Team Outlaws

Final Report

Project Sponsor and Mentor:

Dr. Eck Doerry

Team Members:

Quinn Melssen

Liam Scholl

Max Mosier

Dakota Battle

May 2nd, 2022

Table of Contents

1 Introduction

As Agile programming practices continue to take the tech industry by storm, the importance of small teams in real world engineering workplaces is quickly increasing. According to Goremotely.net, over 71% of tech companies either already use, or are in the process of adopting agile methods, where small, flexible and cross-functional teams play the central role. The prevalence of small team workgroups in the professional world has made some wonder: why are more engineering classes in higher education not team-based, to provide specific training in small team projects? A main reason for this is the difficulty for faculty to manage and maintain the teams involved in such an undertaking.

Our client Dr. Doerry has spent the last 15 years working to perfect the Northern Arizona University's Computer Science Capstone Program, and as such has dealt with many of these difficulties first hand. Every year Dr. Doerry must painstakingly gather and communicate with enough clients to provide projects for the year. This process consists of hours of back-and-forth emails between many different potential clients to develop appropriate project proposals. Once the projects have been gathered and finalized, students are expected to review them and submit their preferences which are then used to assign them to their respective projects, along with other information such as their GPA. This process too, requires a high amount of hands on effort that could be streamlined by a successful technology. Once the projects have a team, Dr. Doerry will then be responsible for running the capstone course and managing the collection of various assignments. In summary, there are three primary phases of this process, each involving large amounts of hands-on effort:

- **Gathering projects** - Involves reaching out to dozens of potential clients, exchanging hundreds of emails, and keeping track of the varying stages of each potential project.
- **Forming teams** - Involves taking in preferences, GPAs, and other information about each student by manually, then inputting all relevant information into an algorithm, and forming teams.
- **Executing class** - Involves using several different modes of communication to run the team project course, including email, websites, and verbal/written communication.

While this process ultimately leads to a successful capstone experience, Dr. Doerry struggles with its inefficiency; this has only been amplified as he has attempted to split leadership of the course with another NAU Computer Science professor, Dr. Michael Leverington.

During this transitional process, Dr. Doerry has realized that his current solution could be streamlined both for himself and future instructors, as well as anyone who needs to run a team-based course or project. TeamBandit will act as a portal to do just this, breaking each of the problem areas outlined above into modules that will lower the effort involved with each step. This document will detail the process from start to finish in order to help interested parties get a broad look at the product as a whole.

2 Process Overview

In order to create TeamBandit effectively the team needed some way to keep track of what people were working on as well as a version control management system. We ended up going with GitHub for our version control system and utilized its KanBan boards to keep track of tasks that needed to be completed. Visual Studio Code was also the editor utilized by the team. This code editor offered valuable extensions for interacting with React, our front end, and our backend frameworks. After ensuring that the team was using similar environments for development to reduce one-off errors, we broke the development of the web application into parts.

Max Mosier ended up taking on a structural role for the application, building a majority of the foundation for the application to make it easier for others to start adding features. This consisted of setting up the page routing, user routing, and the backend express routes to the database application. With this skeleton built, the rest of the team was able to implement their modules.

Liam Scholl took the role of heading specialized database interactions. One of the primary features implemented by Liam was the uploading and storing of files onto our application and database to be served back to users. This feature was utilized by many other modules in the application.

Quinn Melssen took it upon himself to develop the email hub where users were able to view emails scraped from their inbox in order to keep them separate from non-course related correspondence. This entailed covering many edge cases as emails are formatted differently by each mail provider.

Dakota Battle focused on making sure commits and pull requests were merged smoothly and remained bugfree. He also worked on UX, ensuring the application's aesthetics were satisfactory, and most importantly uniform.

Our development process consisted of meeting weekly as a team to discuss what each member would need to accomplish for the week. This was to help avoid merge conflicts in the Github repository as we were able to prioritize working on different files and features.

3 Requirements

In order to accurately know what we wanted to accomplish in the creation of our application, our team began a requirements acquisition phase early on. This consisted of us meeting weekly as a team, and with our client to refine previous requirements and to discuss new ones. We wanted to identify functional requirements, performance requirements, and environmental requirements. A more detailed description of these requirements includes:

- Functional Requirements - A specification defining what the product should be able to do. As an example, what functions the product should be able to carry out for a user.
- Performance Requirements - Specification of criteria that is used to judge the operation of the system from a user perspective. For example, how long it takes a user to perform an operation while utilizing one or multiple functions of the product.
- Environmental Requirements - Also referred to as environmental constraints, these are any restrictions on the product related to which hardware or software to use defined by the client.

For the web application, functional and performance requirements, as well as environmental constraints have been identified through the process of requirements acquisition. In total we identified 82 functional requirements, 25 performance requirements and no environmental requirements. Some of the key requirements we identified include:

- User Accounts: allow users to create accounts to save their information
- Email Hub: help keep track of a mass amount of emails from many clients
- Course Management: provide a tool for adding courses and managing the content inside
- Team Assignment: make it easier for assigning students to teams through the application
- Pre-Generated Team Website: automatically generate team websites, containing details about a pre-existing project

With the requirements identified our team needed to figure out the architecture of our application and how we were going to go about building it. We will discuss the architecture we ended up going with in the next section.

4 Architecture and Implementation

TeamBandit is made up of several distinct modules with each serving a unique purpose. These modules interact with each other in several ways to produce the desired application functionality; planning the interactions between each module helped us to modularize the system into manageable components. Our system’s overall architecture is displayed in the diagram below.

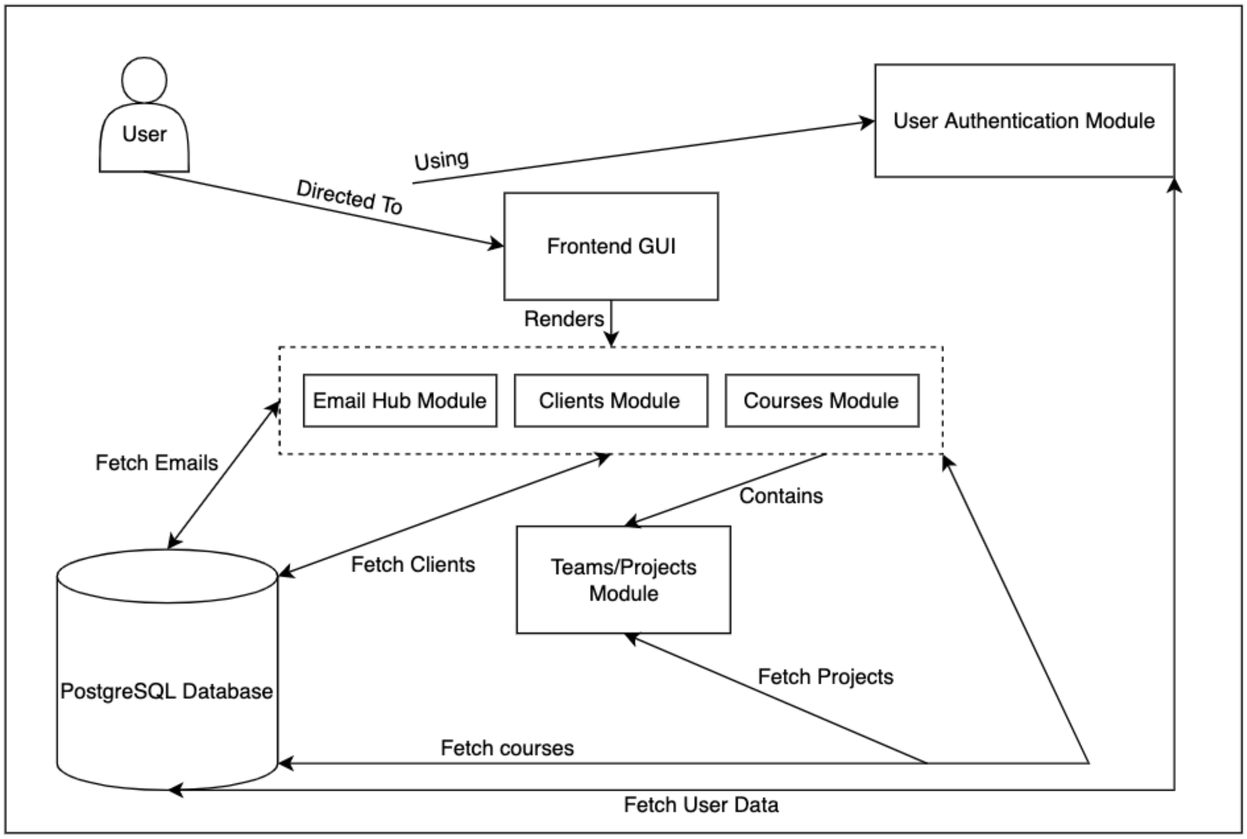


Figure 4.1 *High level architecture of the TeamBandit web application*

From the figure above, we can see that the web application does in fact have several modules that interact with each other seamlessly throughout the use of the application. Before each of these modules is explained in detail, the technology stack that we chose will first be briefly explored as it is important to understand how our technologies can assist us detail each module.

Our technology stack primarily consists of three main technologies that help the architectural modules interact with each other. These technologies are React, Express, and PostgreSQL. The web application also utilized smaller supporting frameworks to complete the functionality of each module.

React

- Although referred to as a front end framework, React is a large JavaScript library for building user interfaces for applications and is supported by Meta.
- Proven efficiency in creating web applications with strong visual appeal. Used by Spotify, Amazon, NASA, and Netflix.
- Utilizes JavaScript functions to return HTML components. This is exactly the type of modularity we were looking for in a front-end framework, as we can break up our web application into reusable components.

Express

- A back end web framework available through Node, a popular runtime environment for JavaScript.
- Provides a robust set of features to allow us to create an API to be used across the application in all of the web application's modules. This allows the web application to create, read, update, and delete data from the database upon user interaction.

PostgreSQL

- An open-source object-relational database management system to store all of the web application's data.
- Perfect for TeamBandit as it can take advantage of relational database tables to connect data thus, easily connecting architectural modules.

Supporting Frameworks

- Bcrypt - A library to help hash passwords.
- Material UI - Provides a library of foundational components for front end design, enabling the ability to develop React applications faster.
- React Router - A standard library for routing in React.
- JSON Web Token (JWT) - Generate tokens to authenticate users and ensure that only these authenticated users can make changes on the application.

Now that our technology stack has been briefly detailed, the responsibilities of each architectural module will now be explained.

4.1 User Authentication Module

The application is account-based and access to various elements of the application depend on the account status, so it is essential for the system to not only be capable of identifying a user, but also to accurately authenticate that identity to ensure a user is who they say they are. Each user is identified by an email address and a correct password, where both are set during the sign up process by the user. The user is then associated with a user id that is stored in the database, which assists in the creation of an authentication token for the user.

A: Description of Responsibilities

This module is key in making sure that our client's, as well as their students' information is correctly stored and pulled from the database. In order to achieve this, this module will have to take full advantage of all of the features that our technology stack offers.

- **PostgreSQL** acts as our database management system. Here, we will be able to store information about the users such as their email, name and password.
- **Express** allows us to set up ‘routes’ that will act as gateways to getting information about users and storing information about the users into the database.
- **React** helps us build better UI and UX experiences for our users when they sign in to or register an account in the application.
- **Bcrypt** allows us to encrypt user passwords for security purposes. This will allow the user's password to be hidden from others.
- **JWT** allows us to cache the users information locally on the user’s browser so that it allows them to stay signed into our application.

With the above technologies, a single sign-in/registration page where users can fill out a form of information to either sign up for or sign into our application was created. Once signed in, the user will be able to access the TeamBandit web application and any information about their account. This is done by utilizing the JWT library to create ‘tokens’ that identify who the user is.

B: Route Diagram

Below is the API for TeamBandit user authentication. It includes express routes for user registration, sign in, and verification. The verification and authorization routes essentially act as middlewares for the other routes to ensure that if someone is trying to view or make a change to the application, they are an identified TeamBandit user.

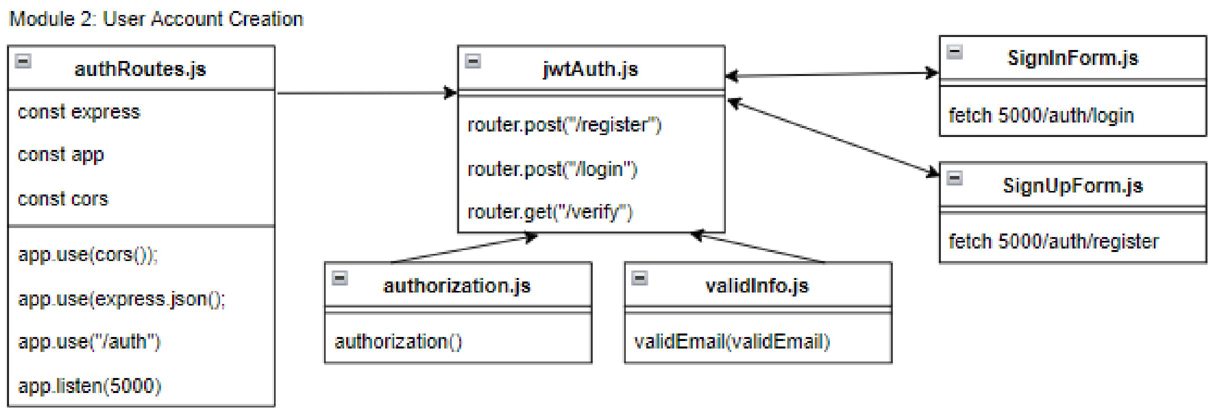


Figure 4.1B: Express route diagram for the User Authentication module.

Note that from this point on, all route diagrams for each module will follow a similar format to the above diagram and will contain the API calls for that diagram.

