Convert an Industry-Leading Native Mobile App to React Native

Design Document

Team Number: sdmay19-02

Client: Buildertrend

Adviser: Mai Zheng

Victor Amupitan -- Chief Engineer of Design

Lucas Kern -- Executive Meeting Facilitator

Michielu Menning -- Lead Report Manager

Kyle Nordstrom -- Co-Team Lead/Meeting Scribe

Francis San Filippo -- Scrum Master

Walter Seymour -- Co-Team Lead/ Team Communications Leader

Team Email: sdmay19-02@iastate.edu

Team Website: https://sdmay19-02.sd.ece.iastate.edu

Revised: December 2nd, 2018 / Version 2

Table of Contents

List of figures/tables/symbols/definitions

0.1 List of Figures

0.2 List of Definitions

- **BT:** Buildertrend, our Client.
- **Components:** Unit of code that deals with a specific feature or functionality. Components break apart the project into smaller subtasks.
- **Component-Based Development:** A software development process that emphasizes the separation of concerns throughout the project.
- **DRY programming:** Don't repeat yourself programming (i.e. duplicating logic)
- **Front-end:** The part of the development that deals with converting data into a graphical interface for users to interact with.
- **Jake:** Director of Software Development at Buildertrend.
- **Native:** Software that is developed for use on a particular platform or device.
- **React:** A javascript library for building user interfaces.
- **React Native:** A framework for building native applications with React.
- **React Router:** Specifies the components that will be displayed with certain routes.
- **Redux:** A predictable state container for JavaScript applications.
- **SaaS:** Software as a service
- **Software Architecture:** High-level structures of a software system.

1 Introductory Material

1.1 Acknowledgement

We would like to give a special thanks to the following members of Iowa State and Buildertrend for the assistance they have provided throughout our project.

- Rich Kalasky
- Daric Teske
- Mai Zheng

1.2 Problem Statement

The problem that Buildertrend currently faces is that the mobile application that they use could be created in a better way. One problem is that they currently are updating and maintaining two applications, one for iOS and one for Android. Another problem is that they need specialized developers to work on these that have iOS or Android development experience. These skills are not necessarily common for Software Engineers, so it is a little difficult to find these people. It also makes the software harder to maintain. Maintaining two applications that are in more obscure languages is costing Buildertrend time and money. They are looking for a way to save money, time, and make the overall development process easier in the future.

For the solution, Buildertrend looked to us. They had the idea to recreate the application with React Native. They put together a proposal for a senior design project at Iowa State so students could help make this transition. A React Native application can solve all of these problems. With React Native, the JS code can be translated into Native code for both IOS and Android. There only needs to be one application, and this means that drastically cut the number of resources needed to maintain and update the application. In addition, Javascript and component-based programming is a far more common skill. This means that more developers will be able to understand the code better and begin working on the project much faster. Furthermore, by transitioning into React components, the project code base will be much cleaner. Reusing and managing components makes for a more efficient and maintainable development process.

1.3 Operating Environment

This is a mobile application, and the main users are construction workers. This means the environment that the application will be used in may vary greatly. Because these construction sites are sometimes in remote locations that may have a poor internet connection, the efficiency of the application is key. It is also possible that the users may not always have direct access to the device itself. This is in part due to the nature of the industry. With massive amounts of manual labor being done on-site, it may not be safe or viable for them to be on their phone actively checking the application. Likewise, construction sites are not the only place the application will be used. The application is used by many project managers off-site or in an office setting. This means that it will be used in places with more predictable conditions. Therefore, the application needs to perform consistently and serve as a productive tool for the construction industry.

Overall, there does not seem to be any direct safety hazards, other than the general dangers of using a mobile device (ie texting and driving). When it comes to using this application, the environment may impact the quality of the product. For this reason, we must consider the variety of uses and environments the application will be subjected to.

1.4 Intended Users and Intended Uses

There is a multitude of different types of users that will be working with the application. These include:

- Construction Managers - These are the people who will be working with projects, but also have capabilities that the average user may not have.
- Construction Workers - Probably the most common users. They will have less access than the managers.
- Clients - People who paid for the construction job. They are able to track their jobs and request changes.
- Sales Employees - They will be able to track the projects that they are assigned to. They will also have access to internal features.
- Developers - Buildertrend Developers will also have access to the application. They will have their own specific features to oversee the product.

The construction managers use of the application will be both to manage the project and to manage the people under them. These uses will be to keep the construction projects on schedule and to make sure employees are up to speed on project expectations. The workers themselves will be able to use the app to handle tasks they have completed, clock into work, and be in contact with the manager. In addition, the construction managers will be able to use the application to track leads and handle financial issues.

The client's main use of the application will be to oversee the progress of their project(s). They are also able to contact the construction manager if regarding any inquiries about change orders. They should also be should also be capable of communicating fluidly with the project manager in case any legal agreements documents require a signature.

