Convert an Industry-Leading Native Mobile App to React Native

Final Report

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: April 29th, 2019

Table of Contents

0 List of figures/tables/definitions

0.1 List of Figures

0.2 List of Definitions

- **API:** Application Programming Interface a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.

- **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 (ie duplicating logic)
- **Front-end:** The part of the development that deals with converting data into a graphical interface for users to interact with.
- **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 Navigation:** 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.
- **OKR:** Objectives and Key Results. A process of objective completion and validation.

1 Introduction

1.1 Executive Summary

The problem Buildertrend currently faces is that the mobile application they use could be created and maintained more efficiently. They currently are updating and maintaining two applications: one for iOS and one for Android. Having two platforms to support is costly and makes the software harder to maintain. Essentially, Buildertrend has to build the exact same application frontend twice. They are looking for a way to save money, time, and make the overall development process easier in the future.

Buildertrend had the idea to recreate the application with React Native, a framework that allows a single app to be built for both Android and iOS. With React Native, the JavaScript code is run by the phone's JavaScript engine instead of being run natively in most cases. Only one application needs to be built, meaning Buildertrend can drastically cut down on the required resources. 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.

Our project was to test the idea of the unified React Native app by implementing a subset of Buildertrend's frontend functionality. We would replicate Buildertrend's existing application using React Native along with other related technologies. The end result would be a mobile application consistent across iOS and Android with a single, maintainable code base.

1.2 End Product and Other Deliverables

Buildertrend's current mobile application is extremely large and complex meaning it would be impossible for our team to completely remake it. Therefore, Buildertrend decided on a subset of their application for us to implement. We needed to make the functionality for two widgets: Daily Logs, and Todos. However, we also needed to create the entire skeleton of the application with complete navigation and login to prototype the look and feel of the app. Other parts of the app were made with dummy components so Buildertrend can have a feel for how the app compares to their current solution. Additionally, we needed to integrate their backend by intercepting HTTP calls and recreating them within our solution.

Throughout the last two semesters, we have generated a design document, project plan, and many weekly reports. We have also created a project poster summarizing our work over the semesters. The main deliverables are the prototypes of the React Native application listed below:

- **Initial Prototype** - This is a skeleton of the Buildertrend application but contains key functionality which gave us a foundation to build off of. It has a functioning menu where the user can pull up the list of components. There are a few subcomponents outlined which show the completed navigation and functional design. The goal of this prototype was to set up the project and instantiate key architectural components.

- **Structured Prototype** - The structured prototype instantiates shared components that are used throughout the application. These shared components include job selecting menus, date selectors, search drop-downs, and image containers among others. Some API services are instantiated in this prototype, however, they are not linked to the UI yet. There is minimal styling because this prototype focuses on the functionality of the application. Some of the components which we were not focusing on were implemented with dummy components.

- **Final Prototype (End Product)** - This is the final product that we produced for Buildertrend, upgrading it from the structured prototype. It has all the features the original application has for Daily Logs and ToDos. The styling is completed and models Buildertrend's application very closely. The API endpoints for daily logs, todos, job updating, login, and logout are created and implemented in the UI for this prototype. Each prototype builds off each other meaning the functionality present in the other prototypes is present in this implementation.

2. Requirement Specification

2.1 Functional Requirements

Because we were recreating the Buildertrend application, there will be screenshots of their application to show the functional requirements.

1. **Daily Logs Widget** - Record of logs with messages to stakeholders which can include information such as dates, attachments, tags, and weather conditions. Instantiate the complete widget with the functionality provided in the BT app. There is a list of daily logs where one can be selected to view additional details.

