Team Outlaws

Software Design Document

Project Sponsor and Mentor:

Dr. Eck Doerry

Team Members:

Quinn Melssen
Liam Scholl
Max Mosier
Dakota Battle

February 15th, 2022

Version 2.0

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. These modules will be examined in more detail in the Implementation Overview section of this document. A primary focus of our design will be on high usability, as their intent is to *facilitate* the running of these types of classes, not add in unwanted complexity. In order to ensure this goal, functional and performance requirements have been developed during the Fall Term; we briefly review these here to establish context.

Many of the functional requirements center around the capabilities of the users, of which there are several types. These user-archetypes can be broken down into students, clients, mentors, and organizers. Our functional requirements have been broken down into sections, with 3 of them loosely corresponding to each of the 3 problem areas listed above.

The first section focuses on Course Initialization which corresponds to the 'Gathering Projects' module. Some of the key requirements in this section focus on the capabilities of organizers prior to a class's formal inception, as shown;

- A.1: A faculty member can create a faculty account.
- A.2.2: Faculty can update a status tracker for each project directly on the dashboard. This allows faculty to keep track of where a project stands in terms of their development stages.
- A.2.3: The dashboard includes a chain of emails from the client(s) associated with a project that a faculty member can interact with.

Section C focuses on Team Assignments, solving many of the problems noted in the 'Forming Teams' module listed above. Some of the key requirements are as follows;

- C.1: Mechanism for students to enter project preferences based on a rating schema.
- C.2: Faculty can assign students to projects based on the previously mentioned project preferences.

Finally, Section D addresses many of the issues raised by the 'Executing class' module, attempting to implement functionality to alleviate the burdens of running these team based classes after their inception. Some of the key high level requirements in this module include:

- D.2: Faculty can create deliverables for students to view.

- D.2.1: Students have the ability to upload their deliverables on the web application.
- D.4: Users can navigate to a course and view information related to that course.

The performance requirements of this project seek to further ensure ease of use by setting failure tolerances. These tolerances will be used during user testing as a baseline for refactoring components in order to make the application more usable. The requirements are formatted in such a way that comparing them to real world results will be easy, such as “80% of faculty members can upload a pdf of a deliverable description in 1 minute after two attempts.” By observing different users attempting to complete this process we will be able to accurately decide whether or not our performance requirements have been met.

This document will highlight the design strategies that we will use to ensure our requirements are fulfilled.

2 Implementation Overview

In order to solve our client’s problem, we have envisioned and designed TeamBandit, a web application that will allow Dr. Doerry and potentially any team manager to organize, collect information, and manage tasks to offer a team-based course management. Features of this web application will include:

- A user authentication module to ensure that a user can sign up for an account and securely login and view all information contained within their account. This information includes all of the below modules.
- An email hub module to organize the process of finalizing project propositions.
- A clients module to keep track of, view, edit, and delete client information.
- A courses module that supports adding projects, adding students, adding mentors, allowing students to submit information like project preferences and assignments.
- A teams/projects module that allows teams to be created, allows students to be assigned to teams, have websites to be created

During our implementation process our team will be using an agile work-flow to focus on implementing the key features we want out of the web application. Every week we plan on meeting as a team as well as with our client to discuss previous weeks accomplishments and discuss what we need to accomplish next. This will allow us to develop and refine our vision for the application and its user experience.

In order to create the desired web application our team identified four technology frameworks in our Tech Feasibility Document that will help us accomplish our goal. These frameworks include PostgreSQL, Express, React, and Node or PERN for short.

- PostgreSQL is a database management system that will allow us to have user accounts that have information associated with them.
- Express JS will be used to pull information from our database and store information in the database.
- React will help us easily create the front end and visuals of our application.
- Node will simply help us create and run our web application.

With our process and technology stack identified, our next step is to start planning how we are going to utilize our technology to create our web application. This will be through detailing our architectural overview below.

3 Architectural Overview

TeamBandit is made up of several distinct modules which each serve a specific 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:

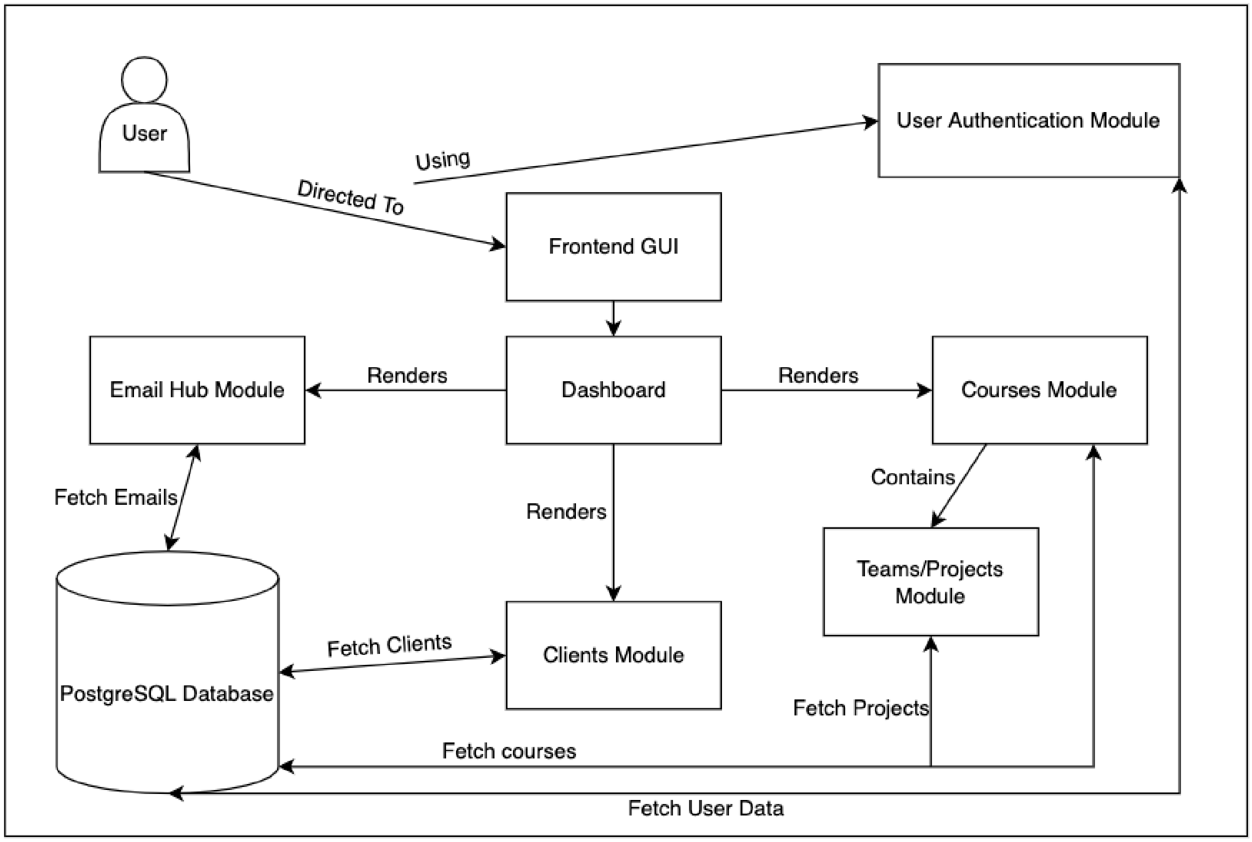

Figure 3: Diagram showing the high level architecture of the web application

As displayed in Figure 3, each module's individual functionality in the system interacts with our chosen frontend framework, React, and the database management system, PostgreSQL to some degree. Rather than treating React and PostgreSQL as separate modules, their functionalities will be appropriately detailed and referenced throughout each module in section 4 Module and Interface Descriptions. Another component that will not be fleshed out in its own module is the dashboard, also shown in Figure 3. The reason for this is that the dashboard's functionality is solely to use React to display the major modules of the application. As it is still a frequently used component in the web application, a more detailed description is provided below.

When a user opens the dashboard, they can expect to essentially see a summary of the most recent course they visited, or favorited courses if the course organizer is the user and chooses to favor them. From the dashboard, the user can access any major component of the system via the accessibility selection menu on the left–hand side of the screen. The data summaries displayed on the dashboard are able to be interacted with in specific ways pertaining to the information being shown. An example of this is the ability to click on a displayed email in the dashboard's email summary and jump directly to the email chain with that particular recipient/sender, rather than simply opening the email hub's standard view.

The key features and responsibilities of each module in the system architecture are detailed belowBelow, the key features and responsibilities of each module in the system architecture is described.

3.1 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.

3.2 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 will have the ability to add, delete, and edit the details of a selected client. This information will be 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.

