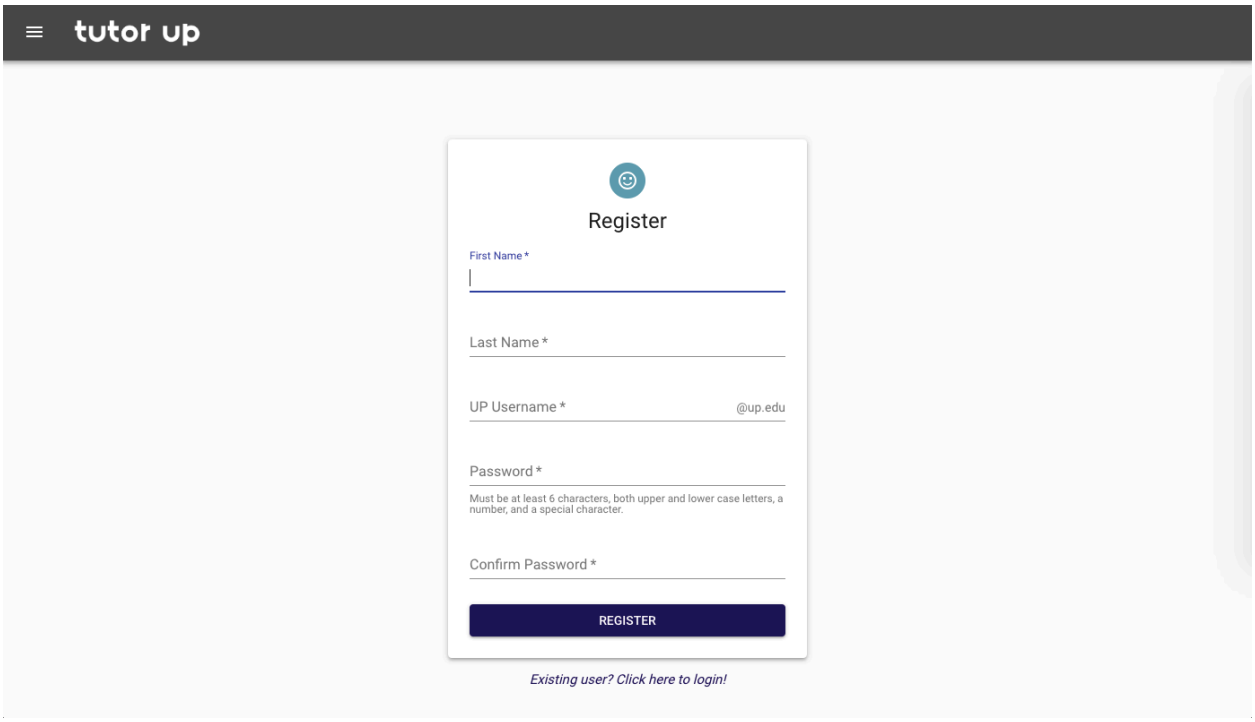


Figure 4.1: Registration Page

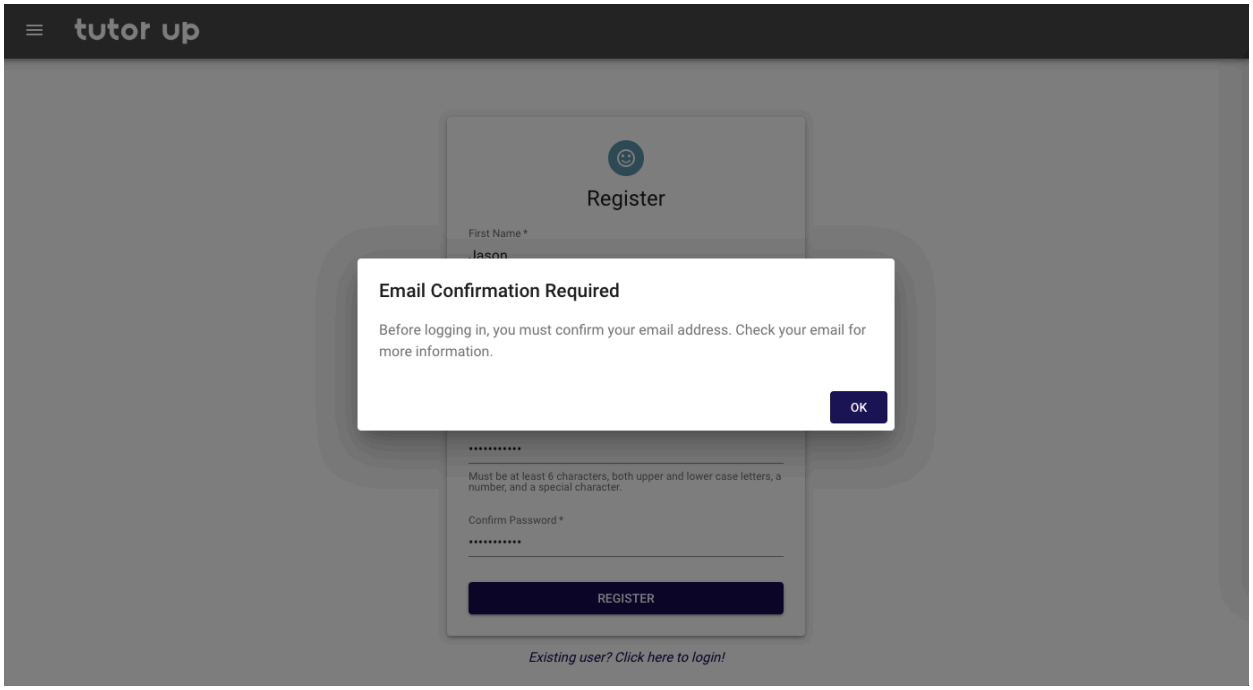


Figure 4.2: Email Confirmation Dialog

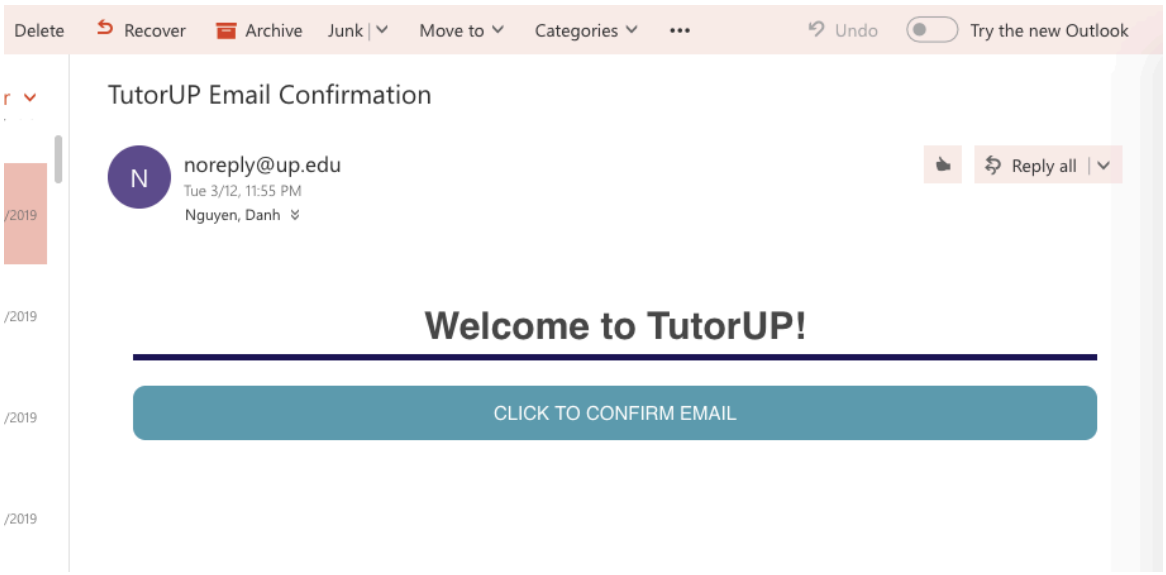


Figure 5: Confirmation Email

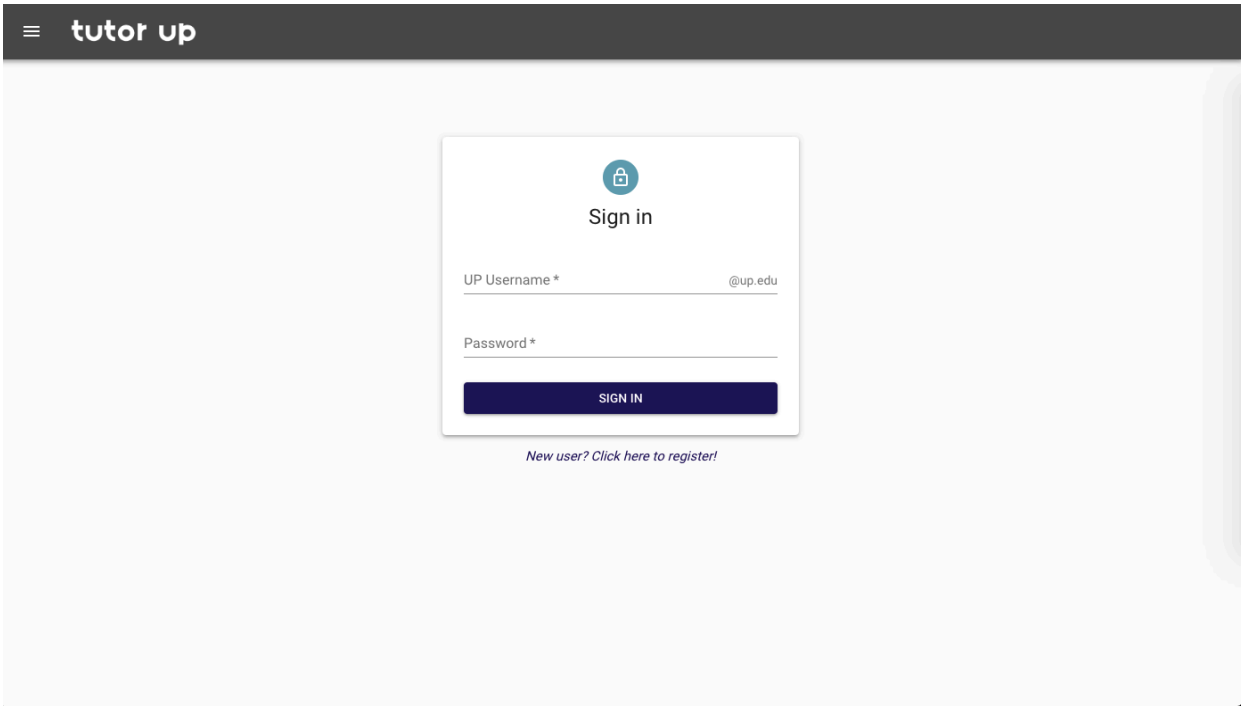


Figure 6: Login Page

After logging in, the new user is redirected to the dashboard (Figure 7.1). There are two options they can choose: 'Become a Tutor' or 'Find a Tutor'. Becoming a tutor requires them to fill out a tutor profile and finding a tutor just allows them to search for tutors if they only want to be a tutee. The next demonstration will be of the user creating a new tutor profile and searching for themselves.