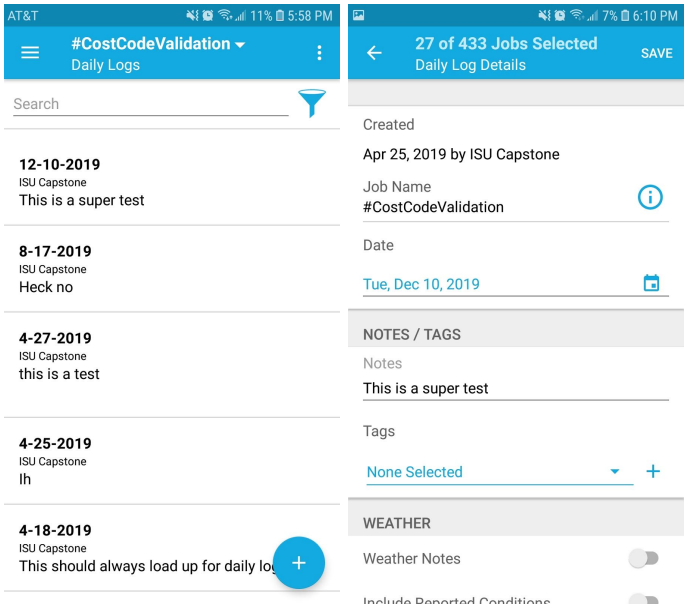

Figure 1: BT's Daily Logs

2. **To-Do's Widget**- Tasks the user can make for stakeholders. Can include text, attachments, assignees, and reminders. Again, instantiate the complete widget with the functionality provided in the BT app.

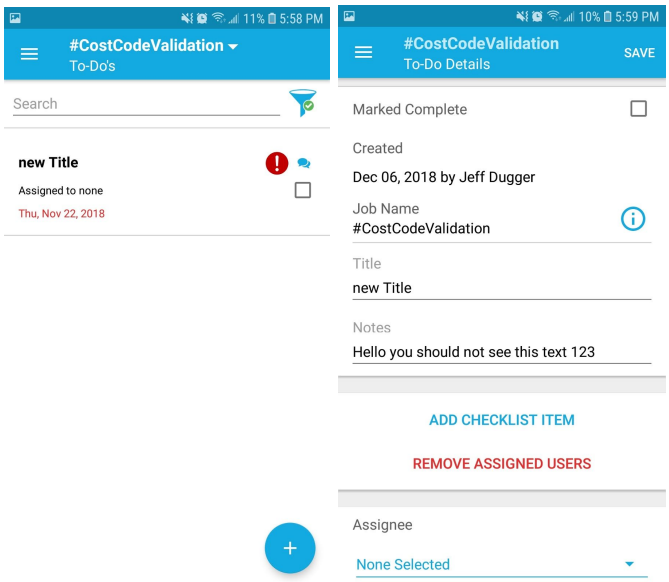

Figure 2: BT's ToDos

3. **App Navigation** - A screen with a sidebar that has icons for navigation between the different components of the application. The home screen also has icons for navigation to the different components.

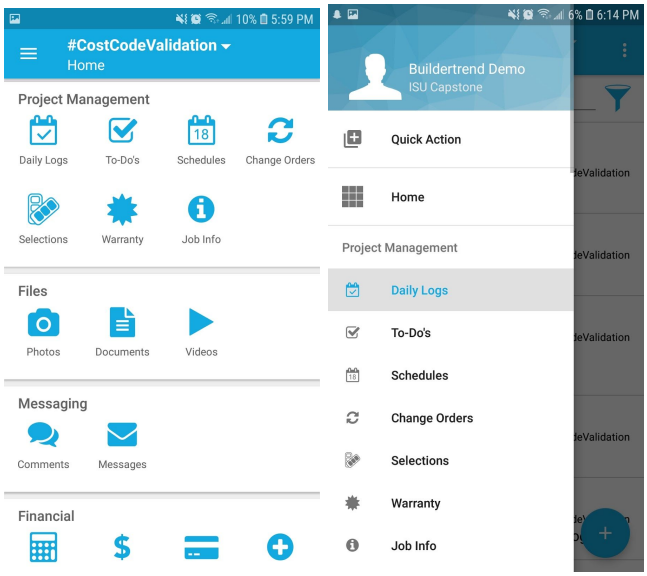

Figure 3: BT's App Navigation

4. **Job Selector** - A drop-down selector for updating the selected job. The job selector is available on many pages and must send an HTTP request to BT's backend to select the job (See figure 4).

5. **Login/ Logout** - A page for logging in and the functionality to log out which uses the BT backend for authentication.

