Team Outlaws

Software Testing Plan

Project Sponsor and Mentor:

Dr. Eck Doerry

Team Members:

Quinn Melssen

Liam Scholl

Max Mosier

Dakota Battle

March 27th, 2022

Version 1.0

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. TeamBandit seeks to be a web-application that serves as a comprehensive technical solution to streamline the delivery of team-based project courses, nullifying many of these complexities.

As a project on the scale of TeamBandit comes together, systems can quickly become complex and convoluted, resulting in the need to perform rigorous testing. This is especially true for user-facing applications that will be used by many people for many different use cases. In order to ensure that the end-user will receive the best product possible, with as few bugs as possible, unit and integration tests are key during the development process.

With all of this in mind, Outlaws have devised a host of software tests focused around the use of Selenium. Our testing plan focuses on exhaustively utilizing the application from the user perspective as failures from the end-user point of view are the most pressing. With a small team such as ours and a limited time scale, user testing can be hard to conduct efficiently, thus the use of automation software such as Selenium to expedite the testing process.

Our integration portion that takes a closer look at the interactions between our various modules will be completed using the React Testing Library, a feature that comes baked in with the create-react-app boilerplate that we used as the foundation of our system. By utilizing something already existing in our software, we hope to lower overall development time for our testing and get useful feedback as soon as possible.

Finally, we hope to do small-scale user testing to have real end-users get hands on with our software. By this point in our testing process we hope to have caught and fixed most of the bugs, however we as developers have an insight into the system and its use that a new user may not. Thus, this portion of the testing hopes to find convoluted use cases that someone may not be able to pick up on their own, and shed light on how we can fix them.

Each of these testing plans will be laid out in more detail below.

2 Unit Testing

In order to make sure our TeamBandit application works as intended, Unit Testing is a foundational step in testing and making sure the application works as intended. Unit Testing is a type of software testing where individual ‘units’ or ‘components’ of a software are tested. The purpose of testing each one individually is to validate that each unit of the software code performs as expected. Unit Tests isolate a section of code and verify its correctness. A unit may be a function, method, procedure, module or object.

Unit Testing is primarily done during the development stages as you are supposed to test each unit after writing them. It’s important to not take a lax approach to Unit Testing, as you will end up doing it anyways in Integration, System, and Usability Testing. Unit Tests help to fix bugs early in the development cycle and save costs. It helps the developers to understand the testing code base and enables them to make changes quickly.

While we did not start actual Unit Testing until later on in the stages of development. We have adapted a Unit Testing style as we implemented features of TeamBandit. For example, when implementing our Login system for TeamBandit, we tested to see what would happen when:

1. You try logging in with an existing user, which should log you in.
2. You try logging in with a user that doesn’t exist, which should not log you in.
3. You try logging in with blank credentials, which should not log you in.

While effective in determining if a unit's functionality properly exists, this system could have been automated to save time and get better feedback on our units.

Now that we are focusing on Unit Testing, we want to keep the focus on testing features that our users will directly be using. Our goal is to ensure that we cover as many use cases scenarios as possible. With that in mind we wanted to find a tool set that effectively mimicked users interacting with TeamBandit. In our research we came across Selenium and PlayWright which both offered tools for writing tests to mimic user interaction with TeamBandit. We ended up choosing to work with Selenium over PlayWright as PlayWright is a new tool that is missing a lot of tools that its competitor Selenium offers. Members of our group also had prior experience with Selenium, reducing the overhead required to develop well rounded tests, an important factor when

considering our limited development window. Playwright also uses a headless browser to perform tests which can be helpful in certain circumstances but robs developers of the ability to watch potential errors play out in real time, a valuable tool in gathering quick knowledge about an issue. Selenium allows us to write tests that closely mock what users will be doing with our software, simulating mass user testing to ensure high loads among other things do not cause issues with our software.

In order to properly test TeamBandit for proper use case outcomes, we identified what units were most important in testing for our application. These units include:

- User Login
- User Registration
- Adding Data and Manipulating It
- Team Assignment
- Student Features
- Team Lead Features

These units cover a majority of what our users will actually be doing when they interact with TeamBandit and we will now detail how we will test each unit individually.