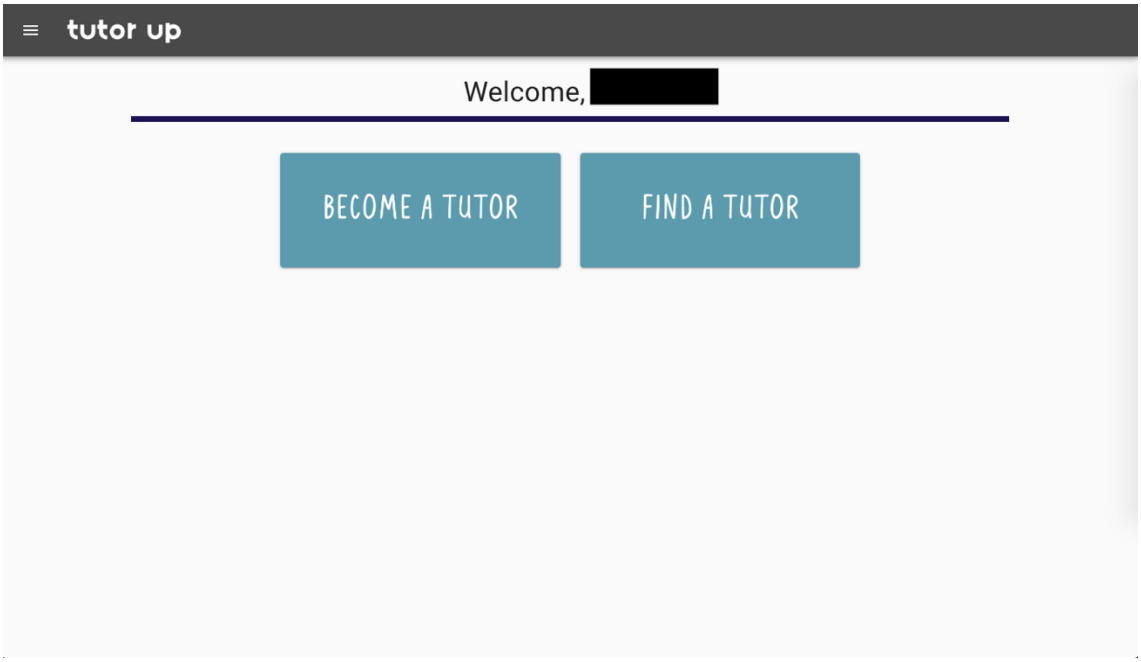


Figure 7.1: User Dashboard Page (if user is not a tutor)

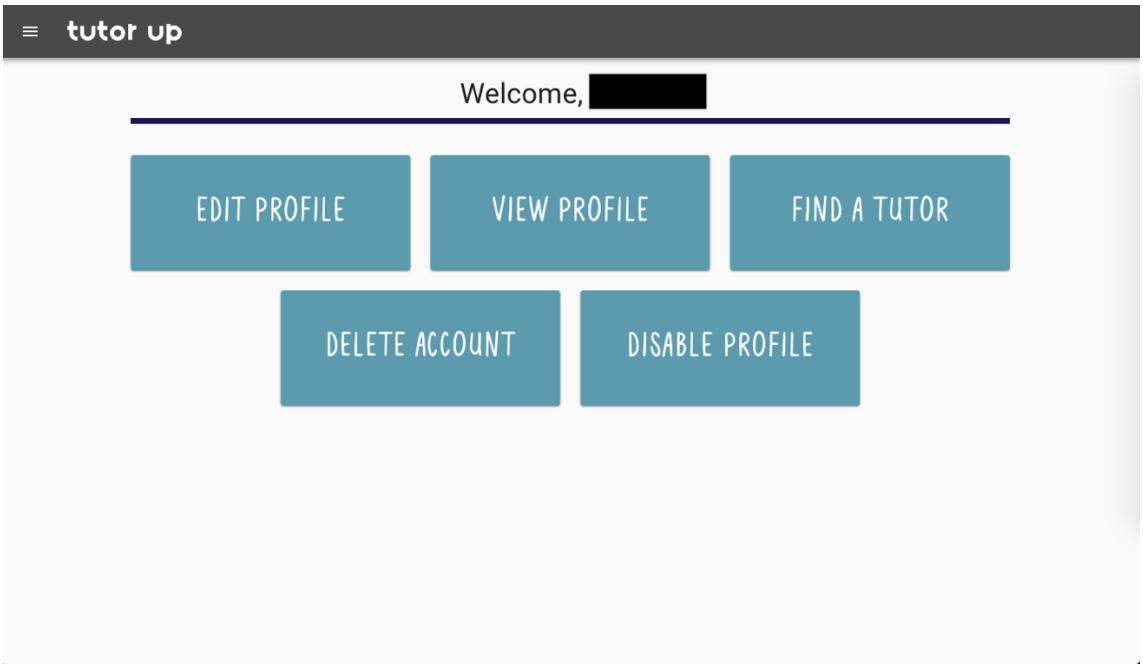


Figure 7.2: User Dashboard Page (if user is a tutor)

Clicking the ‘Become a Tutor’ button will redirect the user to a form to create their tutor profile (Figure 8). If the user is already a tutor, they have the option to edit their profile by clicking the ‘Edit Profile’ button. The profile component contains the user’s UP email, the option of being a paid or volunteer tutor, a short bio, and a statement of availability. Tutors also input their major(s) and minor(s) and the classes they are able to tutor.