Sales employees will be able to use the application to track their project managers and make sure that they are not having any issues with the application. Above the sales team, the developers will have access to virtually any functionality. This is to account for any testing and adjustments they may want to explore.1.5 Assumptions and Limitations

- Assumptions
  - Users will have access to a mobile device.
  - The primary user will be construction managers and workers.
  - The backend of the application, which will not be provided to us, will handle all security measures.
  - The backend will handle any financial transactions or legal transactions between the builder and clients.
  - The backend will give us data in an efficient and clean way.
  - The backend of the application will handle all the accepted different file extensions.
  - No one has access to the backend code besides us and employees of Buildertrend.
  - All margins and styles will be standard to cell phones.
  - Code will be tested by the QA at Buildertrend.
- Limitations
  - The speed of the product relies on the phone it is on and some old phones may not handle the software as well.
  - Usage of the application depends on a solid connection to the internet.
  - Many of the developers in our groups do not have much experience with React, Redux, and some other tools.
  - Clients pay a lot of money to use the product, so the product must be at a high standard.
  - The product must be completed by the end of the second semester.
  - The functionality of the application must match the functionality of the current application.

1.6 Expected End Product and Other Deliverables

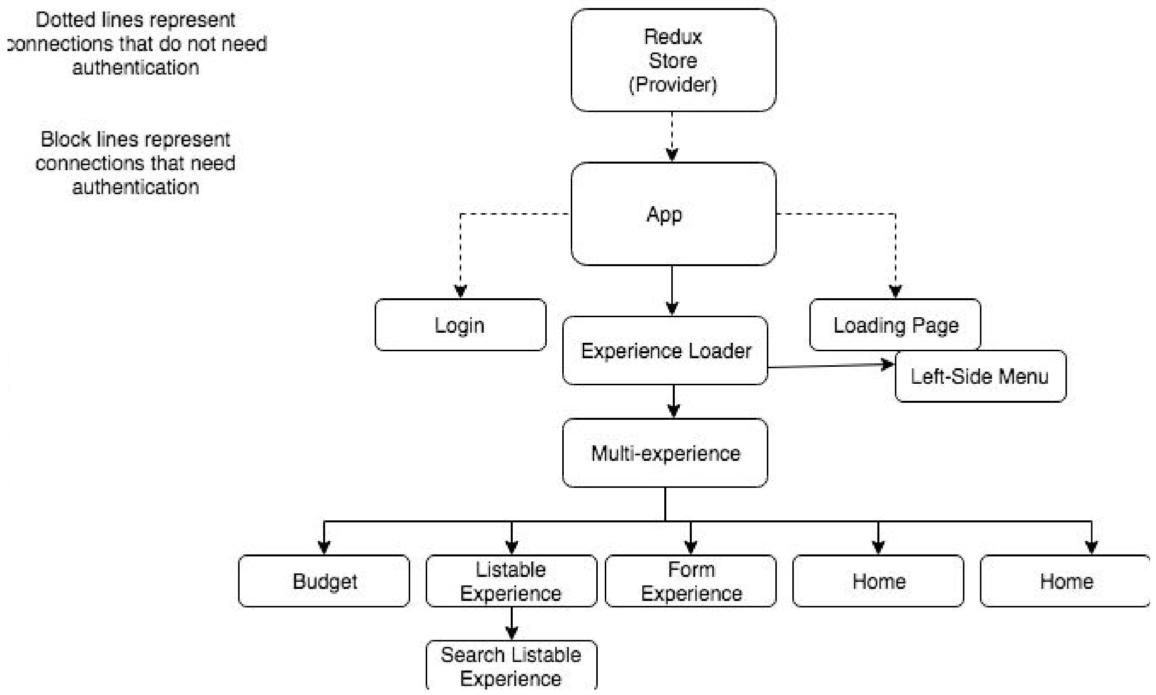
Throughout the semester, we will have various documents completed throughout the development cycle that will be minor deliverables. Some of these delivered items will include weekly reports, schedules, and documentation. Moreover, our main deliverables will be as follow:

- **Initial Prototype** - This will be a fully functioning React Native project. It will have a functioning menu, and the user will be able to pull up the list of components. There may be a few subcomponents completed, but the main purpose is to establish user movement throughout the application. This will lay out the foundation for the following deliverables and will serve as a framework for further development.
- **Structured Prototype** - This deliverable will be a project management application with the core features implemented. It will not be a completed version of the initial prototype, but rather a partially finished version. It will have all the same functionality and structure, but some of the components will still require completion. In order to fill the necessary features, the components within the existing application will be ordered by priority. This prototype is designed to gauge our progress with a semester of learning many new technologies. There will be a demo with our client, and this will demonstrate both our progress with the application and our improvement with these new skills. This also lays the groundwork for future development and gives us flexibility based off of client feedback. We deemed this to be a necessary intermediate step in order to verify that what we are creating aligns with Buildertrend's intended goals.
- **Final Prototype** - This is the final product that we will produce for Buildertrend, upgrading it from the structured prototype, it will have all the features the original application has. We will have all the existing components built out and fully functioning. There is potential that we may add new features as the client and we see fit.
- **Documentation** - This will be the API for the public functions that we use while making the product. This will be used for the people who will be working on the project in the future. It is also be used for developers at the company who may have questions about the application.