2.a. User Login

This unit covers the act of a student, organizer, and mentor user logging into our application. To properly test this unit, we came up with three main tests, these tests include:

1. **Logging in as an Existing User**
   This test is meant to verify that an existing user can sign into the application and that it properly identifies what type of user is signing in, whether it be an organizer, student or mentor.
2. **Logging in with False Credentials**
   This test is meant to verify that someone who doesn't have an account with TeamBandit cannot log in. There are two key features we wanted to check, logging in as a user who does not yet exist and logging in with an invalid password information.

2.b. User Registration

This unit covers the act of an organizer registering to utilize our application. To properly test this unit, we came up with three main tests, these tests include:

1. **Registering as a New User**
   For this test we simply wanted to create a new organizer user in order to test our database’s capability to process larger amounts of data. This consisted of

creating dummy information and utilizing Selenium to create new users en masse.

2. **Registering As an Existing User**
   For this test we wanted to register with existing organizer information in order to test potential conflicts and how our database schema handles them. To properly test this we needed to pull a list of registered organizer information to use.
3. **Registering with Invalid Information**
   For this test we wanted to input invalid information such as an invalid email format to ensure users are always alerted as to why their account may not have been created.

2.c. Adding Data and Manipulating It

This unit covers all of the data that the organizer can add/edit. There are a lot of data entries including, courses, clients, projects, students, mentors, assignments, and schedules. To properly test this unit, we came up with two main tests that we just duplicated for all of these data points, these tests include:

1. **Add Data Entry**
   This test is meant to verify that when proper fields are filled out, the corresponding data entry will be filled out accordingly. This is to make sure that when adding a new course, client, project, student, mentor, assignment, and schedules the proper information is assigned to their respective data.
2. **Edit Data Entry**
   Sometimes data can change or might not have been inputted correctly from the start. This test is meant to verify that organizers can change the data fields and that they will properly update and display the new data.

2.d. Student Features

Students have several features that allow them to perform their tasks throughout the semester, many of which need to be tested to ensure that no one is unable to do them, potentially damaging their grade.

1. **Choosing Project Preferences**
   The project preferences module has gone through many iterations to reduce its erratic nature but will still need to be tested to ensure that rapid and frequent changes can be tolerated by the database. This will consist of a Selenium script utilized across multiple user accounts, potentially simultaneously, to ensure that our database can process all changes.
2. **Submitting Assignments**

Submitting assignments will need an automated test to ensure that revisions can be submitted multiple times as students should be able to change their submissions.

2.e. Team Lead Features

This unit encompasses all of the features available to the Team Lead. This is a student role with more meta capabilities pertaining to the team, including changing logos, names, and team website information. Some of the tests developed for this role include:

1. **Altering Team Website Information**
   Throughout the semester, teams are expected to keep a meta information website about their project updated with current data, resulting in many changes to it over the course of the semester. A Selenium test will be written to simulate this, rapidly updating documentation, descriptions, and other areas of the Team Website and checking to see that these updates are reflected on the live version in a timely manner.
2. **Updating Team Meta Information**
   Team Leads will be expected to upload key information at the start of the semester such as PDFs, Team Descriptions, and Team Roles. These will be susceptible to change, thus requiring a Selenium test to make sure successive changes to this information does not cause issues.
3. **Organizer Changing Team Lead**
   Team Leads can and undoubtedly will change throughout the course, requiring a new student to be assigned this role by the organizer. We will create Selenium tests to simulate this, having an organizer account change the role to another student and logging into said account to make sure the abilities of this role have been transferred.

3 Integration Testing

This web application contains many different modules that cohesively interact with each other to ensure that a dynamic and single page application is generated to the user. In order to confirm that these modules are correctly interacting with each other, their integration needs to be tested. Integration testing is a type of testing where software modules are tested as a group. The overall purpose that this testing serves is to inform the development team of any possible defects the system experiences during the interaction of the modules. Before describing how this integration will be tested, the modules will be briefly detailed below.

- 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 and mentors, to a project.

It is just as important to understand how each of the above modules interact with each other. Below is a diagram of the high level architecture of the web application.