3.3 Email Hub Module

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

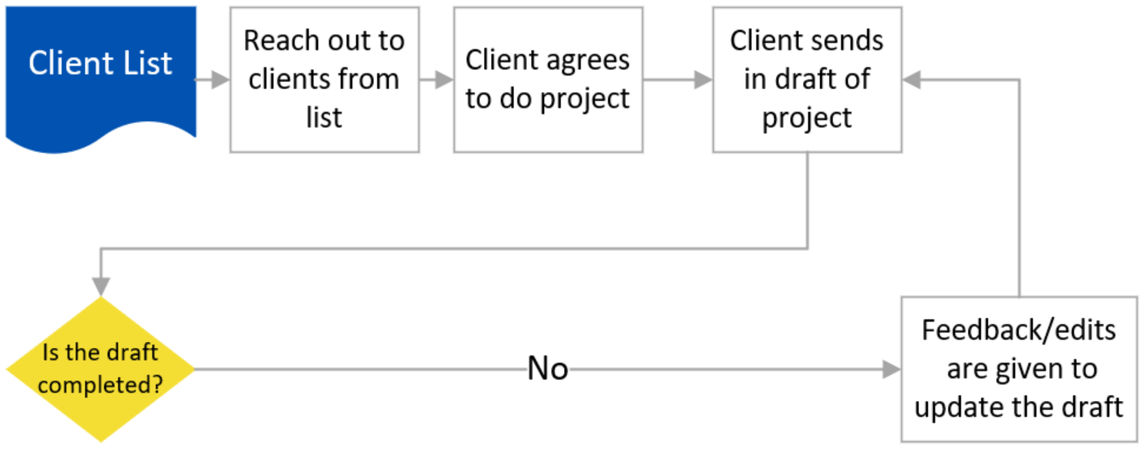
Client List
Reach out to clients from list
Client agrees to do project
Client sends in draft of project
Is the draft completed?
No
Feedback/edits are given to update the draft

Figure 3.3: 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 will be created for faculty to see all of the emails organized by clients in one place on the web application. The clients and each of their associated emails will also be associated with a project specified by the faculty mentor.

TeamBandit's email hub will be accessible from the dashboard and simply displays emails to and from a course organizer and the client for a project. The organizer will be able to 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.

3.4 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.

3.5 Teams/Projects Module

The teams/projects module is within the courses module as there will be 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 administrator has the ability to add or remove students from a team. Members of a team can submit deliverables within a course, and it 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.

Now that the outline of the general features and components have been determined, we will now break up these features into modules for further clarification, this will be done in the below module with UML diagrams and further functionality descriptions.

4 Module and Interface Descriptions

In order to better analyze the modules in the architecture above, each module will be broken down into A) a short natural-language description of the responsibilities of the component and how it fits within the larger context of the architecture, B) a UML class diagram of the classes involved in this component, and a description of the public interface of the component that explicitly outlines services that the component provides. These modules are separated into their own sections below.

4.1 User Authentication Module

Every individual using TeamBandit as a means to participate in a course will need to sign up for and have secure access to an account. These individuals include faculty members, mentors, and students. In order to accurately detail what this module entails, we will discuss the description of its responsibilities, draw a detailed UML diagram to show how it's going to work, and describe the functionality of each function in the diagram.

A: Description of Responsibilities

This module is key in making sure that our client, students, and mentors can ensure that 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. These technologies include PostgreSQL, Express, React and some minor frameworks, including Bcrypt and JWT.

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

With the above technologies, we will create 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. Once signed in, the user will be able to access the TeamBandit web application and any information about their account. This will be done by utilizing the JWT library to create ‘tokens’ that identify who the user is.

B: UML Class Diagram and Descriptions

In order to properly plan out the User Authentication module, we broke it down into a Unified Modeling Language diagram to help us plan out the code for this module. This diagram is displayed below.

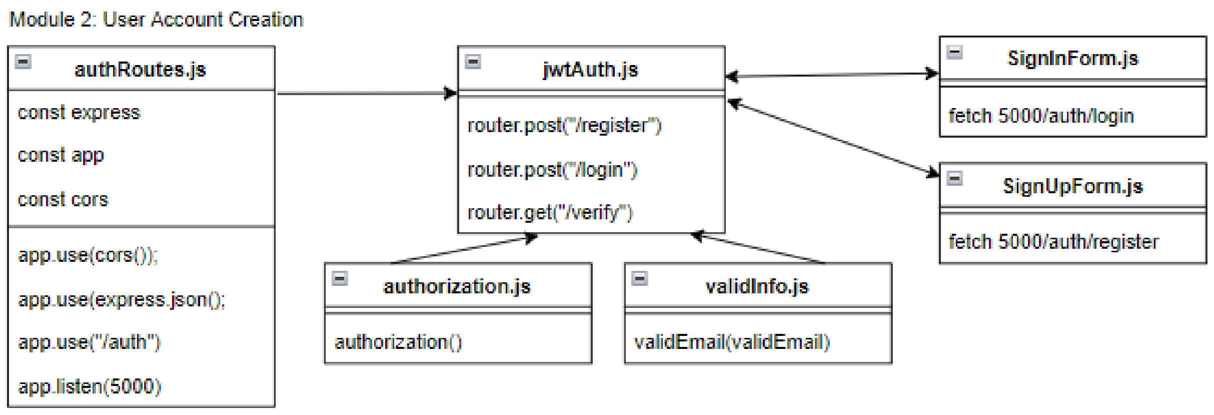

Figure Module 2: UML Diagram displaying the UML structure of the User Authentication module of the web application