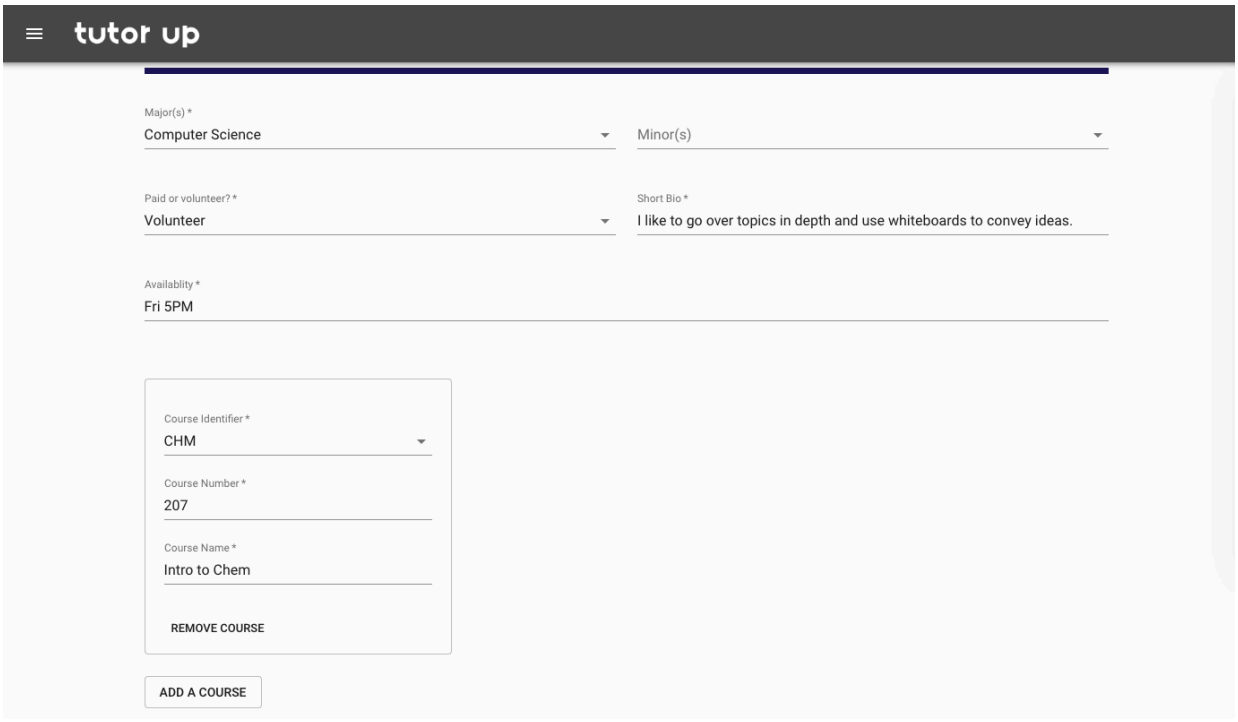


Figure 8: Profile Create/Edit Page

Once a profile is created, the user is redirected to the dashboard that now has new options (Figure 7.2). A tutor is able to disable their profile from this page by clicking the ‘Disable Profile’ button. Disabling a profile will prevent it from being displayed on the search page. When the user clicks the ‘Disable Profile’ button, it is switched to a ‘Enable Profile’ button, and they will need to click it again to bring their profile back to the search page. The tutor user can view their profile by clicking the ‘View Profile’ button. This shows their profile the way other users will be able to view it through the search page. The user can also delete his or her account from this dashboard if desired.

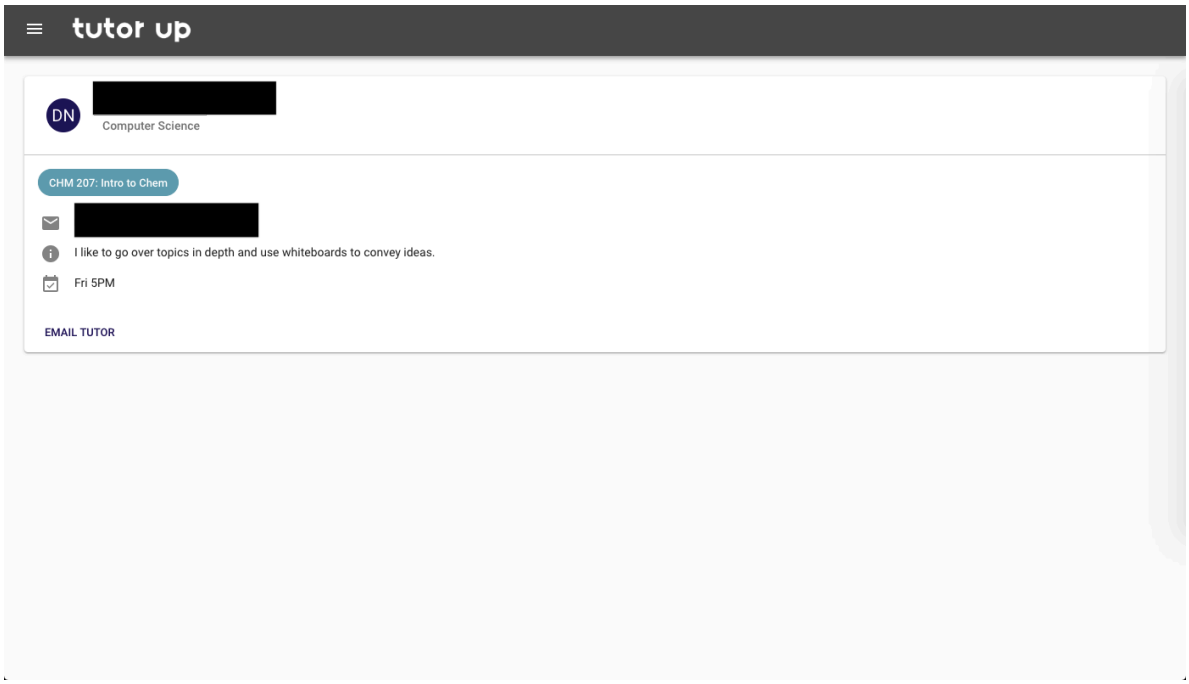


Figure 9: View Individual Profile Page

A user can search for a tutor by clicking the 'Find a Tutor' option in the hamburger menu. Users are able to search by name, major and course using the search bar. Possible filters and ordering options include: filtering by subject, paid/unpaid tutors, and filtering by first name or major. Users are able to shuffle tutors displayed and the data pulled by the API is pre-shuffled so that every tutor has an equal chance to appear at the top of the results.