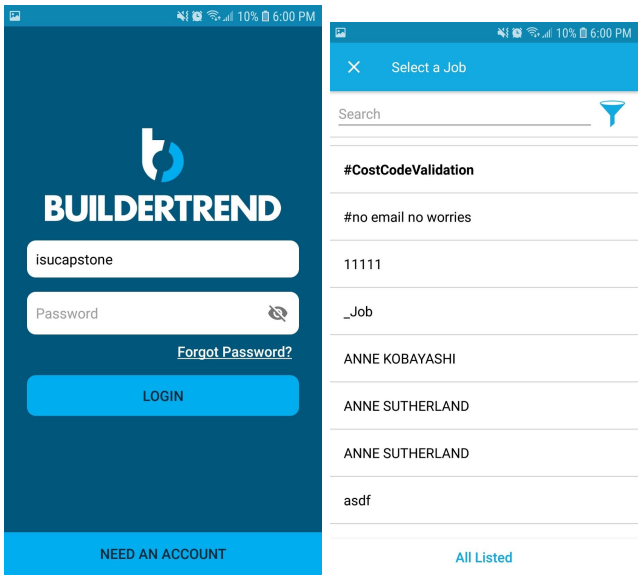

Figure 4: BT's Login/ Job Selector

6. **Dummy Components** - Components that represent some component in BT's app but are not built out. This allows BT to have the look and feel of the app without us having to create every component.

7. **Date Picker** - A reusable date picker that conforms to the date representation format used by BT's backend.

8. **Job Info Widget** - A page which shows the details of a job. A job can be selected and a user can view the information pertaining to a job.

2.2 Non-Functional Requirements

- **Consistency:** The application will have no noticeable UX differences between the Android and iOS application.
- **Responsive:** The mobile application should be user-friendly by providing quick results and reducing lag.
- **Performance:** The application is expected to have close to real-time results so it should be efficient with handling user requests.

- **Availability:** The application should be available at any point as long as the phone is active.
- **Maintainability:** The application is to be built in a modular way using a component-based architecture to make it easily maintainable
- **Data Integrity:** The application is to be built with data integrity in mind. This is done by displaying and sending accurate data and maintaining consistency.
- **Usability:** The application should be user-friendly and easy to use without a detailed explanation of how the application works.

2.3 Use cases

- User Sign in/ sign out
- Navigate through application
- Home Page
- View All Daily Logs
    - View individual Daily Log
    - Edit Daily Log
    - Create Daily Log
    - Search Daily Log
- View All TODO's
    - View individual TODO
    - Edit TODO
    - Create TODO
    - Search TODO
- Communication with Buildertrend's API
- View Job Info

3 System Design & Development

3.1 Objective of the Task

The desired outcome of this project is to have a React Native application which recreates Buildertrend's app and can be run on Android and iOS. The scope of the BT app we are implementing is defined by the functional requirements previously mentioned. It should have all these functional requirements along with non-functional requirements through the implementation of the last prototype. The user experience of the application should replicate BT's app as close as possible by replicating the functionality and styling. It should also interact with their backend services by recreating HTTP API calls used in their app.

3.2 System Constraints

- **Labor Resources:** Our team has a fixed sized with each member having their own strengths. In addition to classes, many of us are working jobs on the side. The balance between school, clubs, and work will be difficult throughout the semester.
- **Technical:** Our client has already chosen the technology they want the product to be developed with - which is React Native.
- **Time:** The product must be completed before the final presentation of the Senior Design class.
- **Integration with Pre-existing Technology:** We are only tasked with building the client end of the application. In order to access and save user data as well as authentication services, we must integrate with BT's pre-existing API endpoints. We do not have any documentation for their backend and we must capture HTTP traffic from their existing front end to discover the format of the requests.