C: Illustrative Examples

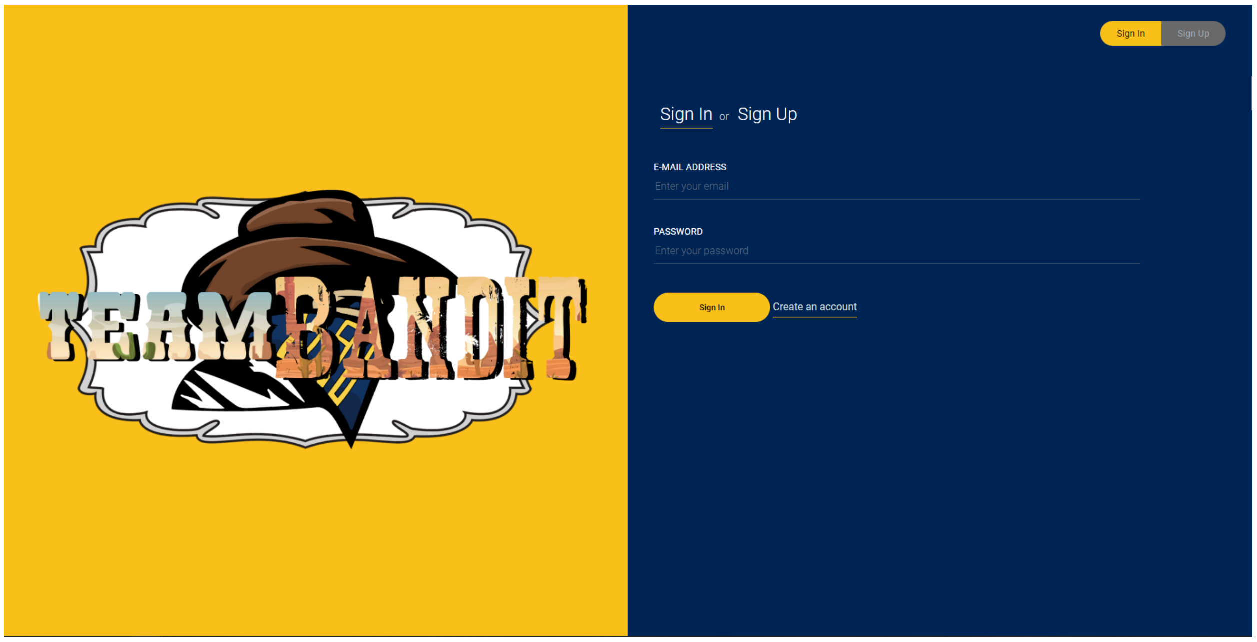


Figure 4.2C: The TeamBandit sign-in/sign-up page for users.

4.2 Clients Module

A course organizer will have the job of keeping track of multiple clients and their corresponding projects. This can get unorganized quickly, so to alleviate this, the clients module consists of a table listing the clients and their proposed projects. Contact information will also be available for each client, and an organizer has the ability to add, delete, and edit the details of a selected client. This information is stored and retrieved from our database system and be displayed using our frontend framework. These can be sorted in various ways, enabling further organization of the project clients. The information stored in this table is essential for communication between organizer and client in the email hub module.

A: Description of Responsibilities

This module is key in making sure that the course organizer can have an organized view of the project clients and have the ability to add, edit, and delete any information for a particular client. In order to achieve this, this module takes full advantage of many of the features that our technology stack offers. These technologies include PostgreSQL, Express, React.

- Using **PostgreSQL,** we store and pull information about the clients such as their email, name and password.

- **Express** allows us to set up ‘routes’ that will act as gateways to getting information about clients and storing information about the clients into the database.
- **React** helps us build better UI and UX experiences for a course organizer when they view the clients table in the application.

With the above technologies, a centralized location of project clients that is easily viewable by the course organizer was created.

B: Route Diagram

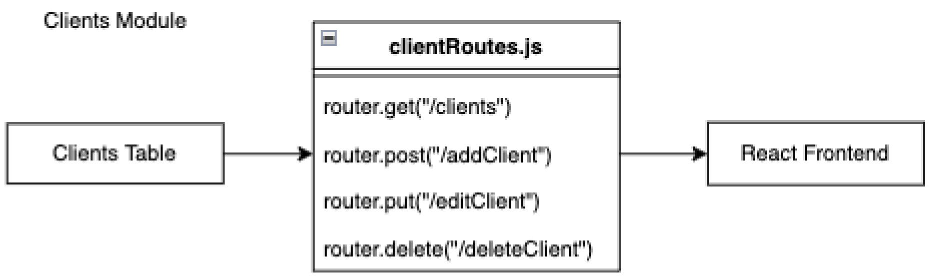
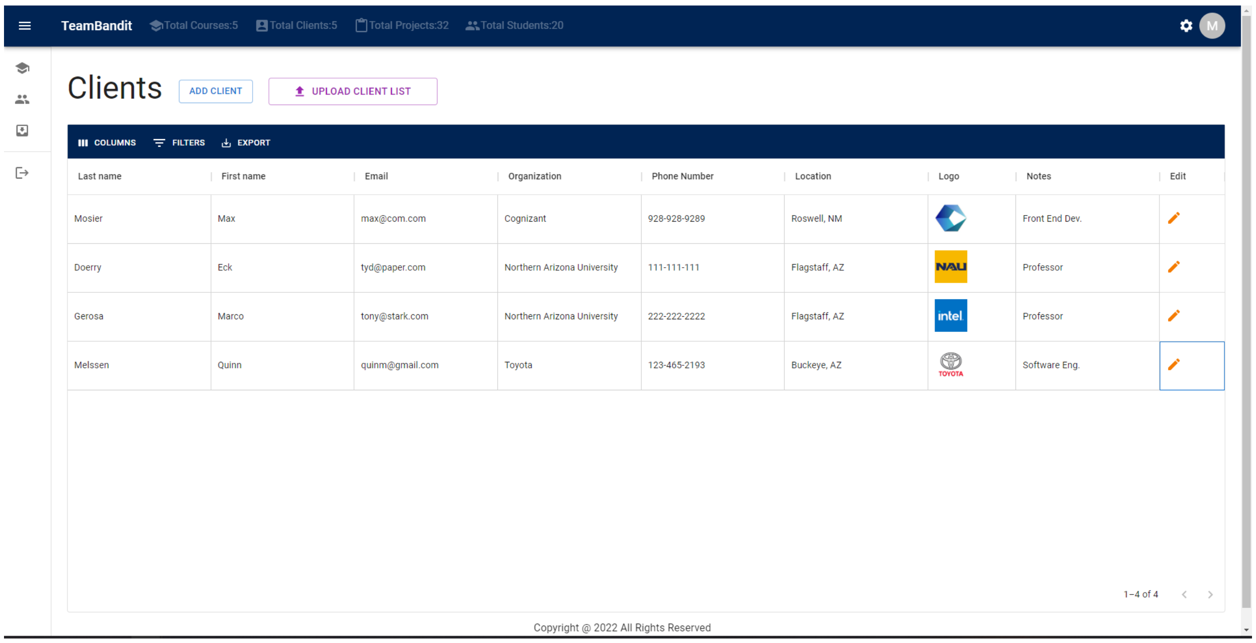

Clients Module
Clients Table
clientRoutes.js
router.get("/clients")
router.post("/addClient")
router.put("/editClient")
router.delete("/deleteClient")
React Frontend

Figure 4.2B: Express route diagram for the Clients module.

C: Illustrative Examples



Clients

Last name	First name	Email	Organization	Phone Number	Location	Logo	Notes	Edit
Mosier	Max	max@com.com	Cognizant	928-928-9289	Roswell, NM		Front End Dev.	
Doerry	Eck	tyd@paper.com	Northern Arizona University	111-111-111	Flagstaff, AZ	NAU	Professor	
Gerosa	Marco	tony@stark.com	Northern Arizona University	222-222-2222	Flagstaff, AZ	intel	Professor	
Melssen	Quinn	quinm@gmail.com	Toyota	123-465-2193	Buckeye, AZ	TOYOTA	Software Eng.	

Copyright @ 2022 All Rights Reserved

Figure 4.2C: A view of the Clients table in TeamBandit.

4.3 Email Hub Module

Below, a flow diagram of our client's current work flow for this module is displayed.

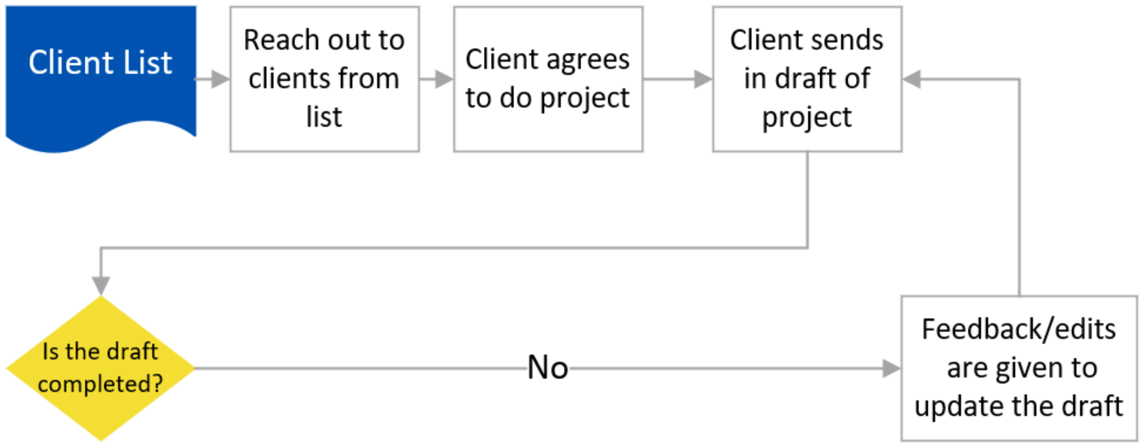

Client List
Reach out to clients from list
Client agrees to do project
Client sends in draft of project
Is the draft completed?
No
Feedback/edits are given to update the draft

Figure 4.3: Diagram showing Dr. Doerry's envisioned workflow utilizing TeamBandit to communicate with clients.

Our client receives constant emails from different clients in different email threads. Keeping track of all of these is an arduous process for our client. In order to fix this current process, email chains are created for faculty to see all of the emails organized by clients in one place on the web application.

TeamBandit's email hub is accessible from the initial landing page upon logging in and simply displays emails to and from a course organizer and the client for a project. The organizer can view all of these emails in a conversation-like format where their sent and received messages are easily differentiated. An external email server is used to handle the overhead of all communications, and a script simply copies the relevant email information to the hub and organizes it accordingly. Rather than truly performing as a server for communications, it acts as an optimized display to provide transported information from elsewhere, the reason being the convenience of messages being available within the application itself. The information flow of the data is as follows: The script listens for activity on a predetermined interval, so when a relevant message is received in the organizer's inbox or a message meeting specific constraints is sent out by the user, the script copies the email information to the database so it can then be displayed on the email hub page.

A: Description of Responsibilities

This module focuses on the early stages of our client's capstone process. It centers on features regarding clients as well as email messages between clients. These features include getting emails associated with capstone clients and storing them into a database and pulling those messages from the database and displaying them on our web application. In order to accomplish these tasks on the web application, we will need to use the following technologies:

- **PostgreSQL** acts as the database management system for the web application. Here, we will be able to store information about clients along with the messages associated with them.
- **Express** allows us to set up 'routes' that will act as gateways to getting information about clients and messages and store that information into the database.
- **React** helps us build a better UI and UX experience for our users related to how they will view the clients and messages as well as their experience creating new clients.
- **Python** has a built-in library called email.parser which allows us to pull information from emails if our created email is carbon copied (CC'd) in the email.

With these technologies, two separate locations are created, one where clients can be viewed, edited and created, and another where messages from clients can be displayed.