We estimate completion and delivery of this project to be May 2019, the end of the Spring 2019 semester.

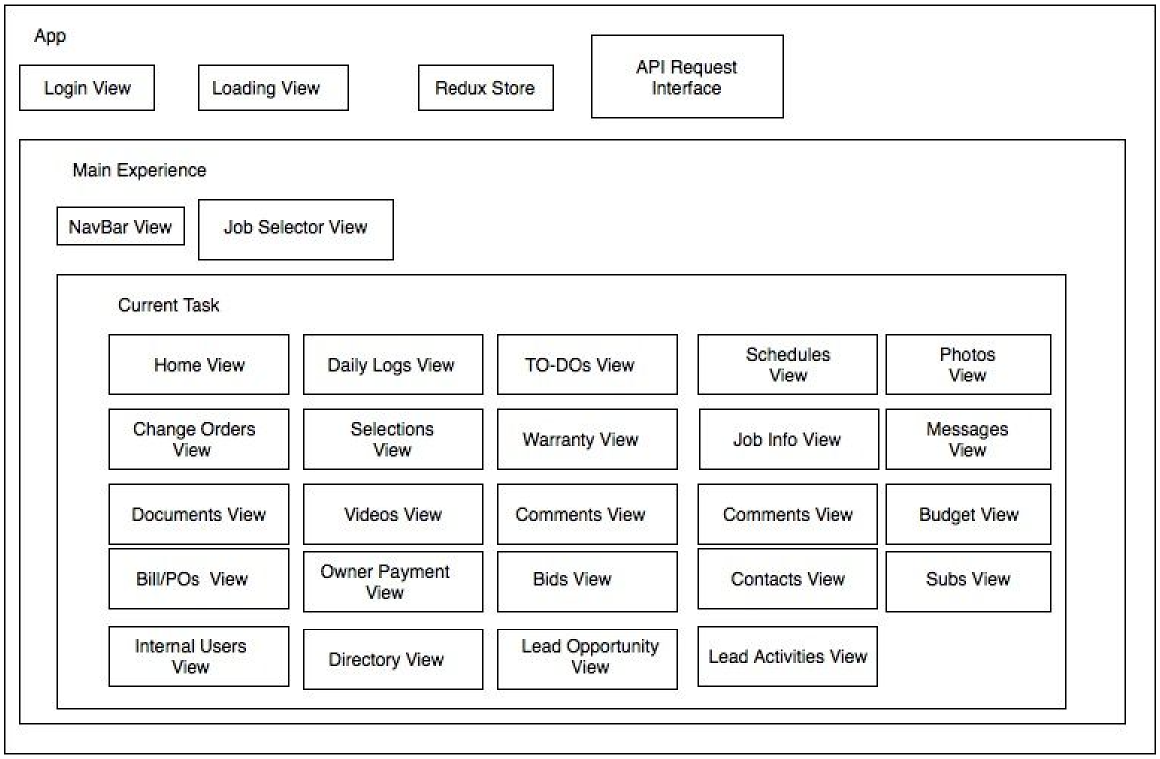
2 Specifications and Analysis

2. 1 Proposed Design


Dotted lines represent connections that do not need authentication
Block lines represent connections that need authentication
Redux Store (Provider)
App
Login
Experience Loader
Loading Page
Left-Side Menu
Multi-experience
Budget
Listable Experience
Form Experience
Home
Home
Search Listable Experience

This chart contains the data flow from the application. It starts out from the top Redux Store and trickles its way down to each part. The App is the main application, which contains a login, loading, and experience loader page.

Experience Loader is where most of the application takes place. This page is where all the user interaction happens. Each of the Multi-experience children loads up an actual screen of the application. By observing the flow, one can clearly derive how the flow comes from the Redux Store all the way down to each individual screen.



Our strategy for the architecture includes using a component-based architecture and central store to save the application state. A component-based architecture is simply splitting up the application to smaller reusable pieces that are independent of each other. By having smaller, independent pieces, it allows the developers to be able to reuse and update quicker. Overall, a component-based architecture allows the pieces of functionality to be replaceable, independent, reusable, extensible, and not context specific. This will make our code DRY (Don't Repeat Yourself) and easily modifiable. The purpose of this project is to help combine two code bases for mobile development. Having a component-based architecture will ensure the modifiability of a unified code base.

The layout of the application consists of a login page and a main experience. Once a user is authenticated, the main function of the app is loaded. At all times, the navbar and job selector is available along with the currently viewed component. There will be three components contained in the main experience: the navbar view, the job selector view and the current task. A current task can be anything displayed other than the navbar and job selector. The current task will be a container that will handle loading and switching between selected tasks.