This User Authentication module will handle all of the functionalities listed below.

authRoutes.js

- app.use(cors()) – Cross-Origin Resource Sharing or CORS is an HTTP-header based mechanism that will allow our application to allow our server to indicate the origins of our requests to the database. This function will allow us to use this framework.
- app.use(express.json()) – This function will specify using Express's JSON functionality. JSON is a file type where information is stored in curly brackets and usually associated with a key value. This function will allow us to send data to and from the database in JSON format for ease of use.
- app.use("/auth") – This function will specify database routes to our PostgreSQL database. These routes will include SQL Queries that will directly interact with our database.
- app.listen(5000) – This function sets up all of our database routes to be set up on port 5000. This port will allow us to communicate directly with our routes to our database.

jwtAuth.js

- router.post("/register") – This function will allow us to check to see if a user is already in the database. If the user is not in the database, this function will bcrypt their password and insert their information into the PostgreSQL database.
- router.post("/login") – This function will check to see if the user is already in the database, check to see if the user has entered valid information, then generate a JWT token which will let the web browser know that the user is signed in.

- router.get("/verify") – This function will be called whenever a user enters a new web page in our application. It will make sure that the user has a valid JWT token, if they don't the user will not be able to access the page.

Authorization.js

- authorization() – This function will act as a middleware helper function that will help router.get("/verify") to check if the user's JWT token is valid.

validInfo.js

- validEmail(userEmail) – This function will act as a middleware helper function that will help router.post("/register") and router.post("/login") to verify the user is inputting a valid email. This function will implement RegEx to help identify if the email is valid.

SignInForm.js

- fetch 5000/auth/login – This function will be called in SignInForm with the information that will get filled out in the form of this page. This function will call the router.post("/login") function to check if the user can log in.

SignUpForm.js

- fetch 5000/auth/register – This function will be called in the SignUpForm with the information that will get filled out in the form of this page. This function will call the router.post("/register") function to check if the user can register.

Together, these functions will help us achieve our end goal of creating a fully functional User Authentication Module.

4.2 Clients Module

As stated in section 3, the clients module will consist of a table of clients created by the course organizer. In order to accurately detail what this module entails, we will discuss the description of its responsibilities, draw a detailed UML diagram to show how it's going to work, and describe the functionality of each function in the diagram.

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 will have to take full advantage of many of the features that our technology stack offers. These technologies include PostgreSQL, Express, React.

- Using **PostgreSQL,** we will be able to store and pull information about the clients such as their email, name and password.
- **Express** will allow 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** will help us build better UI and UX experiences for a course organizer when they view the clients table in the application.

With the above technologies, we will create a centralized location of project clients that is easily viewable by the course organizer.

B: UML Class Diagram and Descriptions

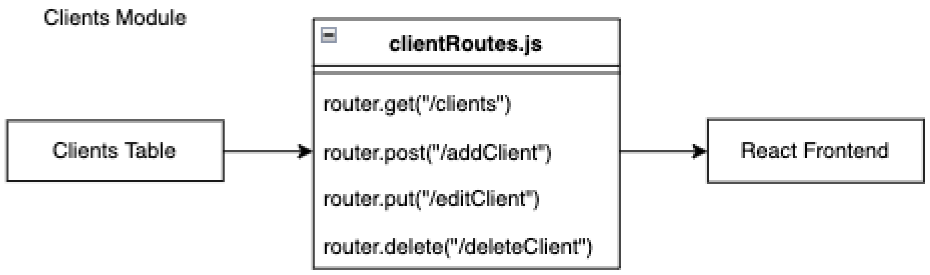
Clients Module
Clients Table
clientRoutes.js
router.get("/clients")
router.post("/addClient")
router.put("/editClient")
router.delete("/deleteClient")
React Frontend

Figure 4.2: Diagram showing the UML interactions for this module

The Clients module will handle all of the functionalities listed below.

clientRoutes.js

- router.get("/clients") – This function will be called when a course organizer enters the clients table in the web application. The function will ensure that all clients are displayed to the organizer to be viewed and edited.
- router.post("/addClient") – This function will be called when a course organizer wishes to add a client to the client table. The information that can be added is the name of a client, their email address, company, associated projects, and any personal notes that the course organizer has about the client.
- router.put("/editClient") – This function will be called when a course organizer wants to edit the information of an individual client in the clients table. The information that can be edited is all of the same information that can be added.
- router.delete("/deleteClient") – This function will be called when a course organizer wants to delete the information of an individual client in the clients table.