B: Route Diagram

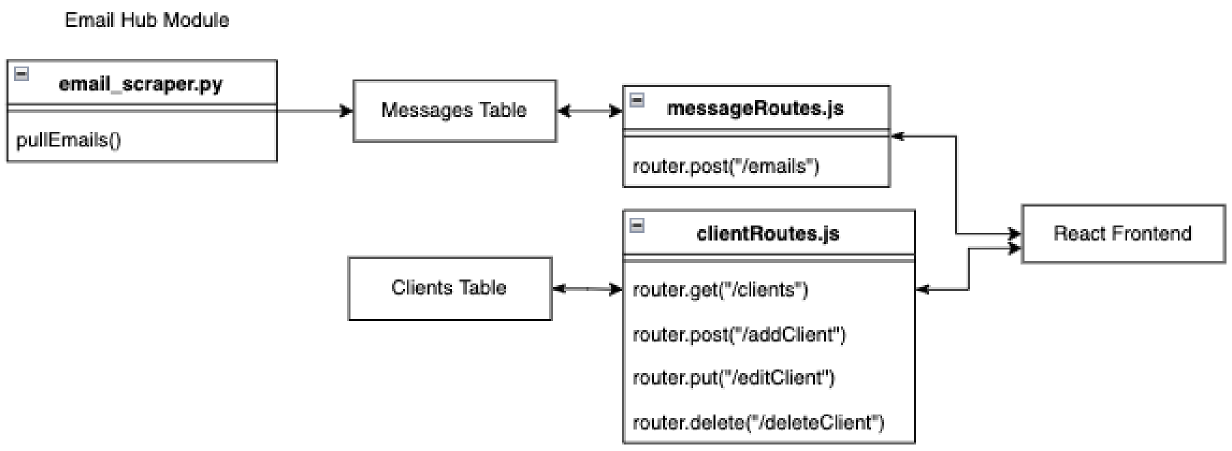
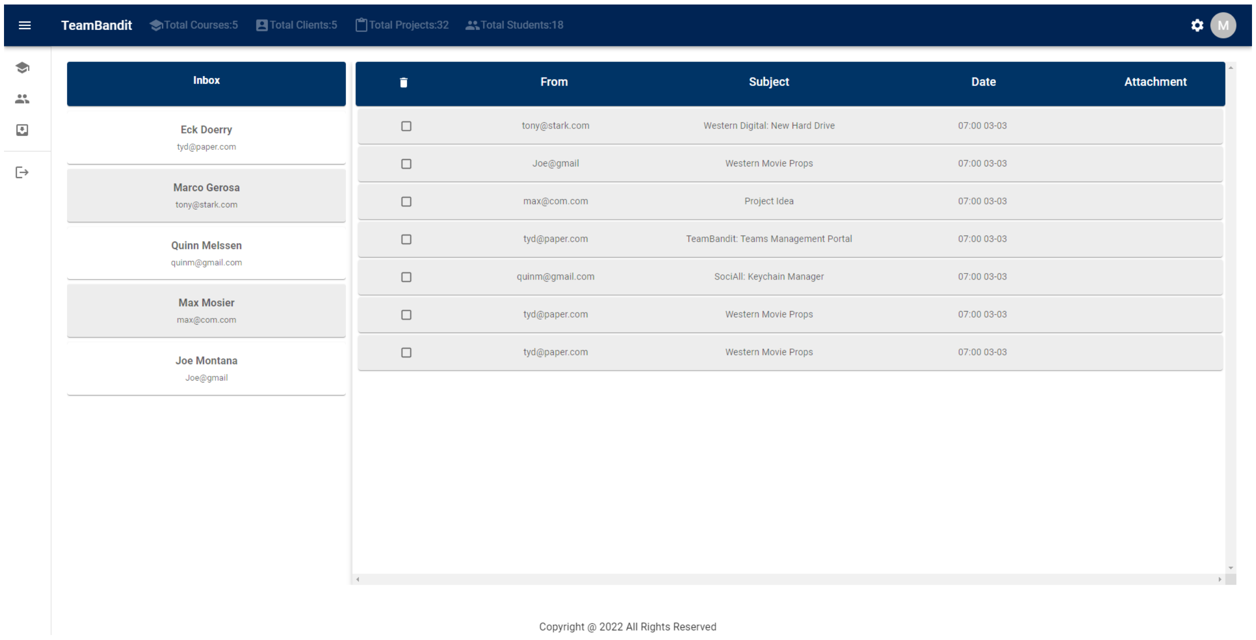


Figure 4.3B: Express route diagram for the Email Hub module.

C: Illustrative Examples

C: Illustrative Examples



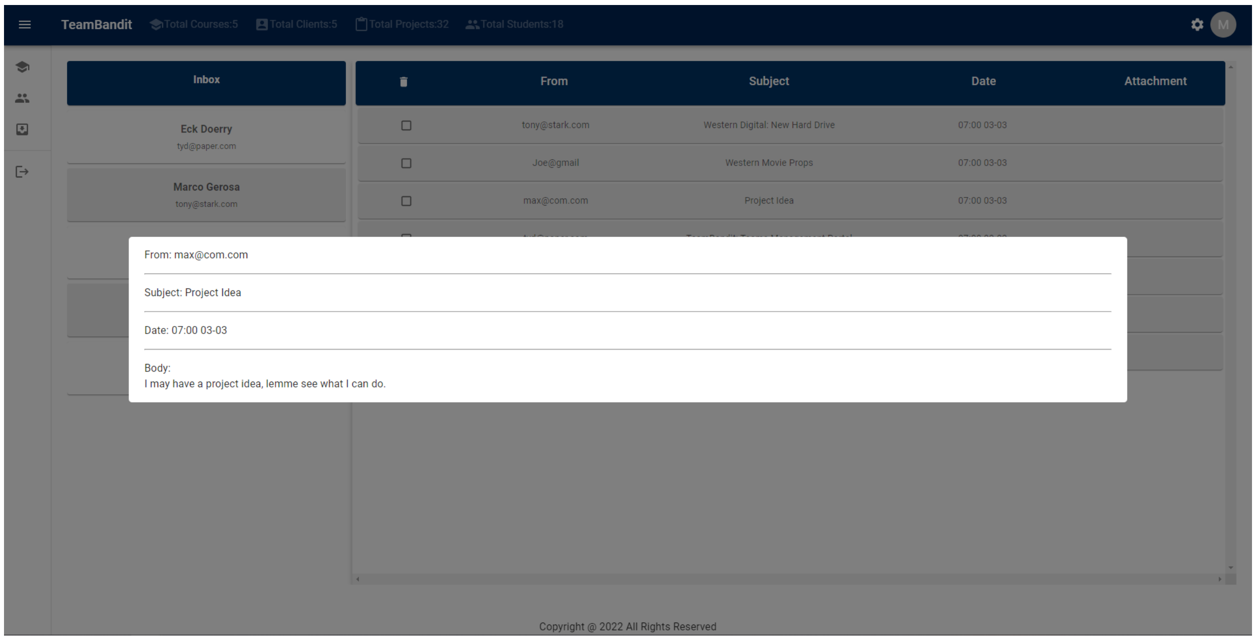
TeamBandit Total Courses:5 Total Clients:5 Total Projects:32 Total Students:18

Inbox
Eck Doerry tyd@paper.com
Marco Gerosa tony@stark.com
Quinn Melssen quinm@gmail.com
Max Mosier max@com.com
Joe Montana Joe@gmail

	From	Subject	Date	Attachment
☐	tony@stark.com	Western Digital: New Hard Drive	07:00 03-03	
☐	Joe@gmail	Western Movie Props	07:00 03-03	
☐	max@com.com	Project Idea	07:00 03-03	
☐	tyd@paper.com	TeamBandit: Teams Management Portal	07:00 03-03	
☐	quinm@gmail.com	SociAll: Keychain Manager	07:00 03-03	
☐	tyd@paper.com	Western Movie Props	07:00 03-03	
☐	tyd@paper.com	Western Movie Props	07:00 03-03	

Copyright @ 2022 All Rights Reserved

Figure 4.3C1: A view of the Email Hub in TeamBandit.



From: max@com.com

Subject: Project Idea

Date: 07:00 03-03

Body:
I may have a project idea, lemme see what I can do.

Figure 4.3C2: A modal popup of an email once it is clicked.

4.4 Courses Module

The courses module provides the functionality necessary to create, remove, edit, and otherwise manage courses in progress or the data contained therein. From here, the course is first created and supplied necessary information for its initialization. Within that initialized course, the course itself can be archived, individual users can be added or removed from the course, course descriptor information can be edited, and deliverables can be created for student submissions.

A: Description of Responsibilities

The courses module is responsible for a multitude of actions that will be carried out by the course organizer. Before this module fully commences, all projects will have been created, all students will have their accounts and be assigned to their courses, and all students will be assigned to their projects. The actions that the course organizer may perform include:

- Creating and updating a course
- Deleting a course
- Adding, removing, and updating users from within a course
- Updating course information
- Creating deliverables for students to submit directly on the web application

The above actions that an organizer can perform will ensure that a course can be initialized successfully.

B: Route Diagram

In order to properly execute the courses module, we broke it down into separate API files corresponding to each module within a course. Note that the diagram below only shows two modules. The modules consist of projects, schedules, assignments, and students. These modules, except for the assignments module, will be detailed in the next section, and follow the same create, read, update, and delete structure in their respective files.

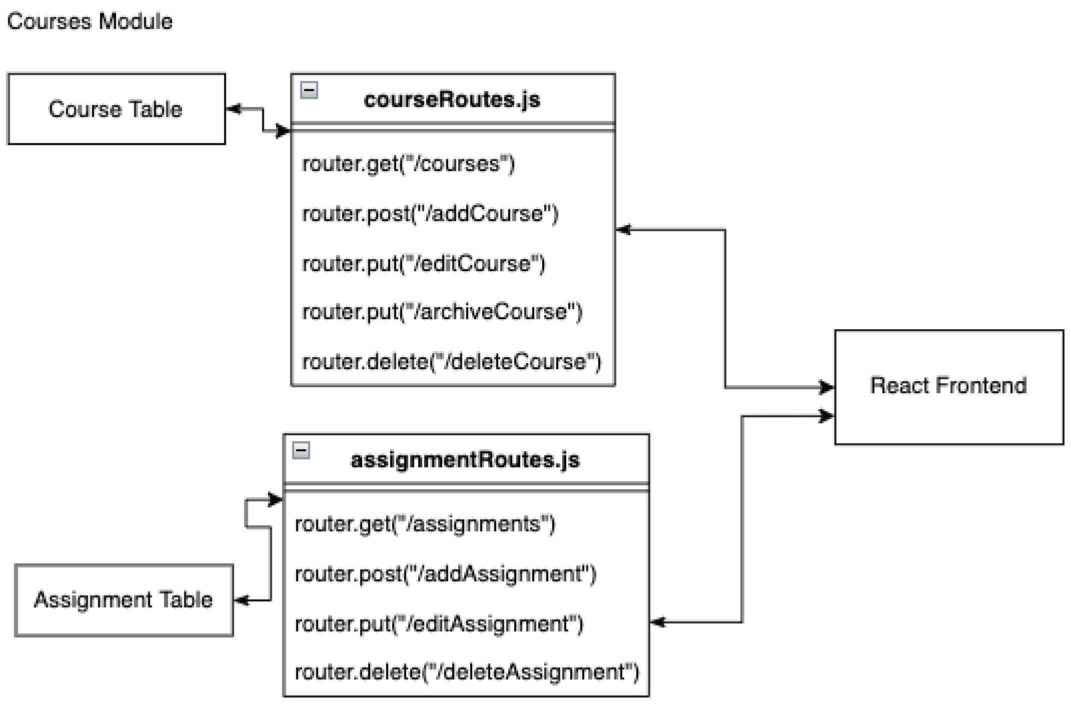
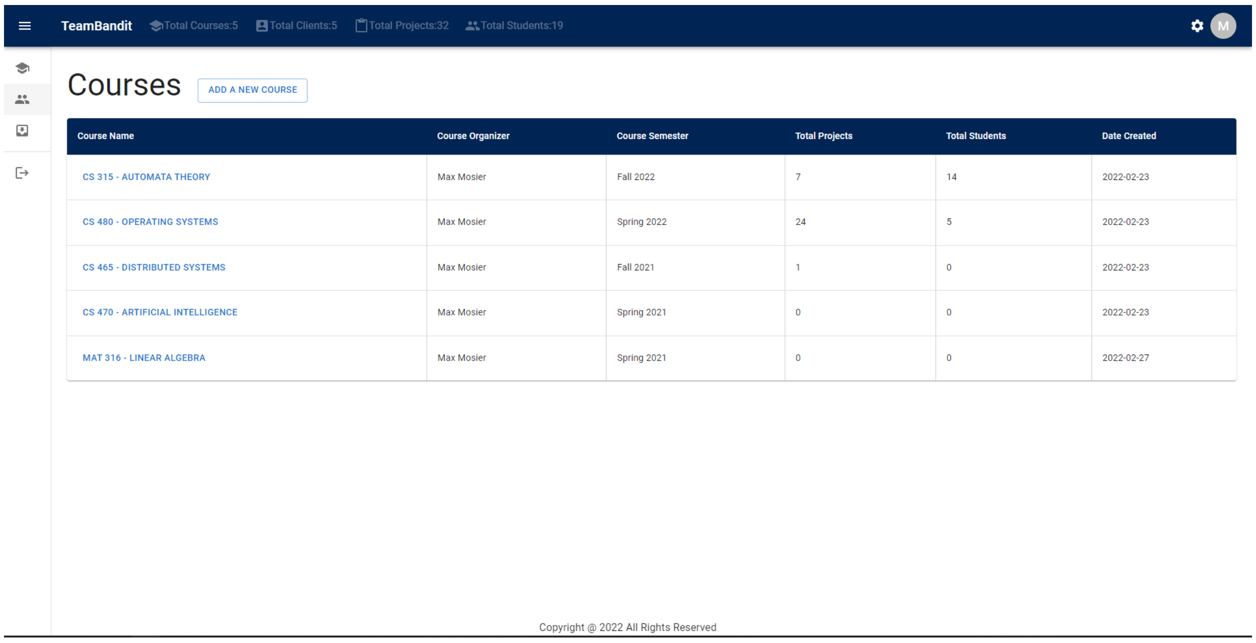

Courses Module
Course Table
courseRoutes.js
router.get("/courses")
router.post("/addCourse")
router.put("/editCourse")
router.put("/archiveCourse")
router.delete("/deleteCourse")
React Frontend
assignmentRoutes.js
router.get("/assignments")
router.post("/addAssignment")
router.put("/editAssignment")
router.delete("/deleteAssignment")
Assignment Table

Figure 4.4B: Express route diagram for the Courses module.

C: Illustrative Examples



Courses

Course Name	Course Organizer	Course Semester	Total Projects	Total Students	Date Created
CS 315 - AUTOMATA THEORY	Max Mosier	Fall 2022	7	14	2022-02-23
CS 480 - OPERATING SYSTEMS	Max Mosier	Spring 2022	24	5	2022-02-23
CS 465 - DISTRIBUTED SYSTEMS	Max Mosier	Fall 2021	1	0	2022-02-23
CS 470 - ARTIFICIAL INTELLIGENCE	Max Mosier	Spring 2021	0	0	2022-02-23
MAT 316 - LINEAR ALGEBRA	Max Mosier	Spring 2021	0	0	2022-02-27

Copyright @ 2022 All Rights Reserved

Figure 4.4C: The courses page where a user can view and enter all of their courses.