What happens inside of a component depends on the state of the application. To save the state and have it accessible to all the other components, we will use a redux store. Also, to centralize all API(application program interface) calls, an API interface will be at the top level so that it is accessible to all other components. This happens because the top level passes down properties down to it's lower components. By having the API interface on the top layer, it will naturally pass down the calls to all of its children components. The API interface will also interact with the store to save data that another component may already have loaded.

Each component inside the current task will have subcomponents, smaller components, and additional functionality. Many of the modules can be instantiated with the same components. For example, the search feature is common for many modules within the app. We will create one search component that can be used in each of the modules. The functionality of the component will replicate that of the original application.

2.2 Design Analysis

As a team, we've been able to come up with processes and an architecture that we are going to use. The process we came up with is an OKR(Objectives and Key Results) process. Essentially, we plan out our design into objectives. The objectives are easy statements that can be graded based on an achieved or not achieved binary scale. We are using this process to help us keep to our design and plan so that we can make and observe achievements, and also see what we're doing right or wrong. This leads to the second part of the OKR called Key Results. Key Results are children of Objectives. Key Results further expand the OKR. They include a practical way in which the Objective is going to be achieved. With the Key Results, we can measure the success of a particular task since each key result represents a major task and also the overall success of the Objective (essentially if all Key Results under that Objective were successful). That being said, the OKRs span for about three weeks to four weeks. We have also come up with an architecture for the application. We have yet to start major implementation but we have already started creating tasks for implementation of the architecture. The application was divided into the UI components and data-moving components. The UI Components (also known as Presentational Components) and the user interface components that are visual to the user while the data-moving components are mainly to control and move data to the appropriate UI component.

We are mostly in the beginning part of out process and can't really judge the process to see whether or not it work. As we are still in the testing stage, we can say it has been working so far and we are likely going to adapt it till the end of the project since we predict it will continue to work. We think so because it has helped us create guidelines on the work that needs to be done and we've completed our tasks based on that. We have already mentioned our observations on the process, but we have some likely ideas to modify in the architecture. As we start implementation, there are parts of the architecture that might not flow as expected, a more realistic example is in reusability of code - right now we plan on using higher-order components to create reusable components to maintain our component-based architecture but eventually, we might have some compromise and decide to use an object-oriented approach in some parts to improve the code reusability.

One of the strengths that we've observed so far from our design is the objectives. Keeping things objective helps in performance estimation i.e. you know when something is done. This also helps with planning because we know exactly what has been accomplished at every point in time and can project and estimate what needs to be done and when. Another strength is the ability to have multiple working together almost seamlessly. With the component-based architecture, we can create a guideline (or component API that classes of components have to adhere to), that way people can work on a component without having to wait for dependency components to be created. They can just use the API or maybe a stub as a black box for the dependency. The seamlessness comes in when dependent and dependency components have been created, they would just work together since they were developed to meet a specific guideline as mentioned above, and there would be no work needed in connecting them. The weakness side to this is the overhead that has to be done before we can get the whole process going. We have to come up with frameworks, structures, and guidelines as mentioned above before the component development can be started. It seems like a fair trade-off since it removes some hectic work from the future.

3 Testing and Implementation

3.1 Interface Specifications

Our project is completely software and has plenty of interfacing that would require testing for our project. The main interfacing that will need to get testing is the interfacing with the software to the phone. Our application should be able to handle all sorts of smartphones with different dimensions and other specs.

The main way we will be testing the interfacing is through an application called *Expo*. *Expo* allows the developers to test a running version of the application on their own phone. To do this, each tester has to have access to the internet and download the *Expo* application. In the application, the user has to scan a QR Code that is generated to have a running copy of the application on their phone.

Expo allows each developer to manually test the application, which is a big part of the interfacing framework. However, manual testing makes it difficult to find faults in the software, which leads us to *React Dev Tools*.

React Dev Tools is a debugging tool that allows each developer to step through their application, continually checking on the internal states to ensure that we are able to catch any inconsistencies.

Overall, *Expo* allows us to manually test the interfacing through our application and various phones while *React Dev Tools* gives us the chance to go into the code once we find some inconsistencies or unintended responses.

3.2 Hardware and software

Jest - Jest is a framework for TypeScript that allows easy to make test cases in JS Code. This software was developed by Facebook, so it especially works well with applications that use React, since Facebook also created React. Jest has many useful features. Jest is easy to implement into a new or existing project using Node Package Manager. Jest uses coverage reports to allow us to know how much of the code is covered and breaks it into statements and branches. Another, great tool that comes with Jest is Snapshot Testing. This is great for regression testing because it tests if a component has the same state as the last time the component was tested. It also has a huge list of assertions that can become helpful in the testing process. In conclusion, this framework is exceptionally good with React.