The above routes will ensure that the correct information can be viewed, added, edited, and deleted in the clients table by the course organizer. Clients also play an important role in the next module, the email hub module, where their information will also be displayed to ensure an organized messaging viewing experience.

4.3 Email Hub Module

The email hub module will be a centralized location of organized email chains differentiated by the communicating email of the project client. In order to accurately detail what this module entails, we will discuss the description of its responsibilities, draw a detailed UML diagram to show how it's going to work, and describe the functionality of each function in the diagram.

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** will act 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** will allow 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** will help 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 will allow us to pull information from emails if our created email is carbon copied (CC'd) in the email.

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

B: UML Class Diagram and Descriptions

In order to properly plan out the Course Initialization module, we broke it down into a Unified Modeling Language diagram to help us plan out the code for this module. This diagram is displayed below.

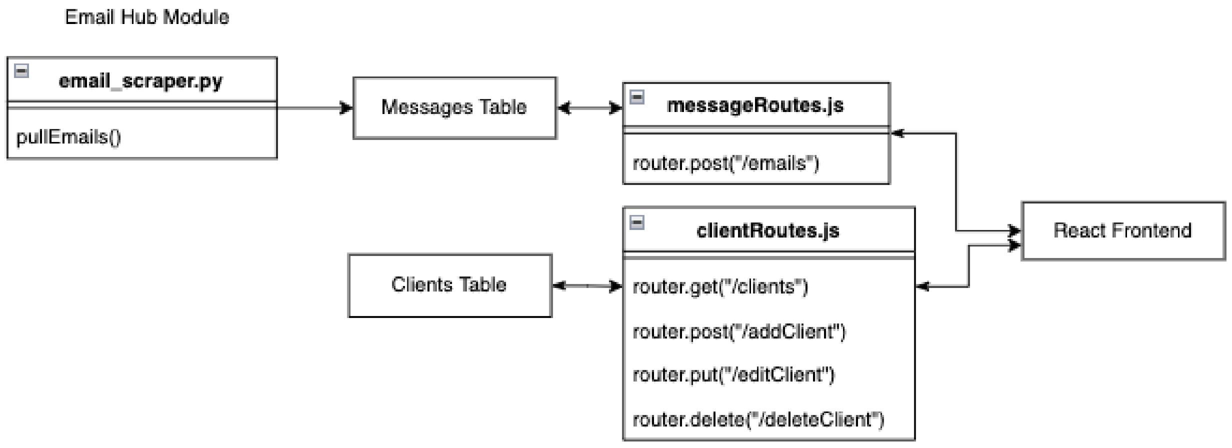

Figure 4.3: Diagram showing the UML interactions for this module

This Email Hub module will handle all of the functionalities listed below.

email_scraper.py

- pullEmails() – This function will be called sporadically to check to see if there are any new emails in our TeamBandit email. If there are any new emails, this function will take these messages and put information from them into our database.

messageRoutes.js

- router.post("/emails") – This function will grab all emails associated with a given client and organizer for use in our React code.

clientRoutes.js

- This is the same file used in the Clients Module, however, only router.get("/clients") will be used to display the clients. To edit client information, the course organizer will need to navigate to the clients module in the web application.

Together, these functions will help us to achieve our end goal of creating an effective Email Hub Module.

4.4 Courses Module

The courses module will act as a key module when creating TeamBandit as it involves the course organizer creating courses. In order to accurately detail what this module entails, we will discuss the description of its responsibilities, draw a detailed

UML diagram to show how it's going to work, and describe the functionality of each function in the diagram.

A: Description of Responsibilities

The courses module will be 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 and mentors will have their accounts and be assigned to their courses, and all students and mentors will be assigned to their projects. The actions that the course organizer may perform include:

- Creating and updating a course
- Deleting or archiving 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: UML Class Diagram and Descriptions

In order to properly plan out the courses module, we broke it down into a Unified Modeling Language diagram to help us plan out the code for this module. This diagram is displayed below.