4.5 Teams/Projects Module

The teams/projects module is within the courses module as there are team assignments and project creations within a course. Individual students are assigned teams, and one team pertains to a project. After a team is assigned to a project, the organizer has the ability to add or remove students from a team. Members of a team can submit deliverables within a course, and the application will submit it on behalf of the team. The teams/projects module communicates with the course module to associate teams/projects with a specific course to be used for deliverables.

A: Description of Responsibilities

The teams/projects module will ensure that a course organizer has the ability to create projects within a course and assign users, such as students, to a project. This module will be responsible for collecting the project preferences of all students. These preferences are filled out by the students and will then be displayed to the course organizer at the time of assigning teams on a team assignments page located within each course. This page will consist of all student names along with some corresponding information which includes, but is not limited to:

- Email address
- University User ID
- GPA
- Top five project preferences

Along with student information being displayed, general information such as project numbers and the amount of students currently assigned to a team will be displayed to the organizer.

When creating a project, a course organizer will be able to assign students and associate clients to that project all in one place.

In order to accomplish these tasks, we will need to utilize these technologies:

- **PostgreSQL** acts as our database management system. Here, we will be able to store information about students and projects.
- **Express** allows us to set up ‘routes’ that will act as gateways to getting information about students and projects and store that information into the database.
- **React** helps us build better UI and UX experiences for the course organizer when they view all students and projects in one centralized location.

B: Route Diagram

This module is by far the most complex, so it contains the largest API that the web application uses.

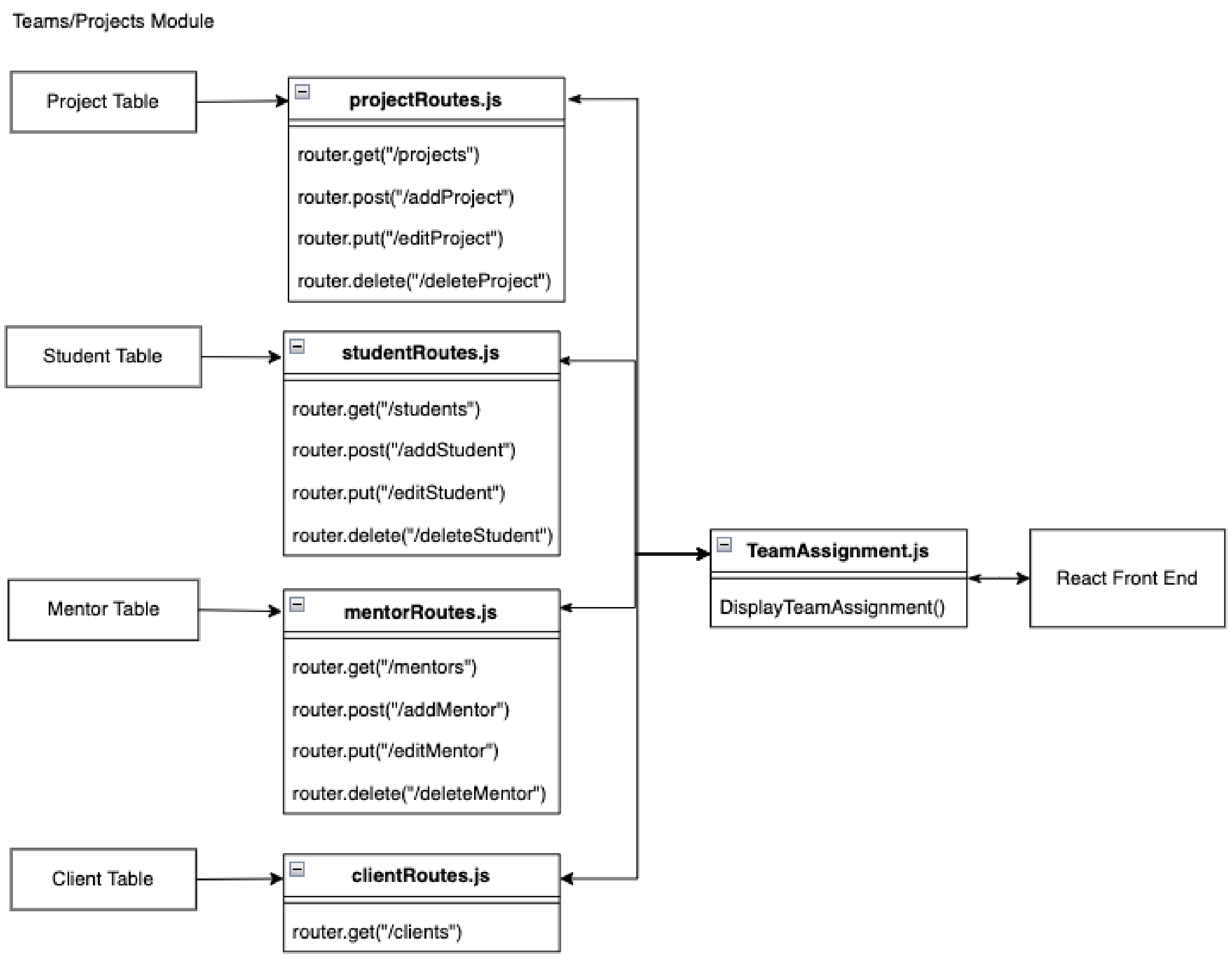
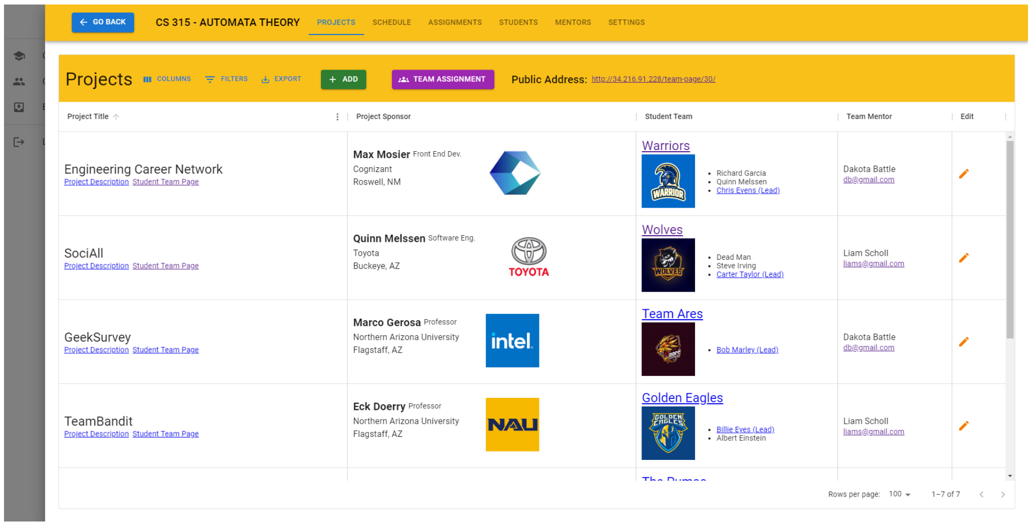


Figure 4.5B: Express route diagram for the Teams/Projects module.

C: Illustrative Examples



Public Address: http://34.216.91.228/team-page/30/

Project Title	Project Sponsor	Student Team	Team Mentor	Edit
Engineering Career Network Project Description Student Team Page	**Max Mosier** Front End Dev. Cognizant Roswell, NM	Warriors • Richard Garcia • Quinn Melssen • Chris Evens (Lead)	Dakota Battle db@gmail.com	
SociAll Project Description Student Team Page	**Quinn Melssen** Software Eng. Toyota Buckeye, AZ	Wolves • Dead Man • Steve Irving • Carter Taylor (Lead)	Liam Scholl liams@gmail.com	
GeekSurvey Project Description Student Team Page	**Marco Gerosa** Professor Northern Arizona University Flagstaff, AZ	Team Ares • Bob Marley (Lead)	Dakota Battle db@gmail.com	
TeamBandit Project Description Student Team Page	**Eck Doerry** Professor Northern Arizona University Flagstaff, AZ	Golden Eagles • Billie Eyes (Lead) • Albert Einstein	Liam Scholl liams@gmail.com	

Rows per page: 100 1–7 of 7

Figure 4.5C: The final projects page after completion of project creation and team assignments.

4.6 Planned vs. Built Application

For the most part, the development team was able to build all features of the planned application in the final built application, with an exception of two features. These two features are a unique TeamBandit view for a third type of user: mentors, and the ability to archive a course.

Throughout the planning process, we assumed that the interaction between a user and the application would be radically different between organizers, students, and mentors. However, as we built up the features of the application, we realized that mentors will essentially be students with the ability to possibly leave feedback, a feature that was not discussed in the planning process. While mentors were an important user view to our client at first, it seemed that organizers and students took precedence over them.

The ability to archive a course was always something that was meant to be implemented, but the development of TeamBandit always contained the sudden implementation of new features. As a result, database changes were frequent, and if the team had spent time on implementing course archiving before most of the baseline features were implemented, the archiving process would have to be constantly revisited for every new major feature. The team did not see this as ideal, and we focused on implementing new course features over an archive system. Essentially, course archiving was not built into the application as there was no time to implement it after the constant addition of larger application features.

Now that the overall architecture and implementation of TeamBandit has been explored, the results of testing the application can be described.

5 Testing

When conducting testing for TeamBandit, we identified three primary testing methods for identifying areas of refinement within the application. We conducted them as follows:

5.1 Unit Testing

Unit Testing was concerned with individually verifying proper functionality of each unit within the application. Two scales of measurement for units were conducted. These were technological units of the application and conceptual units. Technological Unit Testing involved splitting each use of different major technologies into units and conducting a series of tests pertaining to their associated functionality.

TeamBandit was split into the authentication module, data handling units for tables, and the database module. Testing of the authentication module simply ensured that our mode of authentication performed as expected during registration, login, and managing active user sessions. The technology of focus for this was the JWT tokens we used for maintaining existing sessions for a given user. Testing data handling units simply involves entering various variations of data in each table that stores content in the application. Any unexpected behavior was addressed as it was encountered. For database tests, we tested retrieval of data from each directly-accessible table from each page it was accessible from when logged into the app.

For the conceptual units, we separated the “Login”, “Registration”, “Data Manipulation”, “Course Management”, “Student Features”, and “Team Lead Features”. This was simple; each unit was isolated from its interactions with the rest of the application and was only provided needed resources for basic functionality. Then, typical actions using that module were performed. No notable issues were brought to light throughout either scope of this method of testing.

5.2 Integration Testing

After verifying proper functionality for each unit/module individually, we were able to conclude that any potential arising issues are unlikely to be caused by a sole module of the application. The next logical step was to combine the modules gradually to see how their performance changes in response to being integrated with other modules. We selected the following modules to perform integration testing on:

- User Authentication Module - Ensures that information is correctly stored and pulled from the database.
- Clients Module - the course organizer can have an organized view of the project clients and have the ability to add, edit, and delete any information for a particular client.
- Email Hub Module - Pulling emails associated with capstone clients, storing them into a database, pulling those messages from the database, and displaying them.
- Courses Module - Centralized location for all courses and their corresponding course information.
- Teams/Projects Module - Ensure that a course organizer has the ability to create projects within a course and assign users, such as students, to a project.

These are the foundation of the overall application are better visually detailed in Figure 4.1 in the previous section.

Effective integration between *all* of these various modules make up the complete application, and ensuring that each module works cohesively with each other module it directly interacts with is imperative in preventing unexpected behavior later down the road. In order to do this interaction–by-interaction we conducted a series of detailed predefined actions that made up a use case within each module.

5.3 End User Testing

With consideration toward the product's intended use, it was apparent early on that the user interface needed to be tested to identify any buttons, links, or functionality in the user's view that causes confusion or uncertainty when using TeamBandit. Our strategy to ensure accurate and relevant results was to gather a sample of test end users who have the same level of experience as our intended end users and have them test TeamBandit live with no prior experience in order to gather feedback. The procedure was as follows:

- Test end user is provided a laptop with TeamBandit running live on it
- Test end user is given a list of tasks to conduct, having never seen or used the application before
- While the test end user is using the application, a member of our team is present. The test end user "thinks aloud", describing their thought process at every step of their experience.
- The team member makes a note of anything considerable
- The test end user is given a survey for additional input following the demo.

This procedure was conducted for each test end user, and after concluding each demo session we analyzed the results to look for *patterns* of difficulty encountered by these test users. Our reasoning behind this is because if several test users are confused by a specific element of the user interface, it is likely that this element is unclear enough to confuse a considerable quantity of users and should be addressed accordingly.

Our sample of test end users consisted of 4 instructors with a background in organizing courses and 2 students. Further broken down, the sample of instructors included 2 instructors with experience organizing *team-based* courses and 2 instructors with experience organizing only typical classes. This enabled our sample to account for a wider range of potential organizer end users, as not all instructors interested in organizing team-based courses have experience doing so.

We identified a few areas that several test end users struggled with. Some of these pertained to operations outside of a course. For example, some of our organizer test users struggled to identify the location of the email hub and/or clients page, as they are accessible from a different navigation bar than course features are. Our response was to make this navigation bar more apparent to the user, especially when they first log in. As a result, this collapsible navigation bar starts out expanded so it is visually obvious upon login.

6 Project Timeline

TeamBandit's development cycle will take place over the course of the next few months and will utilize a continuous integration approach in order to get the product in user's hands as soon as possible. Development has already begun, starting in December with the creation and formatting of our AWS server, which is where we will be hosting our application. This included tasks such as downloading Node (server backend), React (frontend framework), PostgreSQL (database), as well as all of the necessary tools for development such as node package dependencies. Our envisioned project timeline moving forward is summarized in Fig. 5.0.