Enzyme - Enzyme is a library made by AirBnB that is specifically made for React. Since React uses components, Enzyme can create shallow or deep copies of the components. It also has ways to test relationships between components. With Enzyme, it makes testing the functionality of independent components much easier. It also makes the state of the component much easier to handle while testing.

Since our project does not use any hardware besides a phone that runs our application, we really do not need to focus on hardware testing. We described how we use interfaces to test the functionality on a phone in the previous section, but there is no specific hardware we use to test our application.

3.3 Functional Testing

Unit Testing - We create individual test files for each component. Here, we will attempt to have 100% coverage for all components. This includes statements and branches with component functionality. Snapshot tests will also help us make sure that each component continuously works throughout the development process.

Integration Testing - Once we have all the individual components and functions working to an acceptable amount by our Unit Testing, we will test larger components and how they interact with the entire application. This can be done with intensive manual testing through *Expo*.

System Testing - We do not have a complete system, but eventually we will use Buildertrend's staff to system test the application. They will be able to test some of the functionality with the component tests that we have built out. The QA staff will actually be able to test the requirements of the prototype we give them. Since Buildertrend gave us functional and nonfunctional requirements for the process, they will be able to easily test these requirements.

Acceptance Testing - Once the Unit, integration, and system testings are at an acceptable coverage percentage, we will set up test mocks to make sure our application makes the appropriate backend requests. Finally, we will once again rely on Buildertrend's QA staff since they will have dedicated people on this stuff.

3.4 Non-Functional Testing

Testing the Non-Functional requirements are very important in a project. Some non-functional requirements that we highlighted are performance, security, usability, and compatibility.

Performance: We will use *Expo* to test the performance of the application. We will set test requirements for performance times for each functionality and component. We can only test so much of the performance since we are not given any backend source code. So every request is out of our scope.

Security: When it comes to security, this is not really one of our worries. The backend API for the entire application is already created. Buildertrend already has secure databases where this information will be stored. The data is being securely handled on the Backend by the developers at Buildertrend. All we need to do is to make sure any sensitive data is being encrypted. We can test to make sure that our encryption functions work with Jest.

Usability: The application should be user-friendly and easy to use without a detailed explanation of how the application works. Since we do not have much control over the application is formatted, so our usability testing comes down to *Expo* and manual testing on different phone frameworks. Having a running application running on each of our phones and using that to test different models will cover our usability test requirements.

Compatibility: Compatibility comes down to how our application integrates well with BT’s backend API calls. We can set up a test mock to simulate the calls we can sending and compare it to the calls the old application were sending out.

3.5 Process

Our method of using a component-based architecture will allow us to build components that satisfy basic needs and then use these components to build bigger components. We use two diagrams to demonstrate the design possibilities. The first design shows how our application will work together with a big idea of the components that will be needed. The second design goes into more detail by showing the smaller components that will be put into other components to complete them and eventually the application. Software design patterns have been tested by many and reproduced by more. The component-based design architecture is a process that has worked in the past and will be a good architecture for us to use to solve our problem. We will need strong components to implement this method which will help us to easily convert an industry leading iOS and Android architecture into a React Native application.

The processes that we came up with for our solution will involve implementation, testing, and refactoring. This will occur by attempting to create the needed components that will accomplish basic jobs. As we advance and implement more of the project then we will begin to start testing our components and make sure they accomplish the tasks that need to be accomplished. After testing we will continue to implement more components and use previous components to create a better foundation that we will be able to incrementally build off of. As this process continues there will be some refactoring as we change things to be more efficient or realize we need more capabilities or less functionality for certain components.

3.6 Results

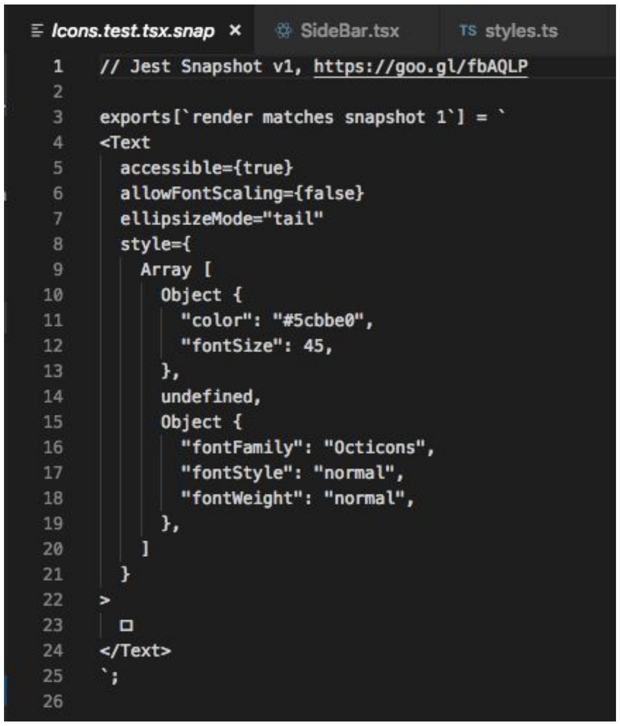
As of now, we have done extensive research and tactical communication amongst our diverse team members. We have successfully looked into testing frameworks in *Jest* and *Enzyme*.