3.3 Design Plan and Module Description

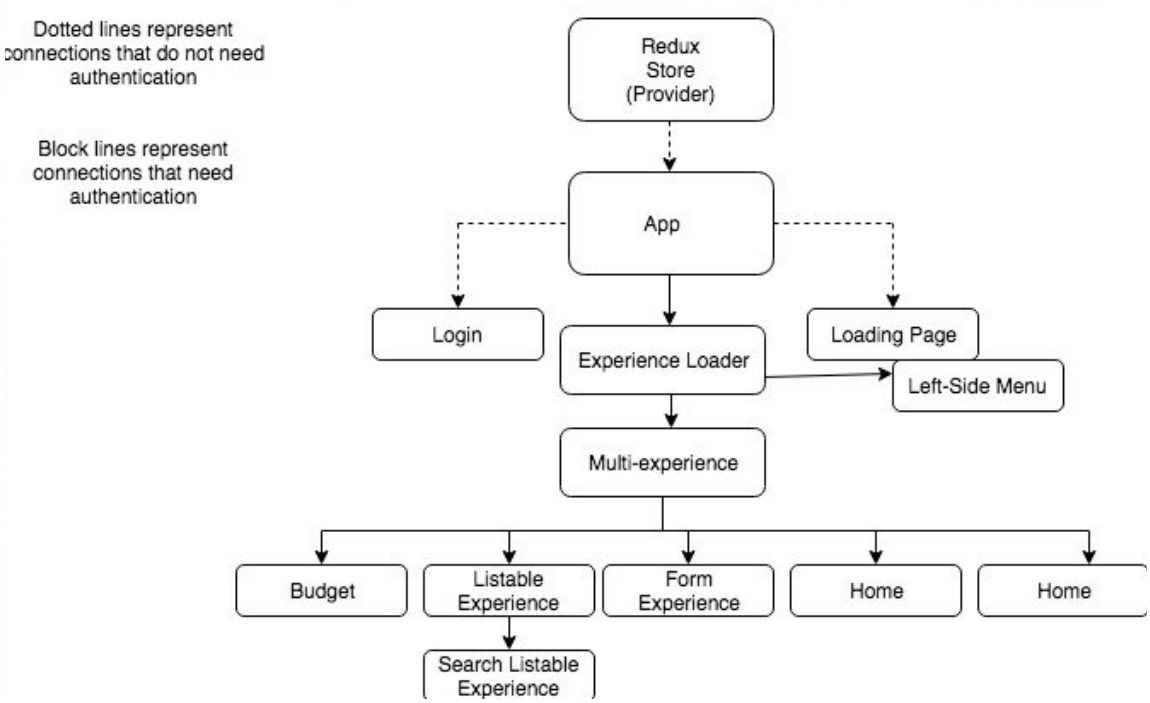

Figure 5: Conceptual Model

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 outermost container component, 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.