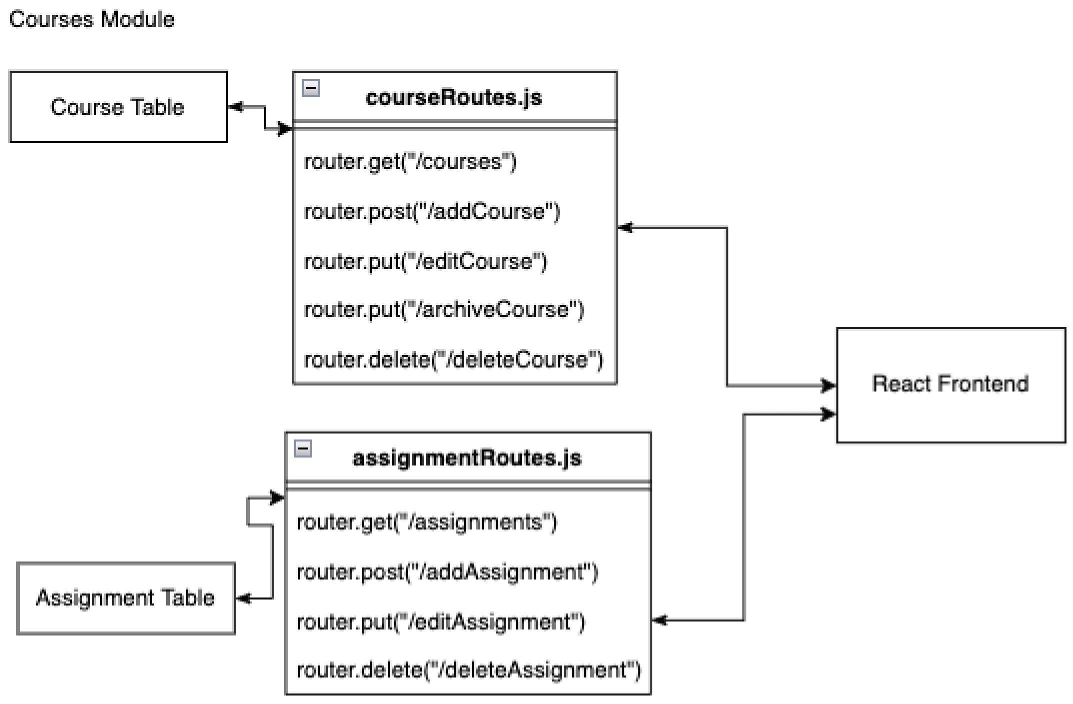
Courses Module
Course Table
courseRoutes.js
router.get("/courses")
router.post("/addCourse")
router.put("/editCourse")
router.put("/archiveCourse")
router.delete("/deleteCourse")
React Frontend
assignmentRoutes.js
router.get("/assignments")
router.post("/addAssignment")
router.put("/editAssignment")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

Figure 4.4: UML Diagram displaying the UML structure of the Courses module of the web application

The Courses module will handle all of the above functionalities, which are described below.

courseRoutes.js

- router.get(“/courses”) – This function is responsible for fetching and displaying all courses that belong to a user to that user on their courses page.
- router.post(“/addCourse”) – This function will add a new course to the courses table in the database when a course organizer specifies that a new course be added.
- router.put(“/editCourse”) – This function will edit an existing course’s information in the courses table in the database when a course organizer specifies that a course be edited. These edits can include changing the name of a course or archiving a course.
- router.put(“/archiveCourse”) – This function will archive an existing course in the courses table in the database when a course organizer specifies that a course be archived.
- router.delete(“/deleteCourse”) – This function will delete a course from the courses table in the database when a course organizer specifies that a course be deleted.

assignmentRoutes.js

- router.get(“/assignments”) – This function is called when all assignments need to be displayed to the course organizer.
- router.post(“/addAssignment”) – This function is responsible for adding a new assignment to the assignments table in the database. It will be called when an organizer adds a new assignment in the assignments page.
- router.put(“/editAssignment”) – This function is responsible for editing an existing assignment in the assignments table in the database. It will be called when an organizer edits an assignment in the assignment’s settings page.
- router.delete(“/deleteAssignment”) – This function is responsible for deleting an existing assignment in the assignments table in the database. It will be called when an organizer deletes an assignment in the assignments page.

The courses module acts as a foundation for the following teams and projects module, which will work directly with the courses module to fulfill all of its responsibilities.

4.5 Teams/Projects Module

After all user accounts are created, including faculty, students, and mentors, students and mentors will need to be assigned to teams. These teams will be formed to work on the projects collected in the Email Hub Module.

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 and mentors, 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.

Along with students and projects, mentors will also need to be created and will be displayed in their own table which will contain mentor names and email addresses. When creating a project, a course organizer will be able to assign students, mentors, and associate clients to that project all in one place.
In order to accomplish these tasks, we will need to utilize these technologies:

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

B: UML Class Diagram and Descriptions

In order to properly plan out the Team Assignments module, we broke it down into a Unified Modeling Language diagram to help us plan out the code for this module. This diagram is displayed below.