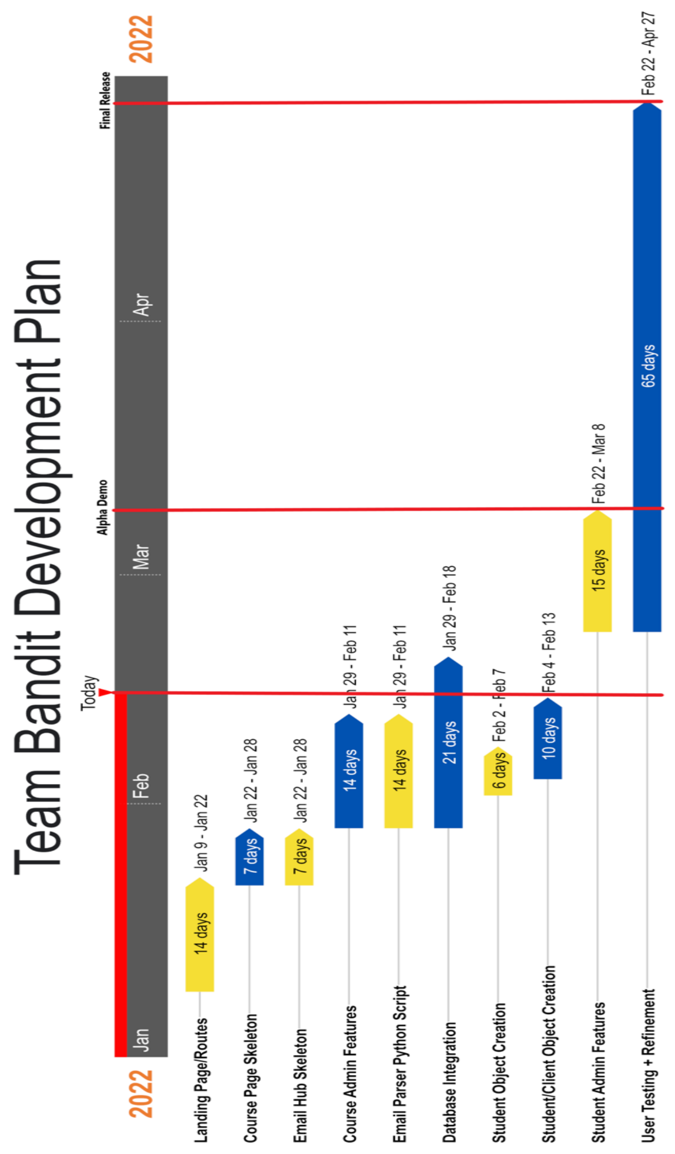


Figure 5.0: This Gantt chart shows key components in our development plan moving forward

As shown, each component is allotted a relatively short amount of time due to our minimal focus on the aesthetics of the project in the early phases of development. Our plan begins with setting up the routes for each of the pages of TeamBandit. Doing this early allows for each developer to work on their individual components without having to merge changes to the routing files, minimizing the amount of conflicts we will have to manage.

After routes have been established the development process will be broken down into modules, namely the course creation/execution module, the client acquisition/communication module, and the student creation module. Each of these modules work with one another to create the whole of TeamBandit, and as such each developer will maintain a strong understanding of the project as a whole.

To begin our implementation we start with loose skeletons of each module to give ourselves a foundation to work off of. This is a short process and should only take a few days as shown by the 7 day allotment. After these initial steps have been completed each developer will move into fleshing out the functionality of their respective portion.

Liam	Quinn	Dakota	Max
-Adding courses -Setting up course page/metadata -Student account route -Site settings	-Email parsing script -Displaying emails in hub -Sending emails to new users	-Homepage display -Status tracker -Adding clients -Client table	-Project creation Assigning students to project -Student logins/account creations

Fig 5.1: Table highlighting each developer's major priority, with more detail found below

The separation of concerns between each of these tasks allows our developers to work simultaneously while minimizing the needs for complex merging. As each of the pieces come together we will move into a more iterative process, garnering feedback from our client and quickly implementing these changes during our User/UX Testing phase throughout the end of the project's development cycle. This will be the majority of the focus during the month of March.

7 Future Work

While TeamBandit fulfills many of the requirements dictated by our client, there have been additions made over the course of the semester. Many of these features revolve around the expansion of the application to use cases other than that of our client, such as other team based organizations (i.e. sports teams, corporate teams, etc).

While we have kept such fluidity in mind throughout development, many suggestions came up that were not feasible to be implemented in our development. Some of these include:

- Dynamic terms
  - Allow organizers to input their own term names can then be used to sort classes in their course catalog (i.e. Spring, Fall, Summer, Q1, Q2, etc.)
- Dynamic student information
  - Currently the web application and team assignment module only can receive and show certain meta information pertaining to students (i.e. GPA)
  - In the future one should have the ability to decide what information they want to upload about their students (i.e. batting average for a baseball team)

There are also some large features of the website that were partially completed but could be improved, such as:

- Automatic team assignment
  - As of now the team assignment module works by the organizer manually seeing the meta information correlated to each student and placing them in the teams by hand
  - In the future an algorithm can be written to automate this sorting (i.e. N-Queens recursive sorting algorithm to create best outcome based off student preferences and GPA)
- Team website editor
  - Currently the team websites are standardized, the order and content of the websites are not changeable
  - In the future there may be a team website editor built into the application, similar to how WIX lets creators drag and drop

Overall, Team Outlaws has strived to meet the requirements highlighted in the original product page and has mostly succeeded. Moving forward work on the project would be mostly refinement based, building off of the modules already in place.

8 Conclusion

Upon completion of the implementation plan, each of these modules will assemble the discussed components into the fluid web application intended by the design. Connecting the course initialization, user account creation and authentication, and team assignment/management mechanisms together will allow the course organizer to effectively oversee the progression of several courses while maintaining organization and control.

This record of design established a concrete set of primary functional components and outlined the high-level architecture and overview to implement it, thereby laying out a template for modularizing each major element of functionality. These modules and their respective interfaces have been extensively elaborated upon, paying special attention to the specific roles each member of the development stack plays in the functionality of these modules as well as how these roles contribute to TeamBandit's overall control flow.

Tying together the features of each interface with its respective underlying implementation details provides a lower-level blueprint of how the application will actually be interacted with to not only the anticipated user/client, but also can be doubled as an explicit guide to the development of each discrete module's distinct feature requirements. The most valuable outcome provided by this design process is the explication of the ground-level specifics that had not been anticipated prior to the software development stage. With these specifics resolved and simplified in writing, the remainder of the development process essentially becomes translation to code.

These specific developments glue together the vision that guides the entirety of TeamBandit: Reflecting the importance of working together in a productive environment. For far too long, adequately preparing students for the world that awaits them was difficult because the real world is not easily mirrored in education. Courses at the university level actively struggle to provide an environment of learning that aligns with the indispensable need for collaborative progress in every industry- until now.

With the development of these details behind us, we're confident this application will not only support the course management of Dr. Doerry, but hundreds of course organizers across the world by heavily reducing the demands for maintaining classes focused around applying the content in a team setting. Even further, the coordinated application of skills will provide the students enrolled in those courses with an approach to learning that reflects reality.

9 Appendix A: Development Environment and Toolchain

9.1 Hardware

For the development of this application, our team used a combination of Windows and Apple machines (Desktops, Laptops, and Macbooks). There are no recommended tech specs as this application will run on lower end systems, it may just take a longer period of time to build.

9.2 Toolchain

Visual Studio Code was the primary development editor. This was a helpful code editor, although some would call it a full IDE, as it would notify you on what branch you were working on (in the bottom left corner), as well as provide some helpful extensions.

Recommend Extensions for Visual Studio Code:

TODO Highlight by Wayou Liu
TODO Highlight makes it incredibly easy to identify portions of code for your team. Simply putting TODO in a code comment highlights it, which brings people's attention directly to that line. With this we were able to notify your team of problems or solutions to code.

Python by Microsoft
When working with Python it's important to have nice color indicators in Visual Studio Code, this extension helps VSCode to know how to work with Python.

Prettier - Code Formatter by Prettier
It's important to have a standardized way to write your code as a team. With Prettier it's never been easier! Simply press CTRL + Shift + P in the Visual Studio Code editor then press 'Format document' then the document will automatically get formatted in an identical manner! Amazing!

JavaScript (ES6) code snippets by charalampos karypidis
JavaScript was a key coding language for our application, making up 95% of the project. This is an essential extension as it lets Visual Studio code know how to work with JavaScript.

HTML Snippets by Mohamed Abusaid
HTML is another essential part as you are utilizing HTML snippets in react. This will let Visual Studio code know how to work with JavaScript.

ES7+ React/Redux/React-Native Snippets by dsznajder
Since React is the primary framework of our application this is another essential framework. This framework will help you code in React easier as you can utilize the handy code snippets.

ENV by Jakka Prihatna
This extension helps with the two .env files we have in our application. We recommend this so that Visual Studio Code knows how to work with them.

Code Spell Checker by Street Side Software
The next three extensions are not mandatory but helpful. This one helps to make you aware of misspellings when it comes to variable and function names.

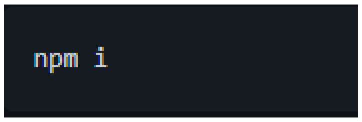
Auto Rename Tag by Jun Han
Another helpful extension. This extension allows you to rename both parts of a tag. For example if you have a <p> and a </p> tag, and you want to change it to a <div> /</div> tag you only have to change one as the other will get changed with this extension.

Auto Close Tag by Jun Han
Another handy extension. This automatically creates the closing tag for any tag you create. For example you will type <p> to make a paragraph tag and this extension will automatically generate a closing </p> tag.

9.3 Setup

The installation instructions for TeamBandit are detailed here:

Initial Setup

1. Clone the TeamBandit GitHub repository to your machine.
   https://github.com/QJMTech/TeamBandit
2. Install Node. This is used to run React and Express.
   https://nodejs.org/en/
3. In the terminal, navigate to the /website directory in your cloned TeamBandit repository.
4. Run the following command to install all of the Node dependencies:
   
   ```
   npm i
   ```
5. Install PostgreSQL to set up the database.
   https://www.postgresql.org/download/
6. In the terminal, navigate to the /website/server directory in your cloned repository. There is a file named “sqlCode.txt” in this directory. Copy the contents of this file and paste them into the PostgreSQL query tool then run the query. The database is now set up!
7. The /website and /website/server directories each have an .example.env file. Each contains comments and examples detailing how to set up your own .env files to set up connections to the database.

Initial Setup Complete!

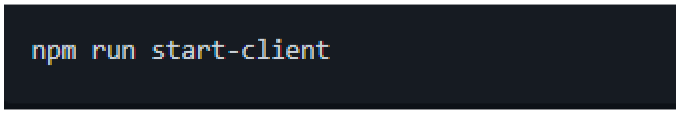
Run Guide

Windows

To run on windows you need to open *two* terminals (either through VSCodes terminal, powershell, or normal CMD terminal).

First Terminal

Navigate to the /website directory, then run:

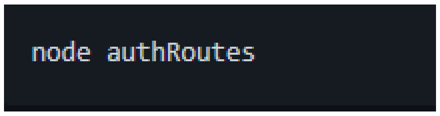


```
npm run start-client
```

This runs the React application on localhost.

Second Terminal

Windows does not execute commands simultaneously, so you need to navigate to the directory /website/server/middleware.
From there, run:

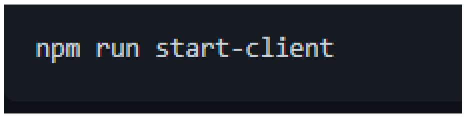


```
node authRoutes
```

This will set up the connection to the database.

MacOS/Linux

Navigate to /website and run:



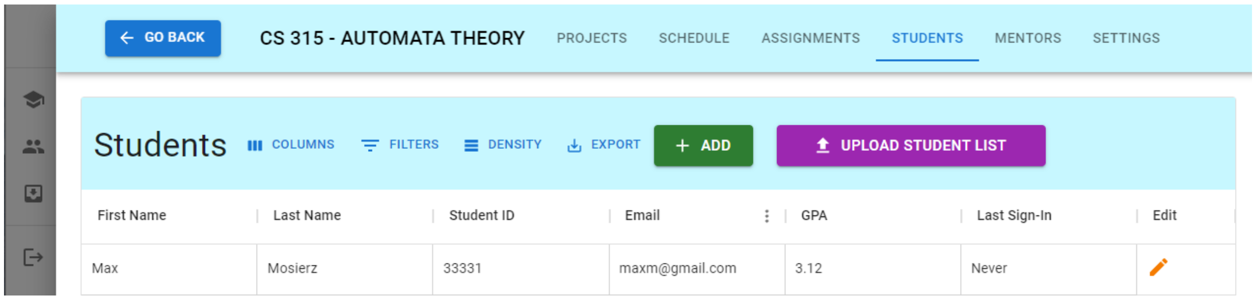
```
npm run start-client
```

You are done!

The application should be set up and running now. Congratulations!