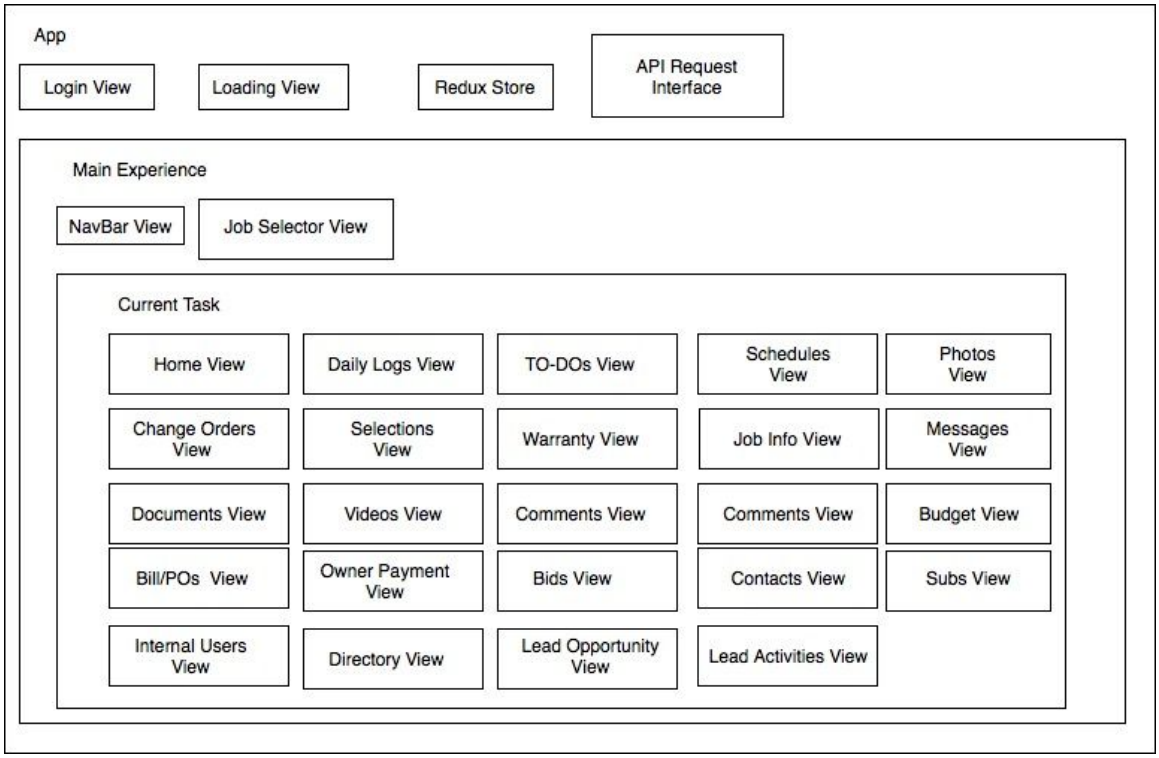

Figure 6: Design Block Diagram

Our strategy for the architecture includes using a component-based architecture and central store to save the application state. 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. In order 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 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.

3.4 Design Analysis

As a team, we came up with a process and an architecture that we used throughout the project. The process we came up with is an OKR(Objectives and Key Results) process. Essentially, we planned out our design into objectives. The objectives are easy statements that can be graded based on a binary scale of achieved or not achieved. We used this process to help us keep to our design and project plan so that we can make and observe achievements. 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 by checking if all Key Results under that objective were successful. That being said, the OKRs spanned for about three weeks to four weeks. We have also come up with an architecture for the application. The application is divided into the UI components and data-moving components. The UI Components, also known as Presentational Components, are the user interface components that are visual to the user. The data-moving components are mainly to control and move data to the appropriate UI component.

We decided to adopt the OKR process for our entire project as it worked well to help us keep track of what has been completed and needs to be completed. The process allows us to have guidelines for us to follow while completing objectives as well as a way for us to determine how well they were completed when we reviewed the progress of our project. This allowed us to analyze our strengths and weakness as we continued to improve our processes. The component-based architecture worked well for our architecture as were able to reduce repeated code and increase readability. We used high-order components to create reusable components that will maintain our component-based architecture. There were some changes that had to be made where we used an object-oriented approach with some components in order to improve code reusability.

One of the strengths that we observed from our process is the objectives. Keeping things objective helps in performance estimation as we are able to track the progress of the project and know when things are completed. This also helps with planning because we know exactly what has been accomplished at every point in time and can predict and estimate what would need to be done and when. Another strength is the ability to have multiple people working together almost seamlessly. With the component-based architecture, we were able to create a guideline or component API that classes of components have to adhere to, that way people could 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 work together since they were developed to meet a specific guideline as mentioned above, and there would be little work needed in connecting them. The downside to this is the overhead that had to be done before we could get the whole process going. We had to come up with frameworks, structures, and guidelines as mentioned above before the component development could be started. It seems like a fair trade-off since it lessened work later on as we had them all planned out.

4 Implementation

Our implementation can be seen in the following figures. These screenshots are not to be confused with Buildertrend's application because they are almost indistinguishable.