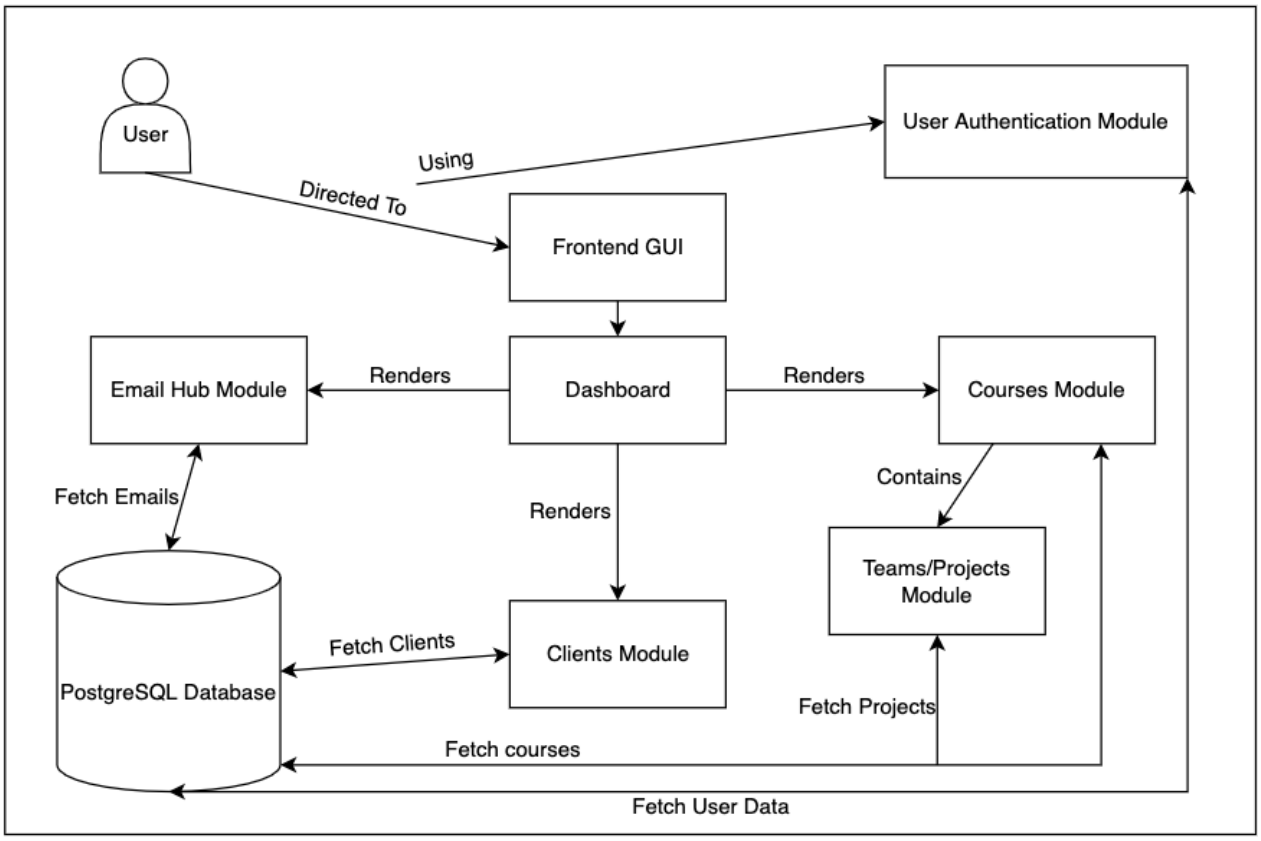

Figure 3: Diagram showing the high level architecture of the web application

Before any testing implementations are detailed, this section will first detail JSON Web Tokens, user identifiers, and the process the user must go through to have these authentication items associated in their browser session. From there, page load confirmation will be detailed.

First, the integration from User Authentication, or user sign-in/sign-up, to the main TeamBandit page will need to be tested. It is worth noting that this initial module interaction is the most important aspect of integration testing for this application as the user will be associated with a JSON Web Token (JWT), which can be continuously referenced in the web application's internal API made with Express, the back end framework for Node.js for testing and security purposes. Hence, there is no need for any other outside testing library for any integration testing related to user access. For example, when a user registers an account for TeamBandit, the data they input, such as their name, email, and password is sent to the back end of the application and will need to follow these steps:

1. An HTTP POST request is made to the API where the input data from the user is deconstructed to test individual aspects of the data.
2. Check if the user is already registered in the database by inspecting all emails already registered.
    a. If the user is already registered, the user receives a simple response from the back end that informs the user that they must use a different email address.
    b. Otherwise, the user's information is stored in the database with their hashed password.
3. At this point, the user is assigned a JWT that is associated with them for a period of time to authenticate their access privileges.