9.4 Production Cycle

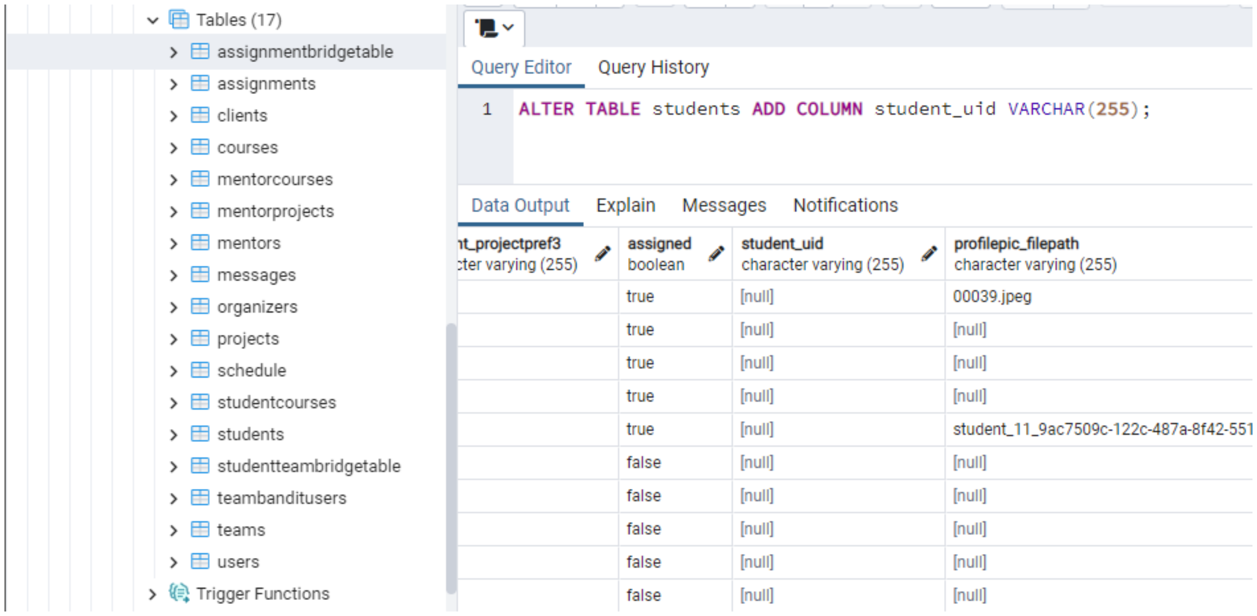
One of the primary things we do with the application is adding things to the PostgreSQL database and then grabbing things from the database. In order to give you an idea of how to do this I will walk you through a scenario.



First Name	Last Name	Student ID	Email	GPA	Last Sign-In	Edit
Max	Mosierz	33331	maxm@gmail.com	3.12	Never	

1. Say in our application we want to add another column to this students table. Firstly you would want to add this column to the PostgreSQL database. For the sake of this run through we will add a Student UID or User ID.

2. The tables are pretty straightforward for our database so we will simply run the ALTER TABLE command to add the new column to the students table.

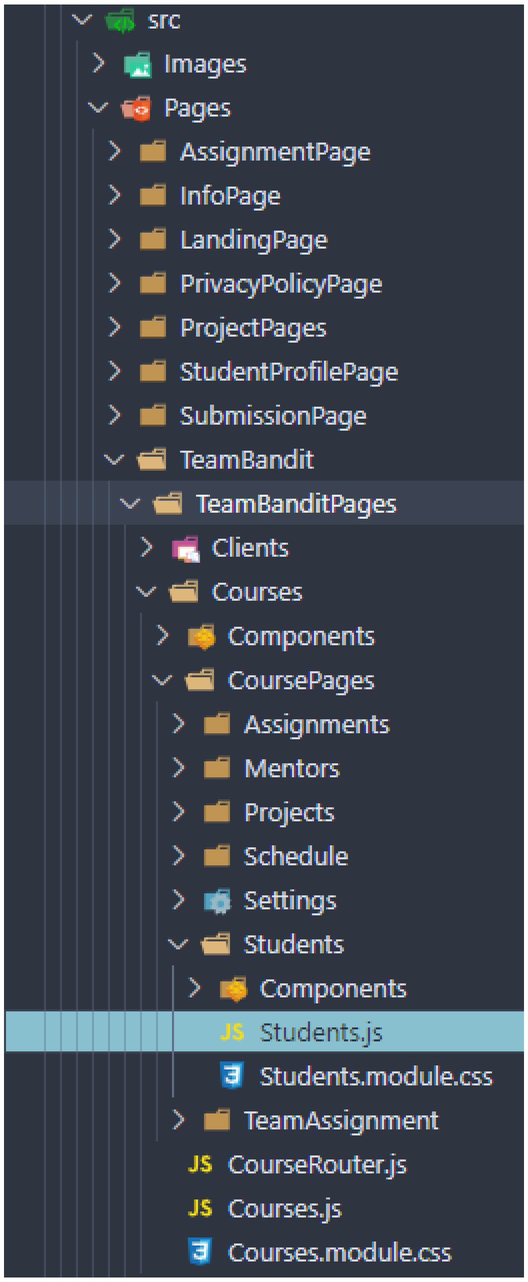


```
ALTER TABLE students ADD COLUMN student_uid VARCHAR(255);
```

nt_projectpref3 cter varying (255)	assigned boolean	student_uid character varying (255)	profilepic_filepath character varying (255)
	true	[null]	00039.jpeg
	true	[null]	[null]
	true	[null]	[null]
	true	[null]	[null]
	true	[null]	student_11_9ac7509c-122c-487a-8f42-551
	false	[null]	[null]
	false	[null]	[null]
	false	[null]	[null]
	false	[null]	[null]
	false	[null]	[null]

3. Now that we have the student_uid added to the database we need to do two things. 1) Have a way to add the new student_uid information to the database, and 2) have a way to grab that information from the database. Luckily, we've already done the hard work in figuring out how to do this… you just gotta figure out how to use our code we have already created!
4. First let's set up a way to add this information to the database, we can't grab anything if nothing is there right? Where would we do this? In the file structure we tried to build it in the way that the application is currently set up. For example, 'TeamBandit'(our main page application) is located in the 'Pages' folder because it is one of the many page locations you can go to with our application. Inside of the 'TeamBandit' we have the 'TeamBanditPages' such as 'Clients', 'Courses', and the 'Email hub'. The students table is located in a course. So we go inside of

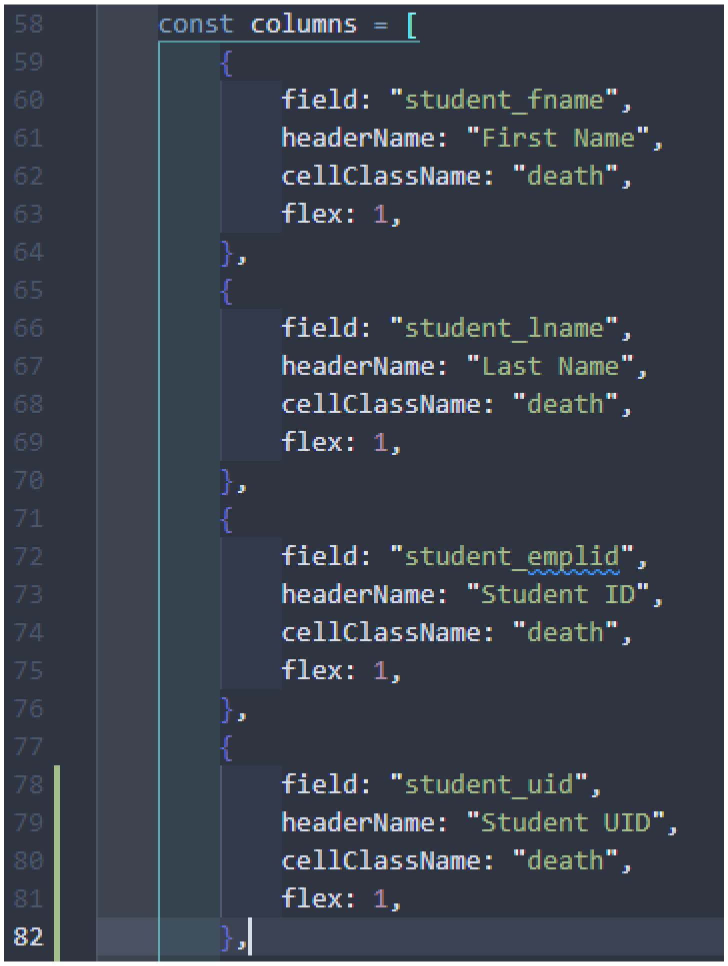
‘Courses’, navigate to ‘CoursePages’ and then inside of there we found the ‘Students’ page!
IF YOU ARE EVER LOST ON WHERE TO LOOK FOR A PAGE, START AT TEAM BANDIT PAGES and mimic wherever you click on the application by opening the corresponding folder.



```
src
  Images
  Pages
    AssignmentPage
    InfoPage
    LandingPage
    PrivacyPolicyPage
    ProjectPages
    StudentProfilePage
    SubmissionPage
    TeamBandit
      TeamBanditPages
        Clients
        Courses
          Components
          CoursePages
            Assignments
            Mentors
            Projects
            Schedule
            Settings
            Students
              Components
              Students.js
              Students.module.css
            TeamAssignment
          CourseRouter.js
          Courses.js
          Courses.module.css
```

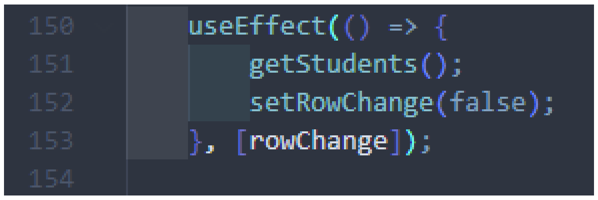
5. The students table is not actually located in the ‘Students.js’ file. If you open up the Components folder inside you will find ‘StudentList.js’. Inside here we want to do two things. a) Create a new column in our table to detail the Student UID, and b) locate where we grab the information about a student so we know where to change it.

a. On line 58 of StudentList.js we have our columns variable. We want to copy an existing column and add our new field and headerName. Field = the name in the database, headerName is what you want to be displayed as a header for the table column. If you want more information on DataGrids I recommend you go to https://mui.com/x/react-data-grid/ .



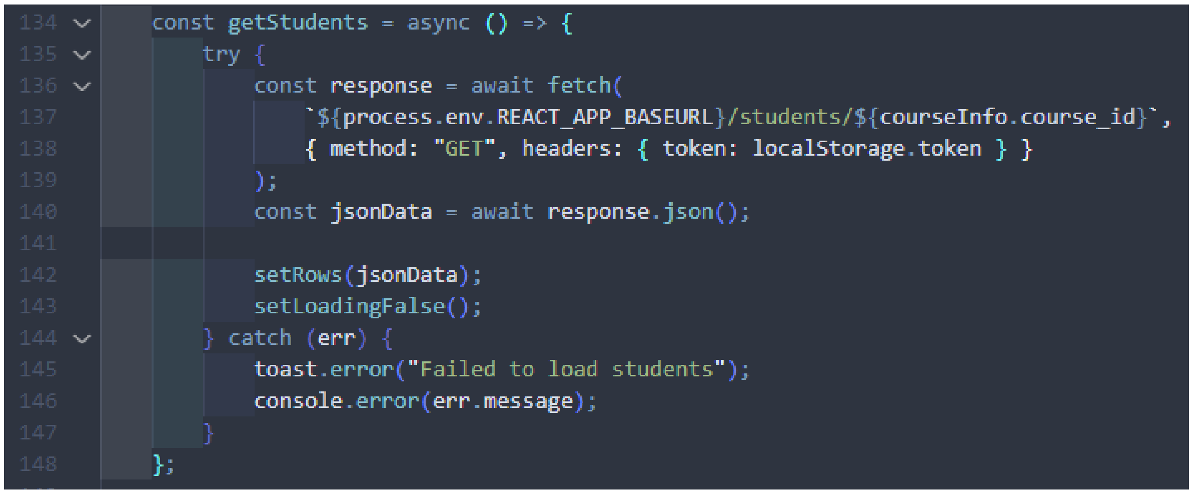
```
58      const columns = [
59          {
60              field: "student_fname",
61              headerName: "First Name",
62              cellClassName: "death",
63              flex: 1,
64          },
65          {
66              field: "student_lname",
67              headerName: "Last Name",
68              cellClassName: "death",
69              flex: 1,
70          },
71          {
72              field: "student_emplid",
73              headerName: "Student ID",
74              cellClassName: "death",
75              flex: 1,
76          },
77          {
78              field: "student_uid",
79              headerName: "Student UID",
80              cellClassName: "death",
81              flex: 1,
82          },
```

b. This is now added as a column! However nothing will get populated since you aren't grabbing the data. From the database. When you first load a react page the useEffect(()) function will be called. If you look inside the useEffect function you will see that we call getStudents();



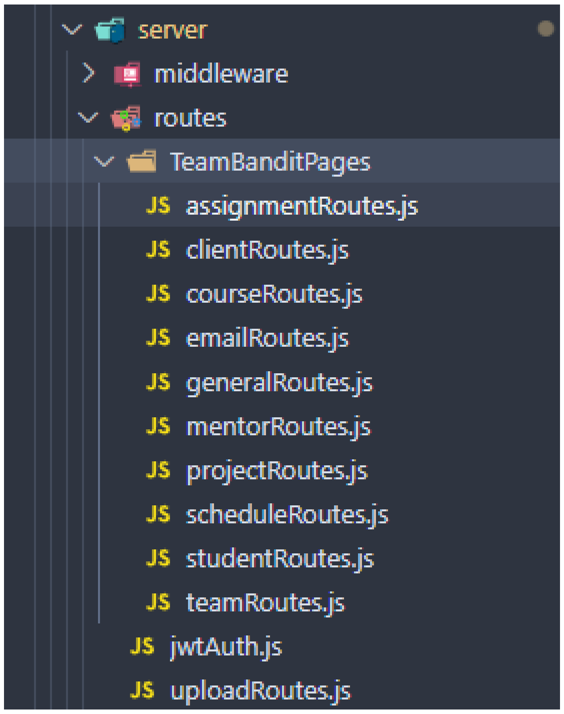
```
150     useEffect(() => {
151         getStudents();
152         setRowChange(false);
153     }, [rowChange]);
154
```

This is what we use to grab the student information, so now if we locate getStudents you will see the express route we use to grab this data