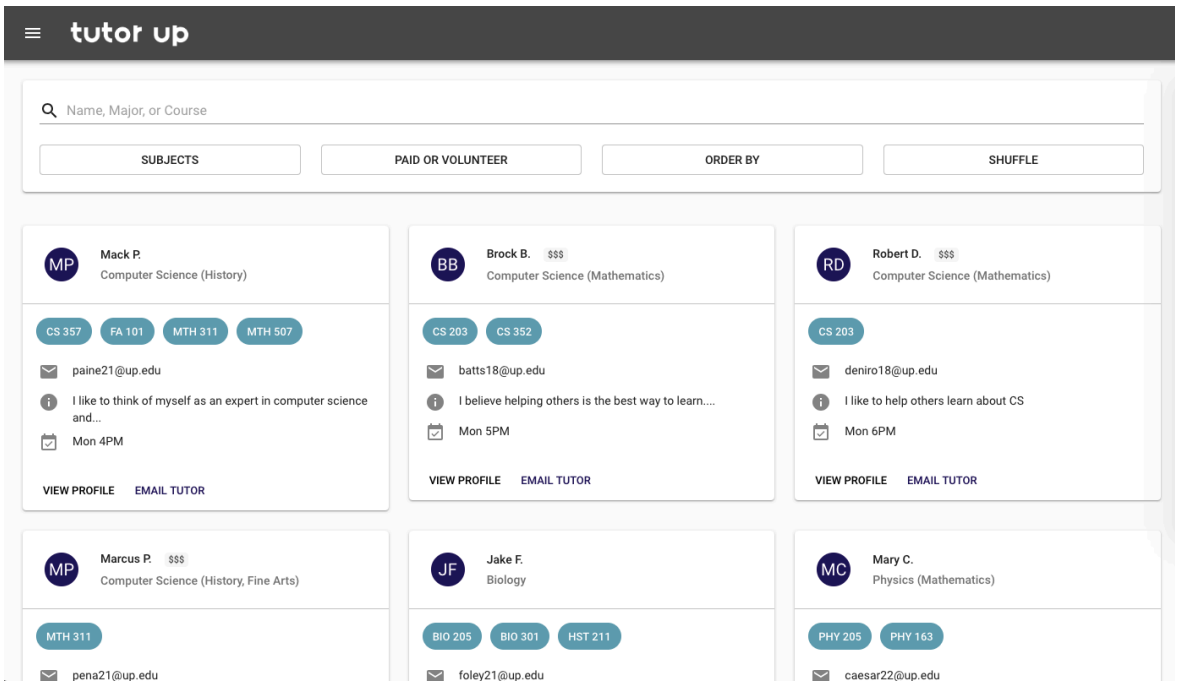


Figure 10: Tutor Search Page

The new user can become an admin of the application if another admin grants him or her admin privileges (the first admin for the application will have his or her admin privileges set manually in the database). After being enabled as an admin, the new user will need to log out and log back in to see the admin tools in the menu (Figure 11). The admin components are protected by routes requiring users with the `isAdmin` property set to `true`. The links for an admin user are only visible to admins, and are not accessible by regular student users. The `isAdmin` property is set `false` for all users by default during registration.

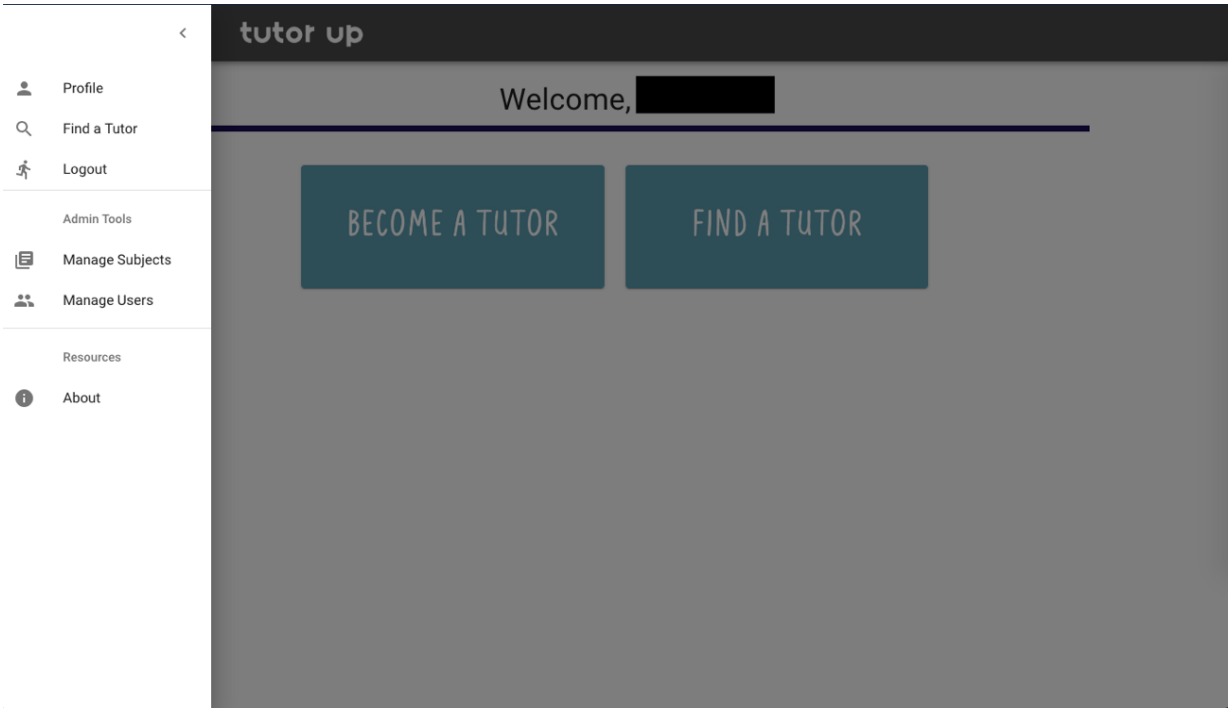


Figure 11: User Has Access to Admin Links After Re-login (Admin only)

Admins have the ability to manage users and subject via the admin tools. When an admin clicks the ‘Manage Subjects’ option the subject dashboard page is loaded (Figure 12.1). Clicking ‘Create/Edit Subjects’ takes the admin to an editable view of the subjects where subjects can be created, edited or removed (Figure 12.2). Clicking ‘View Subjects’ navigates the admin to a table view of all the existing subjects (Figure 12.3). If the admin goes back to the hamburger menu and clicks ‘Manage Users’ a table of the users loads, and the admin can change admin privileges and whether a profile is enabled or disabled for each user. Admins can also delete user profiles on this page.

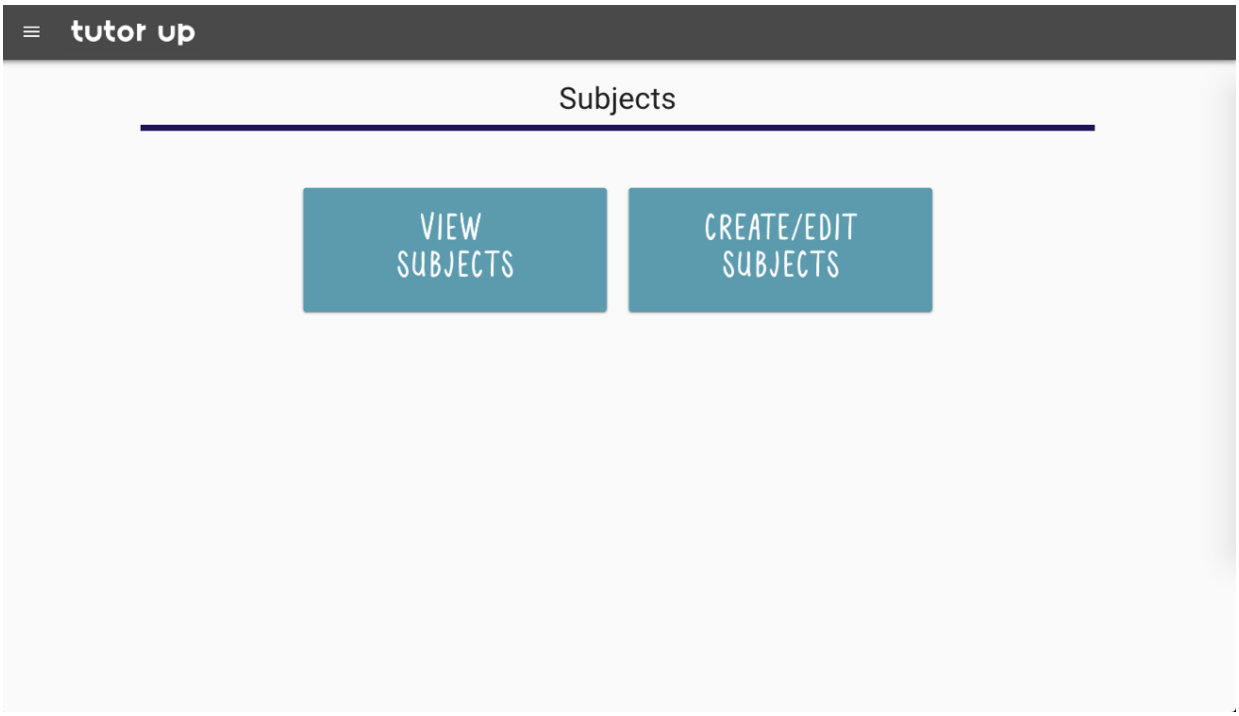


Figure 12.1: Subject Dashboard Page (Admin only)