We have successfully integrated our project with *Typescript* and *Expo*. Having these successfully integrated allows every developer to incrementally test their progress. We are currently able to test functionality we currently have implemented with Expo, seeing how it renders on an iPhone or an Android. We also have a Python script to create components, and with this comes a test file for each component. Each test file currently has a very basic test, but it is a framework to enhance our testing and coverage.

Our current challenge is that we have not had time to implement our tests. Our client is focused on us creating functionality first, and then creating tests. Our main focus has not been testing but getting the development process going. When it comes to testing, we are focused are on setting our application up in a way where testing will be easy in the future.

3.6.1 MODELING AND SIMULATION

We use a testing framework called Jest to model our application component. This stores the state of each component and records state changes when transitions are made during tests. This is used for regression testing and continuous integration since these models are used to check that components still have the correct properties. These models are snapshots from Jest snapshot testing. We can also create simulations of functionality with mocks and stubs. We can make sure that functions are actually being called without having to import the functionality. This is a great way to simulate key functionality throughout the development process. An example of the code for s snapshot test is displayed in an image below.



```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`render matches snapshot 1`] = `
<Text
  accessible={true}
  allowFontScaling={false}
  ellipsizeMode="tail"
  style={
    Array [
      Object {
        "color": "#5cbbe0",
        "fontSize": 45,
      },
      undefined,
      Object {
        "fontFamily": "Octicons",
        "fontStyle": "normal",
        "fontWeight": "normal",
      },
    ]
  }
>
  □
</Text>
`;
```

3.7 IMPLEMENTATION CHALLENGES AND ISSUES

The design of this application presents challenges because some design choices constrain us. Also, there are issues we need to face because decisions which address some problems have side effects. For example, we are required to use React Native which only a few group members are knowledgeable with. Furthermore, React Native is for mobile devices meaning members will need to either use an Android or iOS virtual device. Virtual devices can be difficult to set up and integrating it into the work-flow can be time-consuming. Our design also incorporates BT's API endpoints, but we are not given any documentation on them. To implement calls to these endpoints we will need to intercept web traffic coming from BT's mobile app and reverse engineer the calls ourselves. Lastly, creating shared components that provide some standardized functionality can be difficult. If the described functionality is too general it will not be useful and if it is too specific the component will not integrate correctly with others.

4 Closing Material

4.1 Conclusion

The team so far has worked together to decide on design concepts such as OKR and component-based architectures. We worked out what objectives we will be accomplishing such as the prototypes. We then need the key results which will be things like the components and other items like documentation that will signal the end of a phase and a prototype.

We will be working towards converting an industry-leading mobile app into React Native. In order to do this, we have broken our two semesters up into three phases and three main goals. The goals are to create an initial prototype, a structured prototype, and a final prototype. These three prototypes are each given about 2-3 months worth of time and will build off of the last prototype. Each phase is broken up into implementation and testing.

The initial prototype will need to be implemented with the goal of having a fully functional React Native project. This includes things like a functioning menu as well as options for users to be able to navigate through the app.

The structured prototype will build off of the initial prototype. This will take the navigation that the user has gained through the initial prototype and start to add functionality to the application. We will start to see more components being built as we work towards creating the project management software. The components will need to be improved upon, but we will see a baseline for almost all the components in this stage.

The final prototype will build off the structured prototype. At this point, we will have the user's navigation, and all the components completed to satisfaction. The final prototype will be packaged and ready to ship. This includes the application as well as the documentation The final prototype will have the functionality as the Buildertrend applications already have as well as the ability to easily add more.

Breaking up our project into three phases allows us to build a project that will start with a strong foundation. The foundation will be built upon to increase the application's functionality. Our project plan is like a pyramid. We will start with a big and strong base and build up from there until we have a completed project. The initial prototype is important to lay the groundwork for all of our future work. The nice thing about programming is that we will also be able to refactor our work and strengthen our foundation as we progress. With our plan, we will be able to see where weaknesses are and repair them before they cause substantial damage to other areas of the code. The three-phase plan is the best solution for our project and will result in a market leading product.

4.2 References

Academind.com. (2018). *React Native vs Flutter vs Ionic vs NativeScript vs PWA*. [online] Available at: https://academind.com/learn/flutter/react-native-vs-flutter-vs-ionic-vs-nativescript-vs-pwa/ [Accessed 29 Nov. 2018].

Airbnb.io. (2018). *API Reference · Enzyme*. [online] Available at: https://airbnb.io/enzyme/docs/api/ [Accessed 29 Nov. 2018].