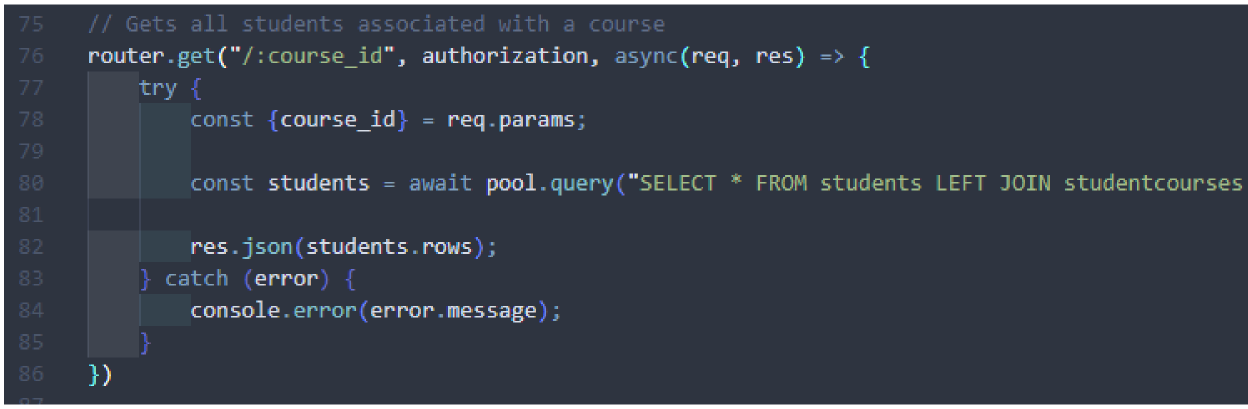
```
const getStudents = async () => {
    try {
        const response = await fetch(
            `${process.env.REACT_APP_BASEURL}/students/${courseInfo.course_id}`,
            { method: "GET", headers: { token: localStorage.token } }
        );
        const jsonData = await response.json();

        setRows(jsonData);
        setLoadingFalse();
    } catch (err) {
        toast.error("Failed to load students");
        console.error(err.message);
    }
};
```

This shows us that the express route is located at **/students/$courseInfo.course_id**. The **/students** indicates that this route is located in the student routes. We now want to navigate to the /server folder to find this route.



```
server
  middleware
  routes
    TeamBanditPages
      assignmentRoutes.js
      clientRoutes.js
      courseRoutes.js
      emailRoutes.js
      generalRoutes.js
      mentorRoutes.js
      projectRoutes.js
      scheduleRoutes.js
      studentRoutes.js
      teamRoutes.js
    jwtAuth.js
    uploadRoutes.js
```

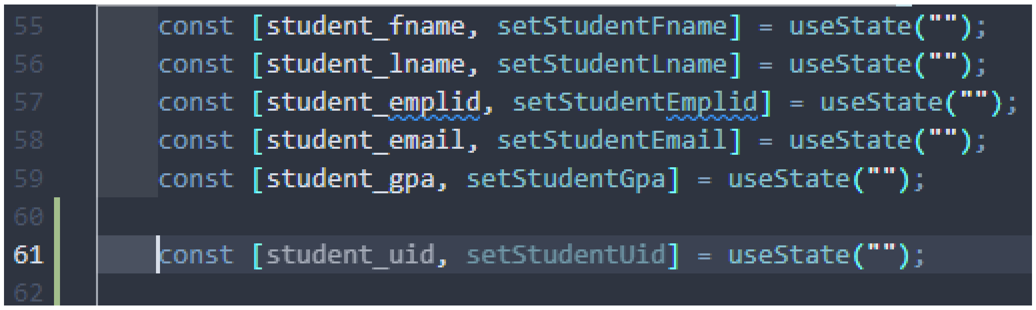
Located in routes we have studentRoutes.js, open this file. If you remember we are looking for a **/$courseInfo.course_id** part of the route. On line 76 we found it!



```
// Gets all students associated with a course
router.get("/:course_id", authorization, async(req, res) => {
    try {
        const {course_id} = req.params;

        const students = await pool.query("SELECT * FROM students LEFT JOIN studentcourses

        res.json(students.rows);
    } catch (error) {
        console.error(error.message);
    }
})
```

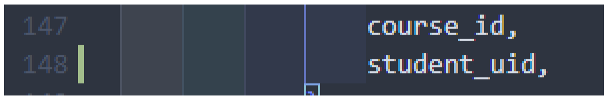
As you can see we now have the SQL query where we get the student information! This may not have been a good example as this query already grabs * from the student database, which means the newly created student_uid is already getting pulled. If it wasn't a * you would need to add student_uid to the SELECT statement.

6. Now we are grabbing the information. Let's get the ability to add that information to the database. If you navigate back to the 'StudentList.js' file you will see an 'AddStudent.js' file. Open this up. There are three things we need to do, a) create a variable to keep track of the student_uid, b) add the variable to the SQL statement, and c) create a React component to have the new field be an option.
    a. We declare all of our variables at the top of the file. So let's declare a variable to keep track of the student_uid. We utilize Reacts useState components for this. https://reactjs.org/docs/hooks-state.html

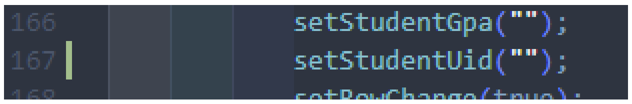


```
    const [student_fname, setStudentFname] = useState("");
    const [student_lname, setStudentLname] = useState("");
    const [student_emplid, setStudentEmplid] = useState("");
    const [student_email, setStudentEmail] = useState("");
    const [student_gpa, setStudentGpa] = useState("");

    const [student_uid, setStudentUid] = useState("");
```

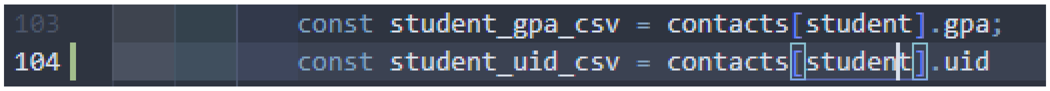
    b. There are two places where we need to add it to our grabbing and adding statements. On line 96 we have our addList function for adding a CSV, and on line 138 we have onSubmitForm for adding an individual student. Lets add the newly created student_uid to these locations to help make.



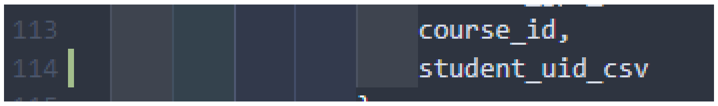
```
                    course_id,
                    student_uid,
```



```
                setStudentGpa("");
                setStudentUid("");
```

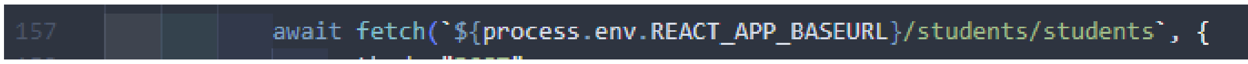


```
const student_gpa_csv = contacts[student].gpa;
const student_uid_csv = contacts[student].uid
```

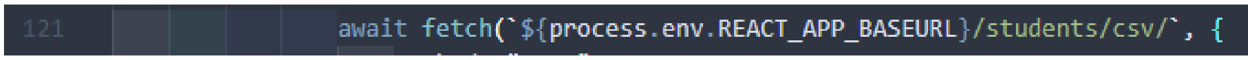


```
course_id,
student_uid_csv
```

With these added we can want to go to the express route to make sure the routes add the new information. We can see that both the CSV and the individual adding routes are located in **/students.**

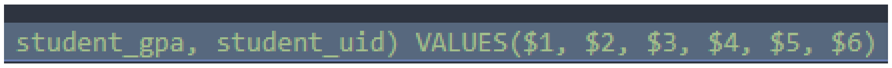


```
await fetch(`${process.env.REACT_APP_BASEURL}/students/students`, {
```

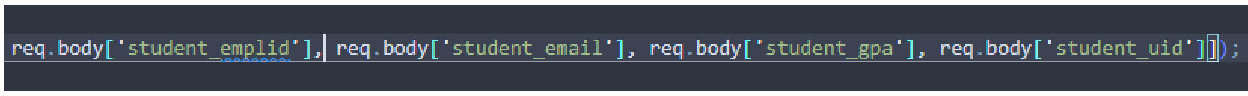


```
await fetch(`${process.env.REACT_APP_BASEURL}/students/csv/`, {
```

We will want to navigate to studentRoutes.js again and find the routes where **/students** and **/csv** are located. After looking at the file the individual one is located on line 8, the CSV is located on line 44.
On line 29 you want to add student_uid like this:

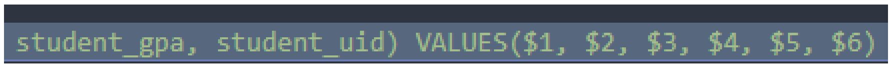


```
student_gpa, student_uid) VALUES($1, $2, $3, $4, $5, $6)
```

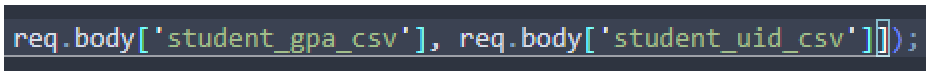


```
req.body['student_emplid'], req.body['student_email'], req.body['student_gpa'], req.body['student_uid']]);
```

On line 66 you want to add student_uid like this:



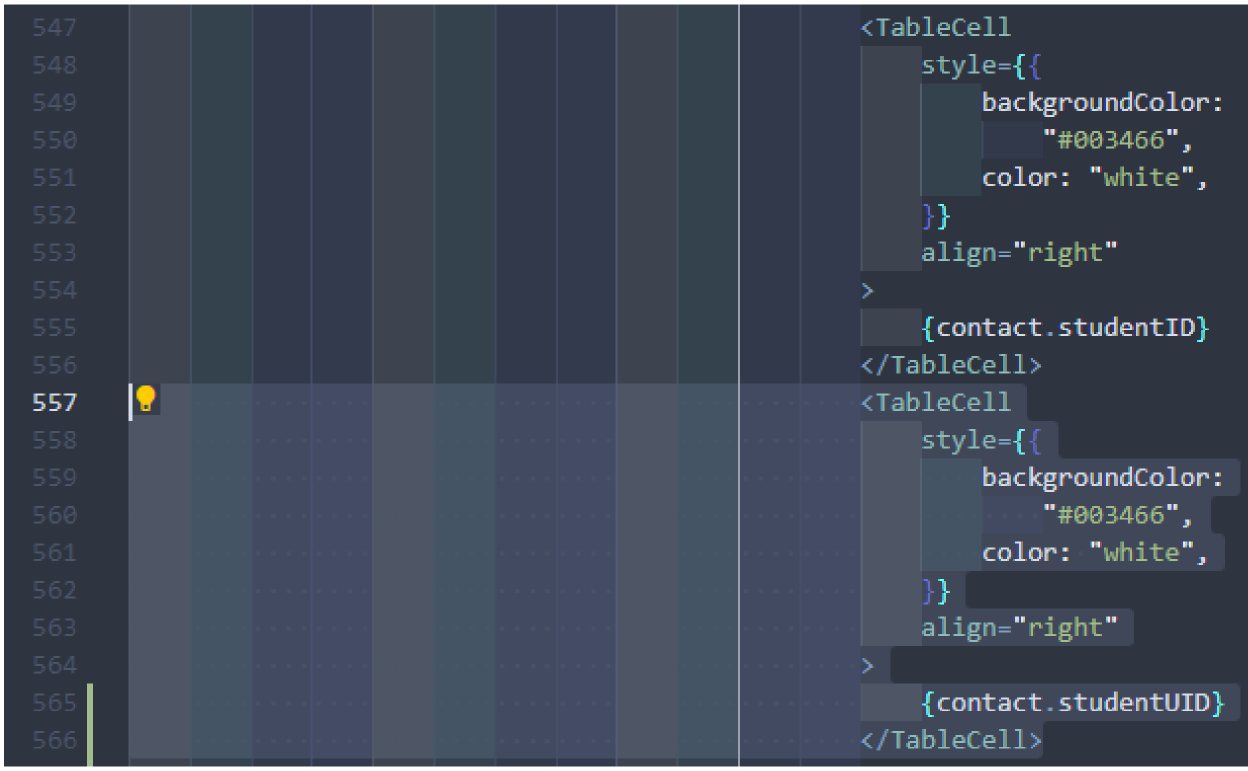
```
student_gpa, student_uid) VALUES($1, $2, $3, $4, $5, $6)
```



```
req.body['student_gpa_csv'], req.body['student_uid_csv']]);
```

Make sure to restart your authRoutes every time you make changes to your express routes. Or else it won't update!!