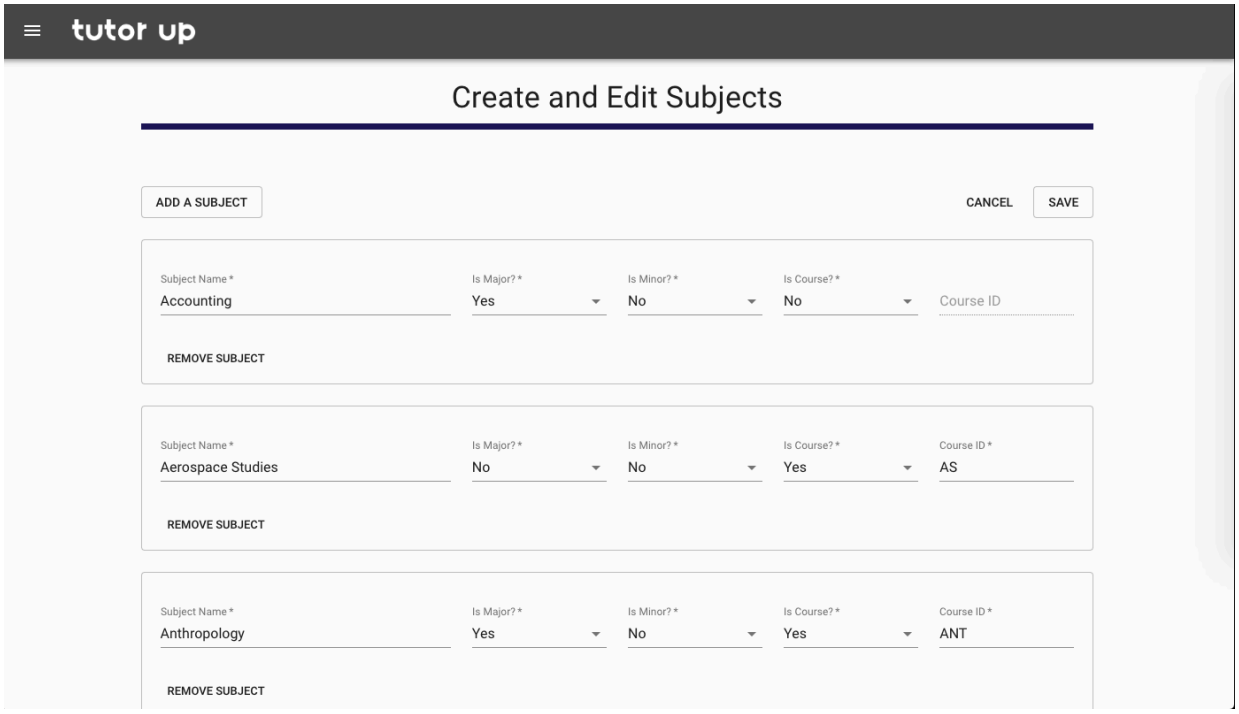


Figure 12.2: Add/Edit Subject Page (Admin only)

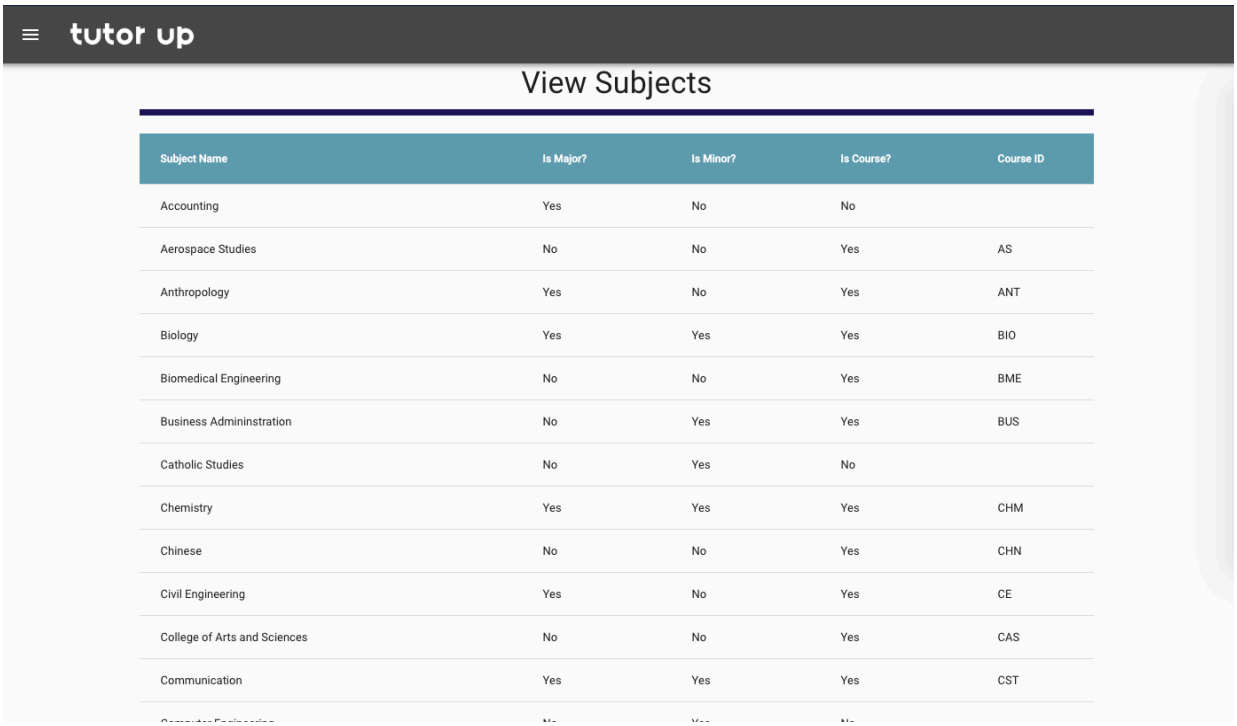


Figure 12.3: Subject View Page (Admin only)

Part II. Application Backend + Frontend Details

The backend is organized into three significant components: models, routes, and validation. Application data is modeled by a specified Mongoose Schema that is used for Object Data Modeling to the MongoDB database. The schemas for our various objects are explained below.

Table 1A: User Schema

User Schema		
Property	Type	Notes
firstname	String	User's first name
lastname	String	User's last name
email	String	User's UP email
confirmed	Boolean	User has confirmed email address with link
password	String	User's hashed password
isAdmin	Boolean	User has admin privileges
hasProfile	Boolean	User has created profile and can be disabled/enabled
disabled	Boolean	User or admin has disabled their profile

Table 1B: Profile Schema

Profile Schema		
Property	Type	Notes
user	User	User the profile belongs to
handle	String	User's username
major	String Array	User's major(s)
minor	String Array	User's minor(s) (optional)
bio	String	User's bio
availability	String	User's availability
courses	Object Array	Courses the user can tutor
type	String	Paid or Volunteer
disabled	Boolean	Toggle for disabling profile
date	DateTime	Date profile was created

Table 1C: Course Subject Schema

Subject Schema		
Property	Type	Notes
id	String	Subject Abbreviation
name	String	Subject Name
isMajor	String	Whether or not subject is a major
isMinor	String	Whether or not subject is a minor
isCourse	String	Whether or not subject is a course

Table 1D: Tutoring Course Schema

Course Schema		
Property	Type	Notes
id	String	Subject Abbreviation
number	Number	Course Number
name	String	Course Title
courseSubject	String	Subject Full Name

The REST API for the application is based on routes that define separate pieces of functionality. The routes are split based on the models used by them. The routes are described in Table 2.