The process of a user signing into the web application is largely similar to the process explained above, but succeeds if a user's information is already in the database. Moments before the user is signed in, the JWT is stored in the browser's local storage, along with a user identifier that is associated with the JWT. The purpose of the user identifier is to allow the web application to display the proper information to the user with the proper credentials. An example of the steps that verify this process are:

1. A course organizer selects a course on their courses page.
2. The course opens and displays the proper tabs of information in the course. These include Projects, Schedule, Students, Mentors, and Settings as an example.
3. The Settings tab will only appear for an organizer as the web application can easily read that the user's identifier is an organizer, and can verify that the organizer is legitimate through their JWT.

The process above is the same for all aspects of the web application that require the display, adding, editing, or deleting of sensitive information.

Now that the basic foundations of how a user is authenticated has been detailed, an explanation of how to confirm that a new module has just been loaded to the page will follow.

The front end framework, React, is a component based application library that simply compiles and places all components into a single HTML file. Unlike the use of JWTs for authentication, ensuring that a new module has loaded will require the use of the React Testing Library. In order to correctly utilize this library to test page loading, Document Object Model (DOM) elements of the final HTML file can be tested with this library. Examples of DOM elements include div tags, body tags, as well as any other HTML tags. In React, developers can essentially build their own tags made up of simpler tags. These developer-built tags are the components. The components can simply be tested with React Testing Library's render function. With the render function, a component can simply be passed in as an argument to ensure that all inner elements are mounted, ensuring that the component is fully mounted.

Now that individual unit tests have been done, as well as the completion of larger picture integration tests, usability testing will now be detailed to complete the software testing process for TeamBandit.

4 Usability Testing

As it pertains to the usability of TeamBandit, the process for testing the application's usability is with respect to the end user's experience in carrying out the intended workflow. The objective of testing the usability of the system is to examine, measure, and (re)assess the quality of the user interface given the technical capabilities and general patience of the target end user(s). This stage of testing is designed to provide a clear understanding of which elements (or lack thereof) serve to positively impact the application's ease of use and/or inhibit the experience of deliberate interactions.

The objectives of conducting usability testing are to acquire a relevant set of data samples to be used:

- as a practical baseline in reforming select areas of the user interface.
- to draw conclusions about the system's understandability.
- to examine deficiencies in the design's addressing of the intended audience.

This manner of testing is conducted by first assembling a sample group of real people who fit one or more elements of the criteria initially set to define the consumer base the product was built for. The individuals included in the sample will then be

provided the current viable product with no instructions beyond what would be available within the finished application itself. They will then attempt to use the product for a practical workflow, discovering and inferring how each major component of the system works. This process is typically done under direct observation or without direct observation but followed by a survey or collection of input from the individual.

The development of the above procedure's specific adaptation for the purposes of TeamBandit was considered with emphasis on two primary characteristics:

- background of end users.
- assumptions/expectations of end users based on standards set by similar products. This may encompass:
    - Interaction time
    - Familiar/relatable interface design
    - Quantity of physical actions necessary to conduct an action (click, type, etc.)

The procedure to identify and enlist the participation of users for the testing process needed to primarily consider the criterion of the background of end users. Doing so permitted us to define practical spectrums for both the anticipated extent of technical capability and the level of knowledge pertaining to the workflow of managing team-based courses. These spectrums serve as the reasonable stipulations for selection of test users to adequately represent course organizers; however there is an additional end user which interacts with the system for alternative purposes that must be taken into consideration: *students*.

Although the structure of the user interface is essentially uniform between the system's organizer and student views, there are variations in application use cases and available modes of user interaction. The lack of change in the physical arrangement of the user interface negates the need to distinguish assumptions/expectations measurements between the two end users. For this reason, this area of criteria can be analyzed independently as an overall system evaluation. Therefore, only the background of these two end user groups must be considered separately.