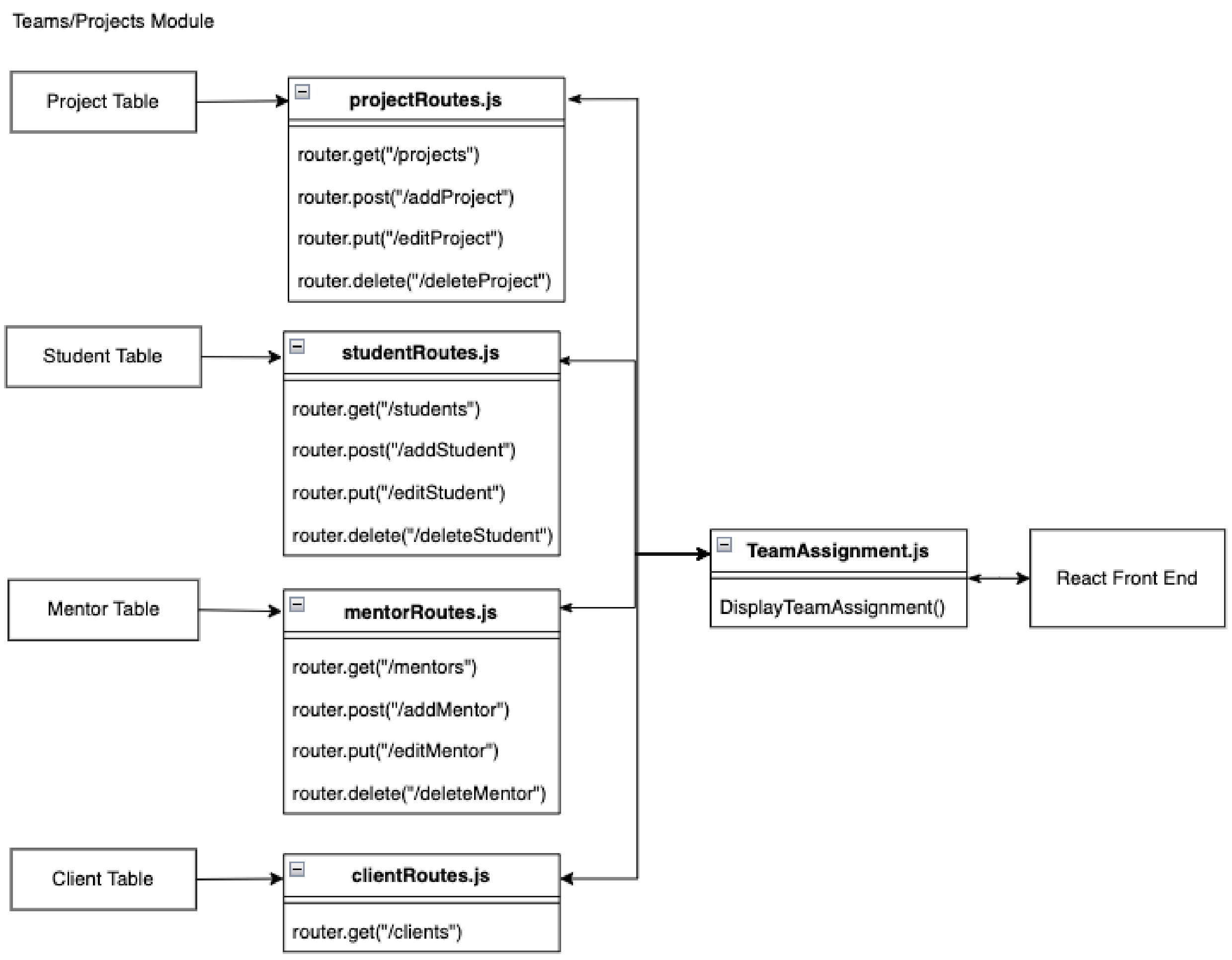

Figure Module 3: UML Diagram displaying the UML structure of the Teams Assignment module of our application

The Teams/Projects module will describe all of the above functionalities below:

projectRoutes.js

- router.get(“/projects”) – This function will be called when a course organizer enters the team assignments portion of a course. The function will ensure that all projects are displayed to the organizer to be assigned students and mentors to.
- router.post(“/addProject”) – This function will add a project inside of a course.
- router.put(“/editProject”) – This function will edit a project with the new information supplied by the course organizer.
- router.delete(“/deleteProject”) – This function will delete a project if specified by the course organizer.

studentRoutes.js

- router.get(“/students”) – This function will be called when a course organizer enters the team assignments portion of a course. The function will ensure that all

students are displayed to the organizer to be assigned to a project by the organizer.
- router.post("/addStudent") – This function will add a student inside of a course.
- router.put("/editStudent") – This function will edit a student with new information supplied by the course organizer.
- router.delete("/deleteStudent") – This function will delete a student from a course if specified by the course organizer.

mentorRoutes.js
- router.get("/mentors") – This function will be called when a course organizer enters the team assignments portion of a course. The function will ensure that all mentors are displayed to the organizer to be assigned to a project by the organizer.
- router.post("/addMentor") – This function will add a mentor inside of a course.
- router.put("/editMentor") – This function will edit a mentor with new information supplied by the course organizer.
- router.delete("/deleteMentor") – This function will delete a mentor from a course if specified by the course organizer.

clientRoutes.js
- This is the same file used in the Clients Module, however, only router.get("/clients") will be used to display the clients to be associated with a project. To edit client information, the course organizer will need to navigate to the clients module in the web application.