Table 2: Application Routes

Route	HTTP Method	Access	Function
/api/users/register	POST	Public	Registers a new user, the user's specified password is hashed with the bCrypt module and stored in the database. User is sent confirmation email
/api/users/login	POST	Public	Login user only if they have confirmed their email, check password with hashed password in the database.
/api/users/current	GET	Private	Return current user information: email, firstname, lastname, email, isAdmin.
/api/users/admin	POST	Private	Update isAdmin property for user.
/api/profile/all	GET	Private/Admin	Retrieve all enabled users' profiles.
/api/profile/allUsers	GET	Private/Admin	Retrieve all users' profiles (enabled + disabled)
/api/profile/handle/:handle	GET	Private	Get profile by handle.
/api/profile/	GET	Private	Get current user's profile.
/api/profile/	POST	Private	Create or edit profile.
/api/profile/disableProfile	POST	Private	Disable a profile.
/api/profile/enableProfile	POST	Private	Enable a profile.
/api/profile/	DELETE	Private	Delete current user's profile and account from the database.
/api/profile/id	DELETE	Private/Admin	Delete any profile or user.
/api/courses/	GET	Private	Get all courses for current user.
/api/courses/	POST	Private	Create or edit a course.
/api/courses/	DELETE	Private	Delete a course from profile.
/api/subjects/	GET	Private/Admin	Get all subjects.
/api/subjects/	POST	Private/Admin	Create or edit a subject.
/api/subjects/	DELETE	Private/Admin	Delete a subject from the database.
/email/confirm/:id	GET	Public	Retrieve id from URL to confirm user by email.

The users route handles registering and logging in new users and user model updates. When registering a new user, the user's specified password is hashed with the bCrypt module and stored in the database. A confirmation email is sent to the user's email address and the user is not able to log in until they click the link confirming their UP email address. The disable route allows users to be disabled on the application search page, the current user route gets the information related to the application user, and an admin route updates the `isAdmin` property for a user.

The profiles route handles profile retrieval and profile creation and updating. There is a route to get all users' profiles (only enabled profiles), a route to get all enabled and disabled users' profiles for

admins, a route for getting individual profiles, a route for getting the current user's profile, routes to disable and enable profiles, a route for a user to delete their own profile, and a route for admins to delete a user's profile.

The courses route handles retrieval, updates, and deletions of courses; and the subjects route handles retrieval, updates, and deletions of subjects. Only admins are able to create, edit and delete subjects.

The frontend contains all the React code related to the client's use of the application. Frontend views are divided into static function and dynamic class components (Appendix C). Functional React components are just pieces of the frontend that do not change based on any logic. Dynamic class components do require changes and have either local state or connected state. The connected state means the global state stored in Redux. Redux is a design pattern in which the global state can only be changed by dispatching actions and reducers. Actions describe the trigger that causes a change in state. Reducers return the modified state to be used in the application.

All frontend routes but the login and register pages are protected by JWT (JSON Web Token). This token is set when the user logs in with the correct password. Every private route is protected by JWT, and admin routes are protected by a variable checking for admin users in the database. For email authentication, all users of the app are required to have a UP email to access tutor profile search and profile creation. The SendGrid API is used to send email confirmation links to users' emails when they first register.

The team uses the Redux state management library for storing the global state essential for the application. React components can have internal state passing state across components can get complicated. Redux solves the problem by connecting a 'global state' to each component that needs access to it. Global state for the application includes state of authentication, errors in the app, profile operations, and subjects.

The team uses the cloud MongoDB database in MLab to store all application data. The data is organized into three collections – users, profiles, and subjects. The application is able to perform create/read/update/delete operations quickly on the database based on the user's inputs and the backend server-side functions. After recognizing the team's success in developing and deploying to the cloud with MLab, Heroku, and Git, UP IS recently contacted the team about migration the application to Amazon Web Services and are currently having discussions on how to extend and maintain the application with AWS.

Although the team was mostly able to stay faithful to the original design and functions, original functional and non-functional requirements had to be adjusted. After realizing that moderating would require significant overhead, the team chose to remove the functionality of commenting and giving star ratings to tutors' profiles. Similarly, the profile page design was changed so default avatars were given to each user based on their initials, because moderating appropriate user avatars would require overhead. Additionally, we had planned to implement a more sophisticated availability system. We were unable to implement this feature due to time constraints.

Process Outcomes

Initially, our team believed that the application would be best if integrated with the university's systems. We hoped to integrate UP's authentication system and student information into our application to facilitate the login process for students and to ensure that access would be limited to the UP-student population. To our surprise, IS did not want to let us access either. They considered the requests to be too risky, as their main priority is to ensure student safety and that their procedures comply with FERPA standards. This was unexpectedly fortunate for us because it likely would have been far more difficult to integrate our application with the university systems than it was to create our own registration and authentication process. We also would have lost much of our freedom to choose the application components and design if we had integrated with their systems. Since IS was not able to provide course data to the team with the Student Banner API, we implemented the subject API that is managed by admin users.

In the search for a client, as required by our capstone class, our team originally thought that the on campus tutoring resource center, the Shepard Academic Recourse Center or SARC, would be a fitting client for our product. We thought that the tutoring resource center would be supportive of an on-campus web application to help connect students with peer tutors. However, during our meeting with the current head of SARC, our team realized that he was not receptive to the idea and so we had to search for a new client.

We then reached out to the university's Athletics Department to query if they would be a fitting client or not. Our point of contact in Athletics was immensely more friendly and willing, however their needs did not align with our application's design and intended functionality. Eventually we did find an appropriate client when we decided to contact the on-campus club, Tau Beta Pi, and they expressed their interest in being the client and maintainer of our application post completion.

Among these technical challenges that our team faced, we ran into our own personal challenges as we continued through the process. In the first phases of this project, our team ran into personal issues due to differing levels and expectations of communication and commitment to the project. At first, this caused conflict within the team, however we were all able to stay calm, move forward, and resolve these issues. We are incredibly lucky that we were able to resolve these problems relatively quickly and painlessly because our team far more productive and functional when we worked together as a team.

Table 3 (below) shows our preliminary sprint schedule from our original design report (only Sprints 1 through 5 were in this report). Although we did not end up following the schedule exactly as it was written, we found that we did follow the general sprint timeline because we were able complete or implement our intended milestones by the end of each sprint. As we proceeded through each sprint, we often re-discussed many of the proposed features of the application, removing and modifying where we saw fit. Sprint 6 and Sprint 7 were the only sprints that were planned right before they started.

Since we were unable to collaborate with IS, our plan to connect to UP's databases by the end of Sprint 4 was archived and replaced with the plan to research, purchase, and connect a reliable database to our application for production. Similarly, our plan to familiarize IS developers with project was completely archived since we were unable to use their systems. Sprint 4 is also when our team recreated the project from scratch with the React framework. Although this was not a part of our initial project plan,

this change solved several of the problems we were having, allowing us to complete the future milestones in an easier fashion.

Table 3: Sprint Schedule

Sprint Number	Sprint Leader	End Date	Intended Milestone(s)	Completed Milestones
1	Alexa	2/26/18	- Basic CRUD functionality for application - Connect application to local database - Application Demo	- Basic CRUD functionality for application - Connect application to local database - Application Demo
2	Danh	3/26/18	- Final Project Plan Report - User Interface Mock-Up - Poster Completed - Switch to Angular Front-End instead of JavaScript template - Avatar upload capability (with Formidable module) - Search Functionality	- Final Project Plan Report - User Interface Mock-Up - Poster Completed - Switch to Angular Front-End instead of JavaScript template
3	Elias	4/16/18	- Poster Presentation - Finalize beta demonstration of application - Design Report Completed	- Poster Presentation - Design Report Completed - Finalize beta demonstration of application
4	Alexa	1/1/19	-Set up authentication -Hook up to UP databases -Familiarize IS developers with project (README documents, mini tutorials)	- Project Migration from MEAN to MERN
5	Alexa	2/1/19	(Add app enhancements) -REST API for IS to easily run CRUD operations -Instant messaging service (using Twilio API) -Visual ratings for tutors and tutees (using Pusher API)	- Familiarization with React - Basic Search Implementation - Created Admin Routes - Improved Design and Styling - Added More Profile Attributes - Disable/Enable Profiles - Background Images
6	Danh	2/22/19	*Planned at beginning of Sprint* - Beta Sessions - Feedback Bug Fixes - Email Authentication - Admin Management APIs	- UP Email Authentication - Admin Subject APIs - Improved Search & Filter - Added UP Subjects to Prod - Created Admin User API - Bug Fixes
7	Elias	3/29/18	*Planned at beginning of Sprint* - User Testing and Deployment - Final Report & Presentation	- Beta Sessions and Feedback - Improved Security, required login for viewing tutor profiles - Final Report & Presentation - Bug Fixes - Planning and Preparation for Deployment and Handoff