The typical background of the organizer user group is expected to be an instructor, likely but not necessarily instructing in a technical field. This instructor must be the organizer of a course who is responsible for arranging assignments, project and team selections, and communications with internal and external parties; because the instructor will certainly be the course organizer, we can safely generalize that the instructor is capable of broadly defining a course *plan*- TeamBandit is the tool to put such a plan into action. A team-based course plan inherently includes a definition of the

project genre (e.g. software engineering), a tentative schedule pace, and communication with affiliated parties. Certainty that this organizer end user will be capable of this supports the assumption that their background will include familiarity with the fundamental components present in TeamBandit such as the communication portal, schedule and deliverables, and project assignment. As a result we need not consider the end user's background in fundamental concepts of course organization but instead only their background's impact on simplicity of use. We will proceed to testing with the most recent version of TeamBandit which consists of what we believe to be a self-explanatory user interface appropriate for even users lacking technical expertise.

Student users' background is expected to be lacking in course organization concepts but equally as capable of operating the interface due to the current simplicity as stated above. However, we determined that lack of student experience with the core concepts of course organization are supplemented by the fact that their instructor is guiding the course, and hence can make any clarifications to uncertainty of the user interface just as an instructor would introduce a student to any other web application used for academic purposes.

To assemble appropriate sample end users to represent course organizers, and to account for the variability spanning the spectrum of these users, we intend to enlist the help of two college/university faculty members with experience organizing team-based courses or projects as well as instructors who have no prior experience overseeing a team-based course but do have experience planning typical classes. This will provide a more precise analysis into potential differences in usability relative to prior experience.

In the case of student sample users, we will select individuals who have used online academic applications before as well as individuals who have not. This is to reflect the experience of both end users who have familiarity working with online academic applications and new or first time users of this type of application. We can then determine if TeamBandit's usability is well-enough suited for users of any experience level.

The usability regime will be conducted in the manner described below, in order as written. The entire process will be observed by a designated team member:

1) Sample end user will be asked to verbally express each step of their thought process, or "think aloud". This user will be asked for honest feedback and informed that both positive and negative feedback is welcomed and beneficial to the refinement of the final product.

2) Sample end user will be given access to TeamBandit and provided with a use case scenario.
3) Sample user will be asked to carry out this use case, having never used the application before.
4) The observer will take note of everything that is said and/or critiqued by the sample end user.
5) Following the end of the use case scenario, the sample end user will be asked if there was anything they found particularly confusing or they feel could be made more clear about the user interface.

This will be repeated with each end user and adjusted according to their end user role. For example, a student end user will be given a use case to be carried out in the "student-side" of TeamBandit. The resulting data will be analyzed by the entirety of the development team to examine any areas of the application that repeatedly confuse different users, with the goal of identifying any trends that suggest something may need to be refined. Then, any unique issues or encounters will be examined to check for possible areas of refinement that had not initially been considered.

For additional insight, we will also seek out a review of the application by two expert software/web developers. These reviews will be used to supplement the actual sample user testing and serve as reassurance if the team has any uncertainty pertaining to a particular concern.

5 Conclusion

Software testing is a necessity in projects on the scale of TeamBandit, especially on a time scale such as ours where bug fixing or UX may take a back seat during development. This is all the more true for user facing applications, where hundreds or even thousands of users may be putting the software to the test.

Our testing plan seeks to cover all bases, from fixing large bugs to righting difficult use-cases for new users. With our limited development time we have sought to utilize popular testing modules to perform our examination into potential failings of our application. Selenium, our main tool, will act as a form of high efficiency user testing, looking to find vulnerabilities by quickly performing front-end tests using randomized data. Our integration testing seeks to use create-react-app's boilerplate testing library in order to test how the different modules of the application will work together under high stress, and our user testing seeks to not only uncover any remaining bugs but alert us to any convoluted or difficult parts of the application that we as developers may have overlooked. By utilizing these three different test types we hope to have a cohesive testing plan in place.

With this plan we aim to target any of these bugs and inefficiencies over the course of the next month as our software development time comes to a close and by doing so deliver a bug free and easy to use experience to our client, Dr. Doerry.