Bowman, D. (2018). *Component Style*. [online] Medium. Available at: https://medium.com/@dbow1234/component-style-b2b8be6931d3 [Accessed 29 Nov. 2018].

Buildertrend.com. (2018). *Construction Project Management Software | Buildertrend*. [online] Available at: https://buildertrend.com/ [Accessed 29 Nov. 2018].

Cordova.apache.org. (2018). *Apache Cordova*. [online] Available at: https://cordova.apache.org/ [Accessed 29 Nov. 2018].

Diva-portal.org. (2018). [online] Available at: http://www.diva-portal.org/smash/get/diva2:998793/FULLTEXT02 [Accessed 29 Nov. 2018].

Expo. (2018). *Expo*. [online] Available at: https://expo.io/ [Accessed 12 Nov. 2018].

Facebook.github.io. (2018). *React Native · A Framework for Building Native Apps Using React*. [online] Available at: https://facebook.github.io/react-native/ [Accessed 29 Nov. 2018].

Flutter.io. (2018). *Flutter - Beautiful native apps in record time*. [online] Available at: https://flutter.io/ [Accessed 12 Nov. 2018].

House, C. (2018). *8 Key React Component Decisions – freeCodeCamp.org*. [online] freeCodeCamp.org. Available at: https://medium.freecodecamp.org/8-key-react-component-decisions-cc965db11594 [Accessed 29 Nov. 2018].

Ieee.org. (2018). *Ethics and Compliance*. [online] Available at: https://www.ieee.org/about/compliance.html [Accessed 12 Nov. 2018].

Jestjs.io. (2018). *Jest · Delightful JavaScript Testing*. [online] Available at: https://jestjs.io/ [Accessed 29 Nov. 2018].

NPM. (2018). *react-devtools*. [online] Available at: https://www.npmjs.com/package/react-devtools [Accessed 5 Nov. 2018].

Postman. (2018). *Postman*. [online] Available at: https://www.getpostman.com/ [Accessed 29 Nov. 2018].

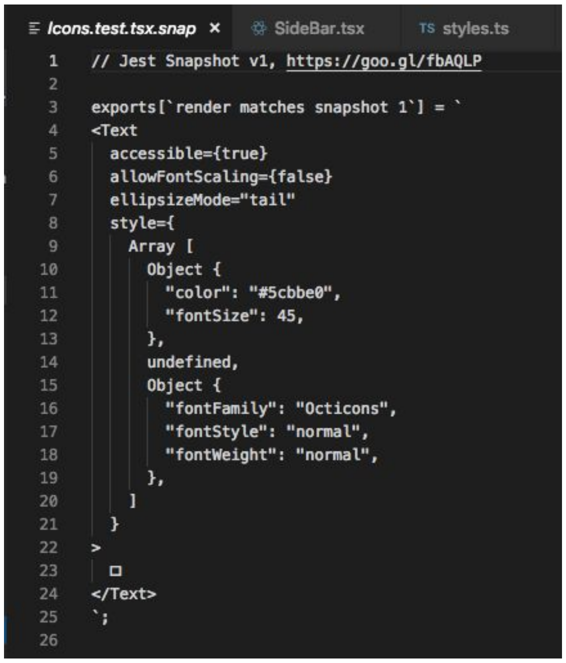
ReactRouterWebsite. (2018). *React Router: Declarative Routing for React*. [online] Available at: https://reacttraining.com/react-router/core/guides/philosophy [Accessed 5 Nov. 2018].

Redux.js.org. (2018). *Read Me - Redux*. [online] Available at: https://redux.js.org/ [Accessed 29 Nov. 2018].

Typescriptlang.org. (2018). *TypeScript - JavaScript that scales.*. [online] Available at: https://www.typescriptlang.org/ [Accessed 3 Nov. 2018].

Yarn. (2018). *Yarn*. [online] Available at: https://yarnpkg.com/en/ [Accessed 29 Nov. 2018].

4.3 Appendices



```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`render matches snapshot 1`] = `
<Text
  accessible={true}
  allowFontScaling={false}
  ellipsizeMode="tail"
  style={
    Array [
      Object {
        "color": "#5cbbe0",
        "fontSize": 45,
      },
      undefined,
      Object {
        "fontFamily": "Octicons",
        "fontStyle": "normal",
        "fontWeight": "normal",
      },
    ]
  }
>
  □
</Text>
`;
```