TeamAssignment.js
- This component will utilize all of the above functions in order to get all necessary information for team assignments for the course organizer.

5 Implementation Plan

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.

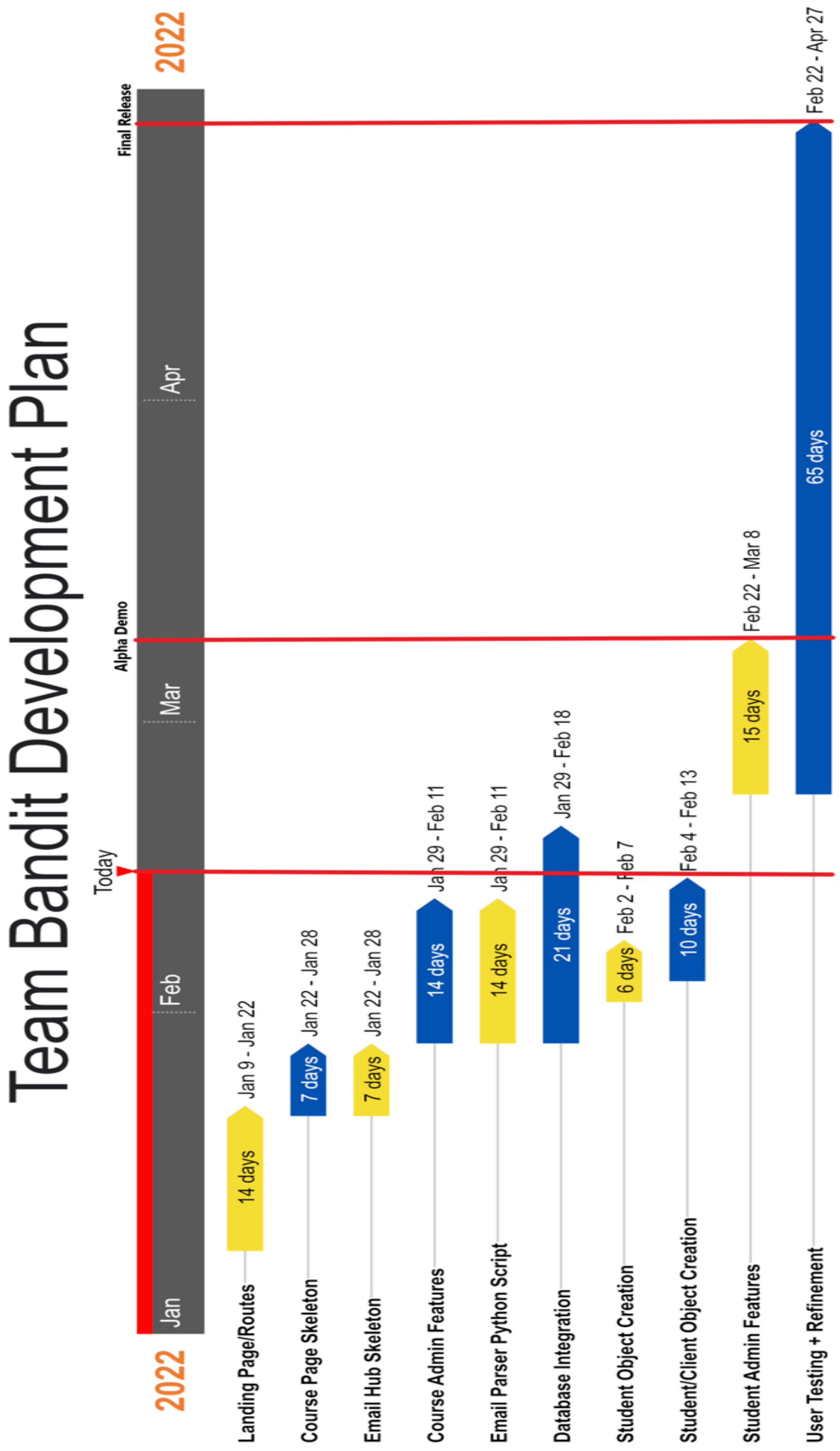

Figure 5.0: This Gantt chart shows key components in our development plan moving forward

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 -Mentor account route -Adding clients -Client table	-Project creation Assigning students to project -Student/Mentor 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.

6 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.