For the features corresponding to Sprint 5, all the milestone enhancements were archived, due to a complete change in framework, components, and design. We had initially hoped to implement an instant messaging service to simplify the process for students contacting tutors and similarly intended to implement a rating and review system so the users could endorse the "good tutors" and potentially report the "bad tutors". However, we decided not to continue forward with either of these features due to the fact that both of these features would require moderators to ensure that these features were not abused by potential malicious users. Sprint 5 instead involved a period of familiarization for half of the team with the new framework. Once up to speed, we were able to lay out the structure for each of the page routes, apply beautiful styling, and even implement most of the desired functionalities.

As Sprint 6 and 7 were not planned until the end of Sprint 5, our completed milestones closely matched what we intended. Both of these Sprints focused mostly on gaining user feedback, fixing various bugs, improving features as requested from feedback, and completing our final report and presentation. We learned a great deal from our beta sessions regarding how well the users liked the app, how likely they would be to use the app, and what features they would like to be improved. As seen in Appendix B1 and B2, there were no users from the either of the beta sessions that said they would not use the app. In fact, the majority of users said they would either be "very likely" or "extremely likely" to use the app as either a tutor or tutee.

Our conversation with IS started up once again when we began discussing how we intend to host the production version of the application and database. Our team has been considering whether to use our allocated money to purchase a lasting production database and domain or whether to have IS host the application for us, ensuring a lasting production version. Once Tutor UP is fully deployed in its production form, it will be handed off to the university club, Tau Beta Pi. Tau Beta Pi's webmasters Jenn Loui and Jason Twigg will serve as the application admins and will have the ability to manage the subjects and users. Since the admin will have control over which users can become the next admin, they have the ability to handoff the responsibilities when surpassed by a new club president.

Testing

Before we had implemented the major features of the application, we had held a survey to gauge interest in the UP community. The results from the survey suggested that most students on campus thought that the app would be useful (Appendix A). These results helped reinforce our perception that a UP students could use more assistance finding help on campus.

Once we had implemented the app's main functionality, we held two separate user testing sessions. Our main goal from these beta sessions was to record the users' thoughts, to discover undiscovered bugs, and to help advertise our product. From the first beta session, we were able to discover a few critical bugs and obtain a lot of feedback. The overall feedback from testers was mostly positive (Appendix B), however we did receive a number of suggestions from the test users.

One example of a suggestion that we implemented was the desire for a label showing the password requirements, as users failed multiple times to create a password when they did not know the criteria. In addition to usability improvements like this, there were also several bugs found during the testing session. One major bug had to do with validation on the profile form. Users were able to submit the form successfully with some of the required fields missing. Prior to testing, we hadn't realized that the

application HTML `required` tag worked differently on the dropdown items than the text fields, thus we implemented a new solution to ensure all the profile fields were filled out before submission.

Project Contributions

Each team member contributed to the project in a variety of ways, in this section we will describe the contributions of each team member.

Alexa acted as our team's project leader, graphic designer, and main frontend web developer. Since the project was originally proposed by Alexa, she was very active in the beginning of the project with the product design and planning documentation. She handled styling and responsivity of the application's frontend. Alexa was also heavily involved with the frontend implementation of the admin page and its respective APIs. Among these major contributions, Alexa assisted in various programming tasks related to the tutor profiles fields and subject creation and validation.

Danh acted as our team's main architect. Danh heavily researched the MERN app architecture and fully recreated the application using MERN because he believed it to be superior to the MEAN architecture we had before. This was immensely valuable for our team since the newly created application in MERN was quick to install and run on each member's local machines, an accomplishment we had not reached before with the MEAN application. Danh also acted as our team's main back-end developer. He set up the configurations for the app database and domain. Danh implemented the JWT password system and the email authentication system, both of which are crucial for the security of our student users. Danh also took charge of the application's deployment pipeline by configuring the project GitHub integration and Heroku production deployment.

Elias was charge in implementing many of the application features related to searching and filtering. He created the search bar and filter dropdowns on the "Find a Tutor" page, modifying their functionality to give the desired balance of quantity of results and similarity to the text being searched. Elias also assisted Alexa with various frontend tasks involved with the application's profile submission validation and user tips. Similarly, Elias took charge of creating the application help/about page containing the FAQs of the app and various tips to help the users understand how to optimally and appropriately use the app.

Conclusion

In conclusion, our team successfully created a stable responsive web application to help connect students with peer tutors at University of Portland. Our team believes we were able to accomplish the main objective of our project.

Throughout this project our team faced both interpersonal and technical challenges. Our team persevered through various communication difficulties to create a product that we are all proud of. Our team is most satisfied with the styling, responsivity, stability and security of our application. One of our top achievements was developing a way to work around our inability to connect to the university's login infrastructure by adding an email confirmation layer to the application.

While our team is proud of the application we developed, there are several things we would do to improve the design and process. The main thing our team would change is the amount of time we spent planning. Had we spent more time looking into what frameworks to use and how we wanted to set up data objects within the application, we would have saved ourselves a lot of time. Additionally, our team wished we had a more realistic scope in mind when we began our project. Had we foreseen some of the issues with integration and implementation mentioned above, we would have eliminated more features at the beginning.

If our team had more time to develop this product, we would like to add a better scheduling and availability tracking system. This was a feature we had initially planned on implementing, but quickly realized it was not plausible given the higher priority features that we needed to implement. Despite this, we are very proud of the TutorUP application that we developed, and we look forward to handing the admin privileges over to Tau Beta Pi.

Acknowledgements

We would like to acknowledge and thank everyone that assisted our team throughout the capstone process.

Dr. Tammy VanDeGrift
Faculty Advisor • Capstone Professor

Casey Sigelmann
Industry Advisor

Ryan Jefferis
Capstone Professor

Tau Beat Pi Oregon Gamma Chapter
Client

Beta Testers

Information Services

Background Photo Models
Curtis Le • Ellie Jacobs • Jenn Loui
Nick Accuardi • Nischal Mali • Surabhi Jogleka