Figure 1: Snapshot Testing

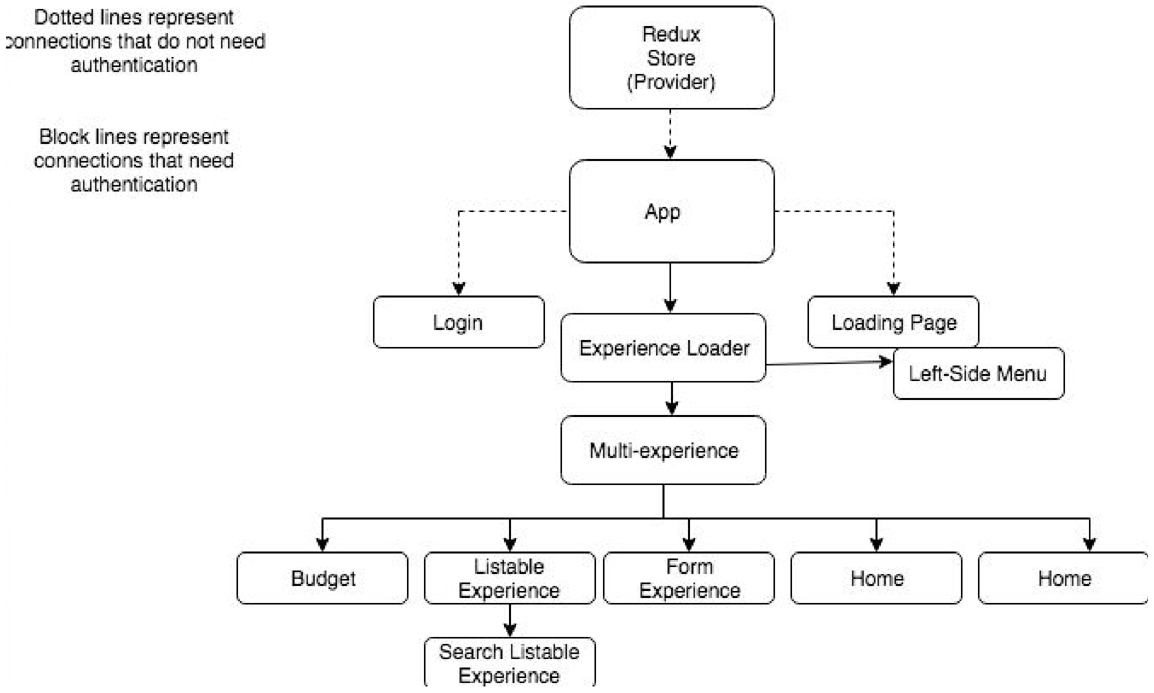

Dotted lines represent connections that do not need authentication
Block lines represent connections that need authentication
Redux Store (Provider)
App
Login
Experience Loader
Loading Page
Left-Side Menu
Multi-experience
Budget
Listable Experience
Form Experience
Home
Home
Search Listable Experience

Figure 2: Component Design 1

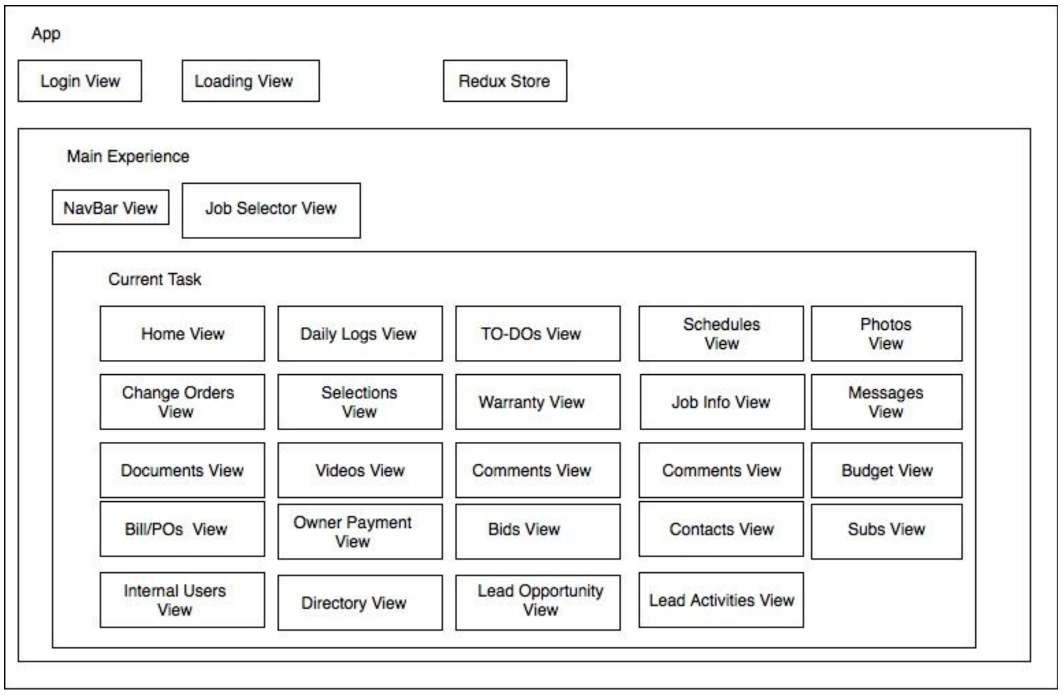


Figure 3: Component design 2