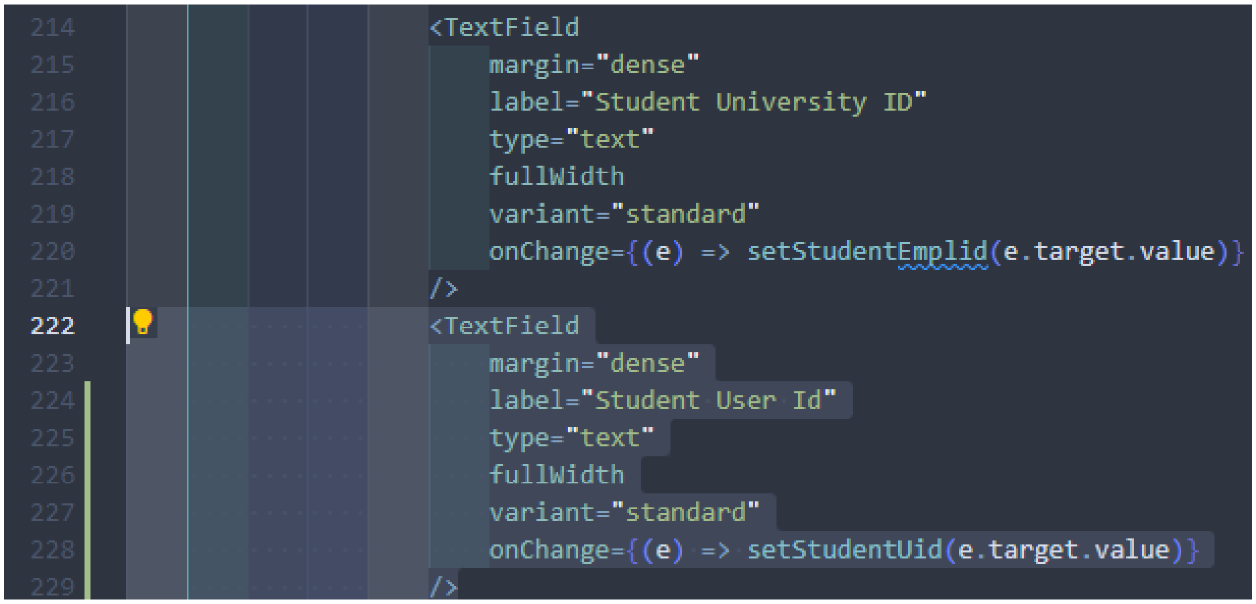
c. Finally we want to be able to add this student information through the React front end. Navigate back to **AddStudent.js**! There are two things we need to do for this, i) Add the new student_uid variable to the CSV upload table, and ii) add the new student_uid variable to the individual add.
   i. You actually don't need to do anything for the CSV uploader, instead we have to change the table to look for the new variable. You can do this by adding this:



```
<TableCell
    style={{
        backgroundColor:
            "#003466",
        color: "white",
    }}
    align="right"
>
    {contact.studentID}
</TableCell>
<TableCell
    style={{
        backgroundColor:
            "#003466",
        color: "white",
    }}
    align="right"
>
    {contact.studentUID}
</TableCell>
```

The parser breaks the values into variables based on the header of the .csv file. In this case we would want to add studentUID to the header of the .csv file so it adds them to this Table Cell.

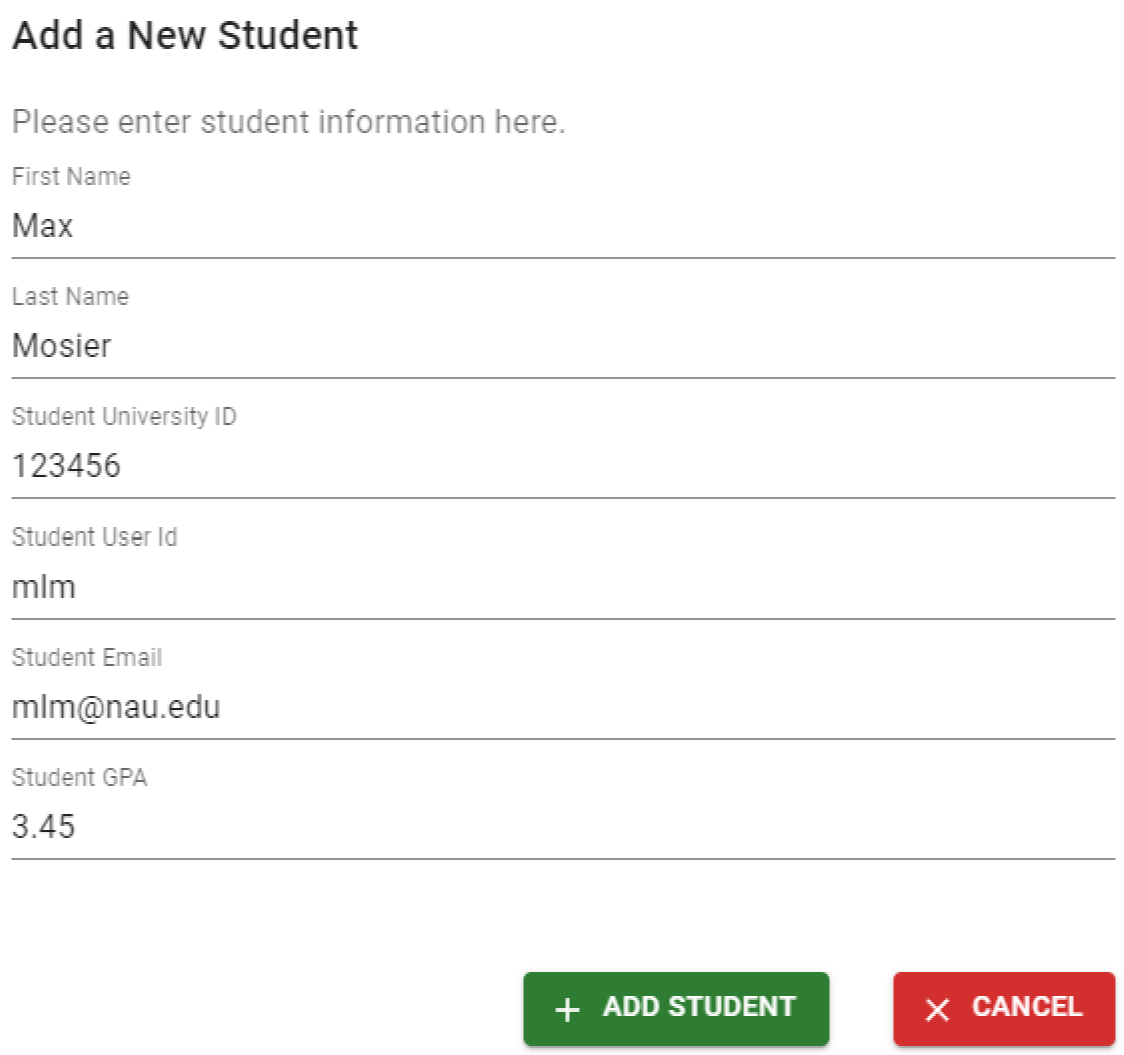
ii. Next, is the adding as an individual student! You'll want to add this:



```
<TextField
    margin="dense"
    label="Student University ID"
    type="text"
    fullWidth
    variant="standard"
    onChange={(e) => setStudentEmplid(e.target.value)}
/>
<TextField
    margin="dense"
    label="Student User Id"
    type="text"
    fullWidth
    variant="standard"
    onChange={(e) => setStudentUid(e.target.value)}
/>
```

This makes it so there is a field to add the student!

7. Now we should be good to add the new information with the addition of a student!

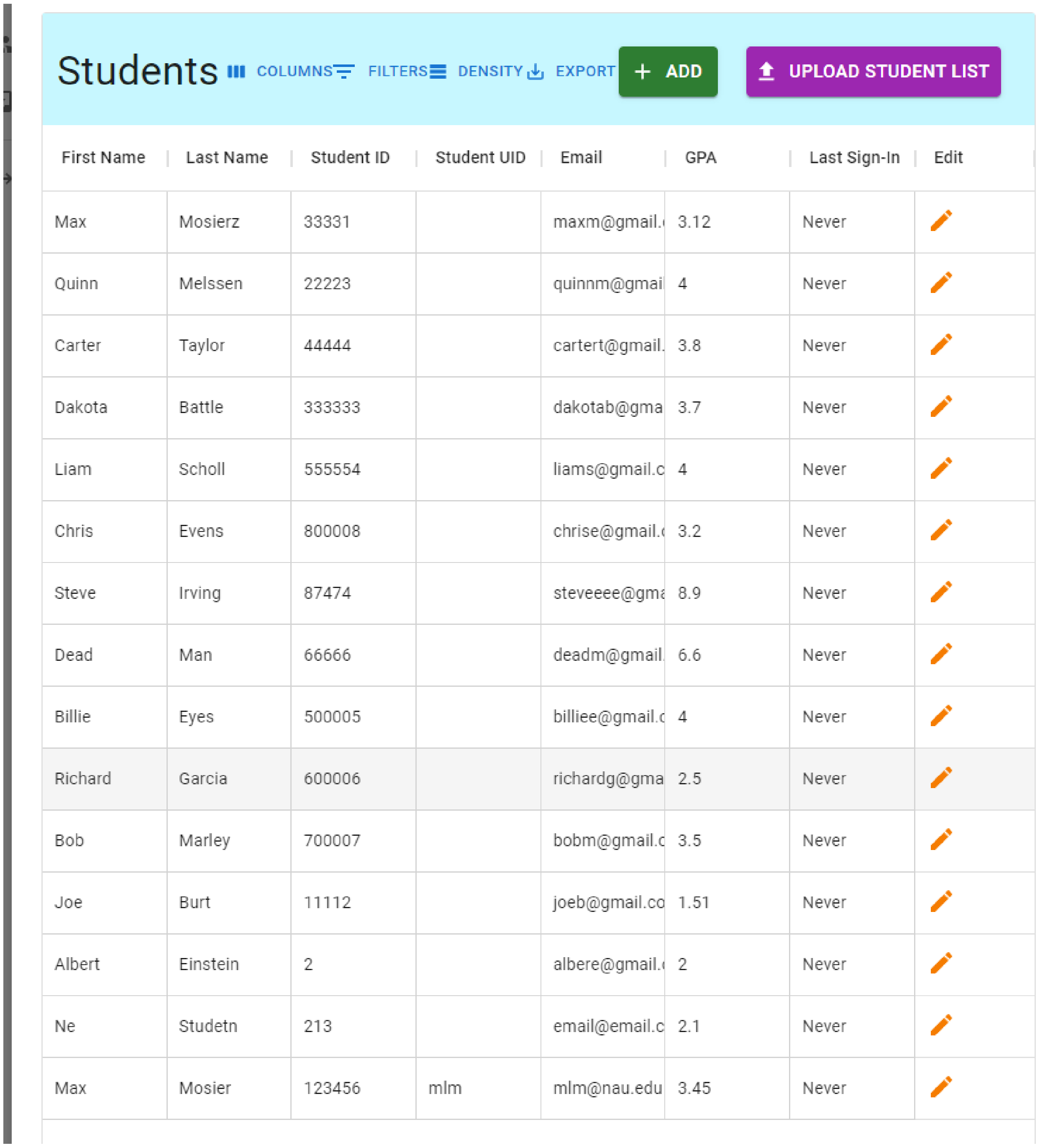


Add a New Student

Please enter student information here.

First Name	Max
Last Name	Mosier
Student University ID	123456
Student User Id	mlm
Student Email	mlm@nau.edu
Student GPA	3.45

After filling out a student lets see if adding them works!



First Name	Last Name	Student ID	Student UID	Email	GPA	Last Sign-In	Edit
Max	Mosierz	33331		maxm@gmail.	3.12	Never	✎
Quinn	Melssen	22223		quinnm@gmai	4	Never	✎
Carter	Taylor	44444		cartert@gmail.	3.8	Never	✎
Dakota	Battle	333333		dakotab@gma	3.7	Never	✎
Liam	Scholl	555554		liams@gmail.c	4	Never	✎
Chris	Evens	800008		chrise@gmail.	3.2	Never	✎
Steve	Irving	87474		steveeee@gma	8.9	Never	✎
Dead	Man	66666		deadm@gmail.	6.6	Never	✎
Billie	Eyes	500005		billiee@gmail.c	4	Never	✎
Richard	Garcia	600006		richardg@gma	2.5	Never	✎
Bob	Marley	700007		bobm@gmail.c	3.5	Never	✎
Joe	Burt	11112		joeb@gmail.co	1.51	Never	✎
Albert	Einstein	2		albere@gmail.	2	Never	✎
Ne	Studetn	213		email@email.c	2.1	Never	✎
Max	Mosier	123456	mlm	mlm@nau.edu	3.45	Never	✎

As you can see, Max Mosier now has mlm as their Student UID! Congratulations!
Make sure that you restart your authRoutes, if on a MAC you will need to restart your whole server, or else the new UID will not get added.

Next steps? Well… we have a lot of students who need the Student UID, the best way would be to do it through the edit option. I guess you might start there by figuring out how to get the editing to work! This is just an example of the workflow generally followed when adding, grabbing and displaying the data on the application!

10 Appendix B: Helpful Resources for Development

10.1 Helpful Videos

When constructing the application, we were unsure of where to start, however, we came across a YouTube video series that helped us build the structure of our application. We recommend you watch these five videos as they will help you get started in the application!

PERN Stack Course - Postgres, Express, React, and Node
freeCodeCamp.org - (1:22:44)
https://www.youtube.com/watch?v=ldYcgPKEZC8&t=6s

Learn JWT with the PERN stack by building a Registration/Login system Part 1
The Stoic Programmers - (1:33:49)
https://www.youtube.com/watch?v=7UQBMb8ZpuE&t=12s

Learn JWT with the PERN stack by building a Registration/Login system Part 2
The Stoic Programmers - (1:11:03)
https://www.youtube.com/watch?v=cjqfF5hyZFg

Learn Database Design by combining our JWT and Pern stack Todo List app together Part 1
The Stoic Programmers - (45:23)
https://www.youtube.com/watch?v=l3njf_tU8us

Learn Database Design by combining our JWT and Pern stack Todo List app together Part 2
The Stoic Programmers - (1:27:10)
https://www.youtube.com/watch?v=25kouonvUbg&t=25s

While these five videos take a long time, it is important to go through them all if you want to further understand the application. We recommend you do them in the order they are presented.

10.2 Helpful Links

For some other helpful information we wanted to give you access to our Draw.io which had sketches of initial concepts and architecture. Hopefully this proves helpful!

Draw.io Sketch
https://drive.google.com/file/d/1-SAk5xolAMnn8JFaYsnEoJxicKu_FsBZ/view?usp=sharing

Additionally we wanted to share the Material UI website. Material UI contains all of the visual elements we used for our application. If you want to make additions or changes you will need to familiarize yourself with them.

https://mui.com/material-ui/getting-started/installation/