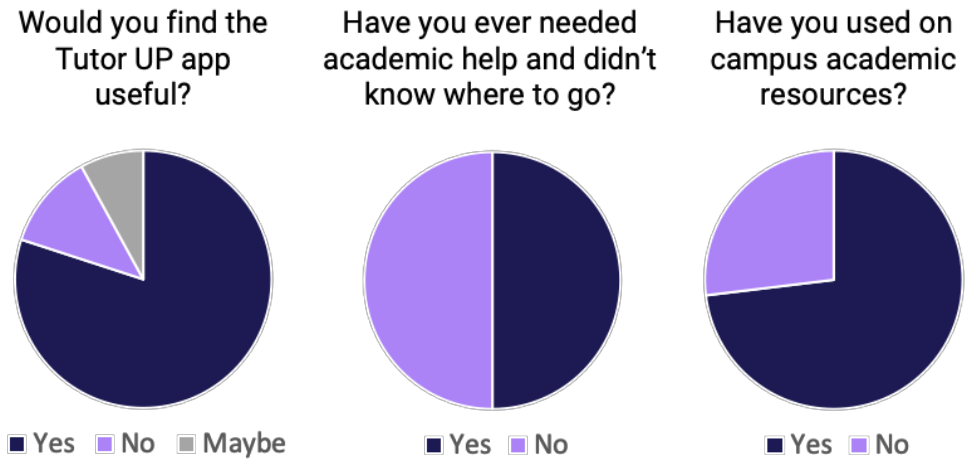
References + Glossary

Angular	TypeScript based open-source web application framework from Google. Original framework chosen for TutorUP. *https://angular.io/*
Express	Open-source web application framework for NodeJS under MIT license. *https://expressjs.com/*
JWT.IO	JSON web token manager that allows for encryption and decryption of user passwords. *https://jwt.io/*
Material Icons	Google's open-source icon library. *https://material.io/tools/icons/*
Material UI	React components that use Google's Material design standards. *https://material-ui.com/*
MongoDB	Cross-platform document-oriented database program. *https://www.mongodb.com/*
Mongoose	MongoDB object modeling tool for NodeJS. *https://mongoosejs.com/*
NodeJS	Open-source JavaScript runtime environment. *https://nodejs.org/en/*
Passport	Authentication middleware for NodeJS. *http://www.passportjs.org/*
React	JavaScript library for building user interfaces. Maintained by Facebook. Replacement frontend library chosen for TutorUP. *https://reactjs.org/*
TutorUP Design Document	Original design document created by our team during the Spring 2018 semester.
TutorUP Poster	Poster summarizing the work completed in the first semester of this project (Spring 2018).
UP Brand Book	University of Portland's marketing standards. Used in our project for color palettes. *https://www.up.edu/marketing/files/up-brand-book-2016.pdf*

Appendix

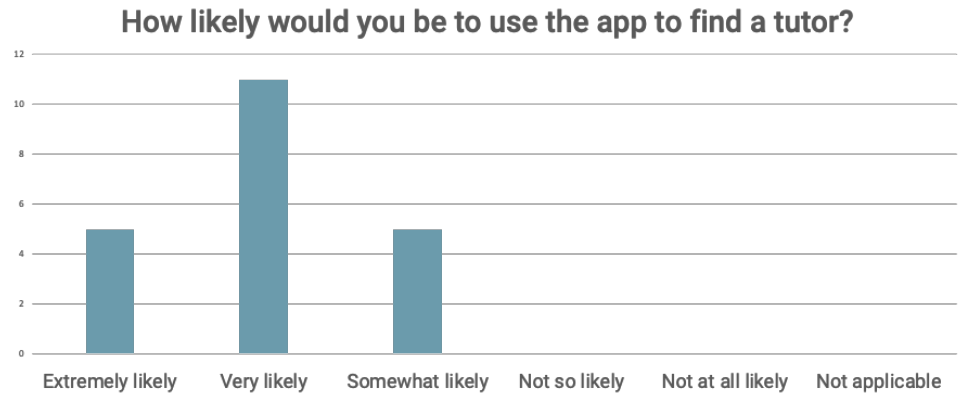
Appendix A: Initial Survey Results

Before beginning the implementation of TutorUP, we surveyed 30 students across various years and majors to gauge how TutorUP would be perceived on campus. The results are summarized below.

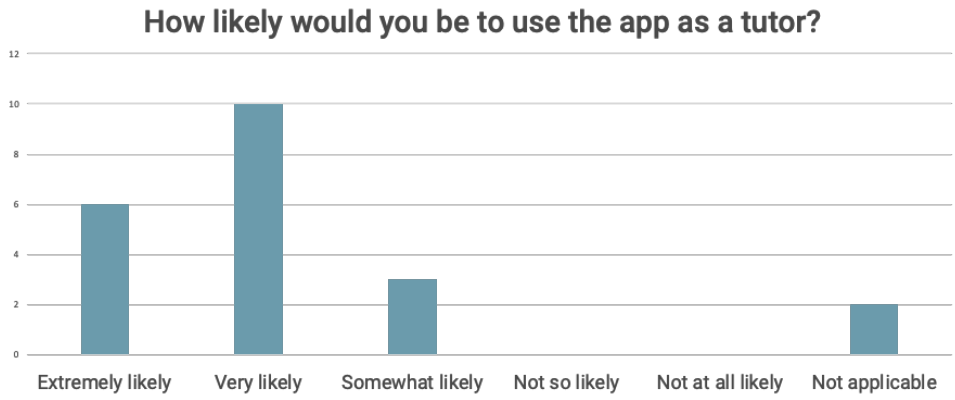


Appendix B: Beta Testing Results

These results are based off a survey that users took after testing TutorUP. There were 21 individuals that participated in beta testing. We asked users to rate ease of use and the overall look of the app on a five-star scale. The average ease of use rating was 4.4 and the average overall look rating was 4.5. We also asked users to rate how likely they would be to use the app as both a tutor and a tutee. Those results are summarized in the graphs below.



Appendix B1: Likelihood to use app as a tutee



Appendix B2: Likelihood to use app as a tutor

Appendix C: Code Samples

```
import React, { Component } from "react";

// All React classes extend from Component
class MyClass extends Component {
  // Initialize state that will refer to this component
  state = {
    blah: data,
    isState: true,
    isExample: true
  };

  // Return the UI chunk to the page
  render() {
    <div>
      <h1>Hi There</h1>
    </div>;
  }
}

export default MyClass;
```

Appendix C1.1: React Class Component

```
import React, { Component } from 'react';      You, February 12th, 2019 12:29am • Email confirmation working in backend, next block
import { Link } from 'react-router-dom';

// Load UI components
import Grid from '@material-ui/core/Grid';
import Paper from '@material-ui/core/Paper';
import Card from '@material-ui/core/Card';
import CardMedia from '@material-ui/core/CardMedia';
import ConfirmImg from '../../images/confirmed-text.jpg';
import axios from 'axios';

You, April 5th, 2019 5:56pm | 2 authors (You and others)
class UserConfirm extends Component {
    // When the component renders get the id from the URL and the backend will match it to a user in the database if it exists
    componentDidMount = () => {
        const { id } = this.props.match.params;
        axios.get(`/email/confirm/${id}`)
        .then(res => res.json())
        .catch(err => console.error(err));
    }

    // Return the UI chunk to the page
    render() {
        return (
            <div className="padding20">
                <Grid container spacing={24} justify="center">
                    <Grid item xs={12} sm={8} md={6}>
                        <Paper>
                        <Card component={Link} to={'/login'}>
                                <CardMedia
                                component="img"
                                alt="account confirmed"
                                height="auto"
                                image={ConfirmImg}
                                />
                        </Card>
                        </Paper>
                    </Grid>
                </Grid>
            </div>
        )
    }
}

export default UserConfirm;
```

Appendix C1.2: React Class Component Example from Codebase

```
import React from "react";

// Pure function that returns the UI chunk, without needing state, good for static data
const myFunc = () => (
  <div>
    <h1>Hi There</h1>
  </div>
);

export default myFunc;
```

Appendix C2.1: React Functional Component

```jsx
import React from 'react'
import { Route, Redirect } from 'react-router-dom'
import { connect } from 'react-redux'
import PropTypes from 'prop-types'

// Route protected by auth, redirect when user logs out or when auth is invalid
const PrivateRoute = ({ component: Component, auth, ...rest}) => (
    <Route
        {...rest}
        render = {props => auth.isAuthenticated === true ?
        <Component {...props} /> : <Redirect to="/login" />
        }
    />
)

// Props checking (like type checking for our components)
PrivateRoute.propTypes = {
    auth: PropTypes.object.isRequired
}

// Connect our global Redux auth state to be used by this component
const mapStateToProps = state => ({
    auth: state.auth
})

export default connect(mapStateToProps)(PrivateRoute);
```

Appendix C2.2: React Functional Component Example from Codebase

```jsx
// JWT auth snippet
const JwtStrategy = require("passport-jwt").Strategy,
  ExtractJwt = require("passport-jwt").ExtractJwt;
const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
// Session secret (stored as environment variable, not hard coded)
opts.secretOrKey = "secret";

// Configure Passport to use JWT authentication
passport.use(
    new JwtStrategy(opts, (jwt_payload, done) => {
        User.findById(jwt_payload.id)
            .then(user => {
                // Return user data if found in database
                if (user) return done(null, user);
                return done(null, false);
            })
            .catch(err => console.error(err));
    });
```

Appendix C3: JWT Authentication Strategy