1. Test Plans

1. Unit Testing

Unit testing will take place on each individual component using Mocha: a feature-rich JavaScript test framework running on Node.js and in the browser [3]. Mocha ensures asynchronous testing of each component as well as flexible and accurate mapping of uncaught expressions to the correct test cases. Tests in this framework would run serially using an assertion library called chai. This would ensure verification of the test results. The testing activity will take place in the app.js component of the client as well as server side of the application. In order to conduct the testing - beforeEach() and afterEach() test hooks will pass each test case in the script, these two hooks would encapsulate the <div> element using which the components would get mounted. To define the test case the describe() function would be defined and then the it() function will be used for function calls to identify each individual test case. To run the test a babel-register which is a developer dependency would be installed as a plugin. The babel register would store .babelrc file in the project root and add the contents in the form of a json document in it.

2. Integration Testing

Integration testing will be conducted between components of the code that interact with each other, such as interaction between multiple services like the API and database and the interaction between react components that work together in a redux environment. For conducting these tests, the framework required includes a combination of Enzyme and Jest [4]. Jest runs in Node and uses jsdom which simulates a DOM environment. The objective of our application is to use MaterialUI and react-bootstrap for styling, but some components of our application might involve the use of CSS. In that case, services like saucelabs would allow us to test that the layouts are looking good across multiple browsers. The tests in this framework would run serially using Jest as an assertion library, which would ensure verification of the test results. The following Jest functions will be utilized:

makeMountRender — a util function that accepts a component and some default props. It returns a function so that we can override the default props if desired. This function returns a react wrapper using enzymes mount.

makeStore — a util function that is used to create a new redux store. We import the rootReducer and the createStoreWithMiddleWare function from our actual application which makes it easy to create a new store for testing purposes.

reduxify — a util function takes a component, props and an initial redux state and returns a new react component which renders the passed in component in a redux environment.

snapshotify — a util function that accepts an enzyme react wrapper returning an HTML string.

We pass our component into **reduxify** and then pass that into **makeMountRender**. The result is an enzyme react wrapper that can be passed to **snapshotify**.

3. System Testing

The objective of our system testing is to check the integration of the components, check that the code runs in a real browser, and to handle cross browser JS concerns. In order to replicate a real browser, a tool called Karma will be used. The reason for choosing Karma is that its testing framework spreads across multiple frameworks, including the ones which we have utilized for our integration and unit testing–Mocha, Enzyme and Jest. On top of that, React web applications need to be versatile across various browsers. Therefore, in order to check cross-browser versatility of modern frameworks, a transpiler called Babel will be used. This will ensure that old JS engines do not face issues in utilizing modern features of the application.

1. Functional Testing

The objective of our functional testing is to test the core features of our application, each feature in this case would be tested for all its components using shallow methodologies of enzyme and mounting methodologies using Jest and the inbuilt react testing libraries [5]. On top of that, project tracking, project planning, and project data storage would be separately tested using the inbuilt react test libraries to ensure that they are working as per the requirements of the system. The various layers of testing for each component would be recorded and all the issues would be added to the kanban board so that the developer who performed the test could add to the team repository and assign the team member whose work is responsible for the given error. That team member must then rectify the bug and write appropriate tests to catch the bug in the future.

2. Performance Testing

The objective of our performance testing is to check the code for its qualitative aspects. The team's goals for achieving the non-functional requirements consists of using object-oriented software principles to ensure reusability, performance, scalability, manageability, and maintainability. In order to confirm that the code is in correspondence with these requirements, a static code analyzing tool will be used. This will ensure that the team uses one coding style and avoids known anti-patterns in development. Arguably, one of the best lint tools for our project would be ESLint due to its support for a variety of plugins to extend the functionality. It also has rich easy-to-use documentation [6]. The create-react-app command (used for creating a react application) comes with ESLint already installed and is inside the package.json under the eslistConfig object. Another non-functional requirement is portability. In order to ensure that we can port the application to a mobile based application in future, the use of Karma will take place. Karma supports cross browser as well as mobile-app compatibility testing frameworks for our code.

3. User Acceptance Testing

User acceptance testing will be carried out during the demo for our projects. This would be an opportunity for our team to test our prototypes over various phases of the software development cycle. The essential requirements of the user acceptance testing would include, project creation, project management (creation, deletion, and editing) and fever chart status. This would not be the only limit to user testing, as during the testing phase our objective would be to get feedback from users on possible unknown issues. The objective of this testing is to make the application as user-friendly as possible. The process for evaluation would be getting feedback from users, if a bug is found or there is user feedback to add a feature it would be created as an issue on github and a team member would be assigned to patch the bug or create a new feature.