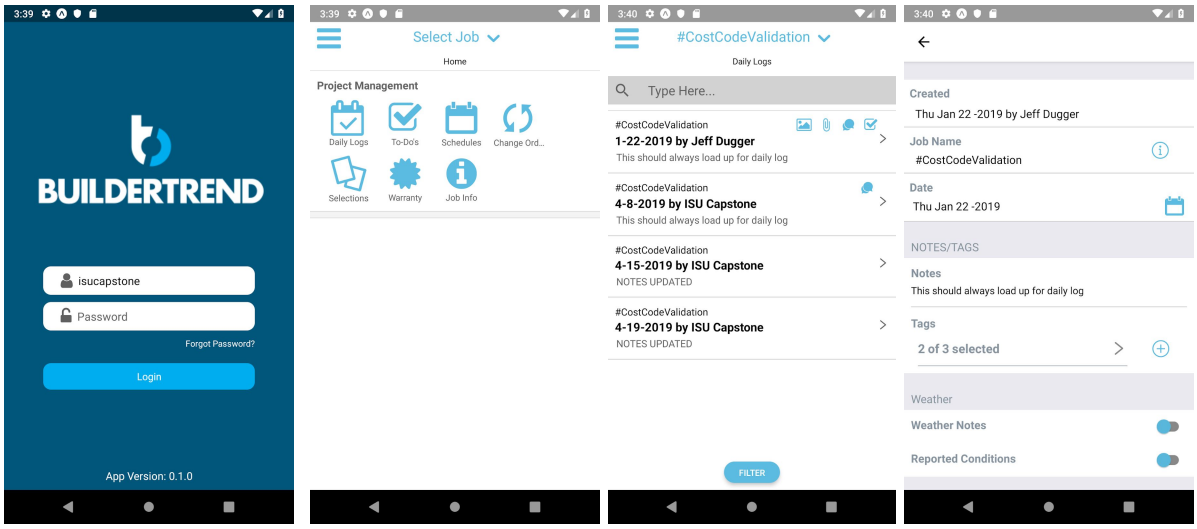

Figure 7: Implementation - Login/Home/Daily Log List/Daily Log

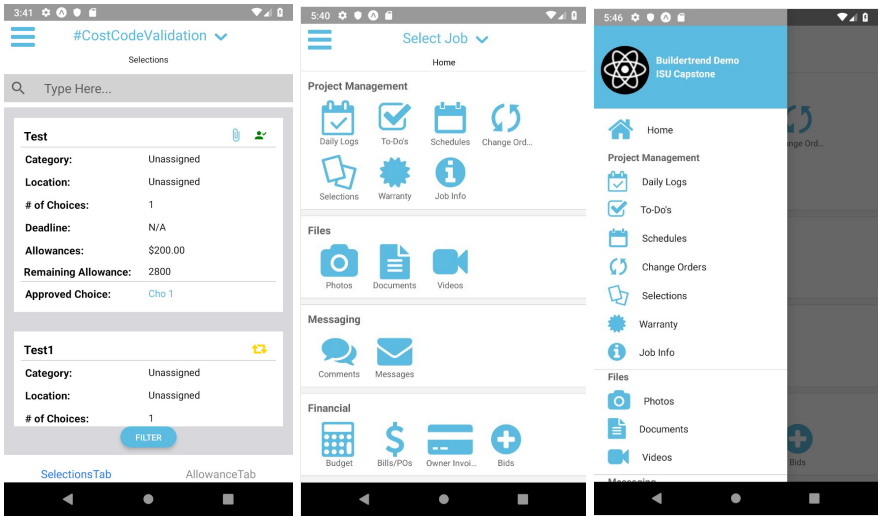

Figure 8: Implementation - ToDo List/Home and Sidebar with Dummy Components

4.1 Implementation Review

The figures below (Figure 9.1 - Figure 9.5) demonstrate the numerous development layers required for this application and give insight into how many layers we implemented. A layer is

defined as a specific depth within the app. A user starts at the top layer and goes down a layer with each subsequent page traversal.

Looking at the figures, each green box represents a layer. The blue boxes group the features on a page together. Since there are lots of features and user input on certain pages, instead of drawing individual lines to each, a blue box groups them all together. A white background means those features are completely implemented while the yellow layers represent that we have some functionality but not the exact same as what is in the original Buildertrend application.

It is important to mention that these figures only show ONE path. There are many numerous other paths with each path going its own path. The figures show the path from the Main Experience (one of five options in *App Layer),* to Job Selector View (one of three options in *Main Experience Layer)*, to Project Management (one of six options for *Job Selector Layer),* to Daily Logs (one of seven options in *Experience Layer)*, to Main Page, to the multiple sub-layers inside the Main Page.

From the multiple options of each layer, one can see that this application consists of many features and pages.

Our task was to develop all the features in a couple of *Experience Layers* and to develop at least several layers on the rest of the *Experience Layer* options.

To be able to implement the *Experience Layer*, we also developed the entire *APP Layer*, which includes the Main Experience, Login View, Loading View, Redux Store, and the API Request Interface. We also finished the Main Experience Layer and the three components that come with that.

We did not deeply develop the other five options in the *Job Selector Layer*. This was because our task was specifically to focus on the Project management's *Experience Layer,* and not the rest of Job Selector options.

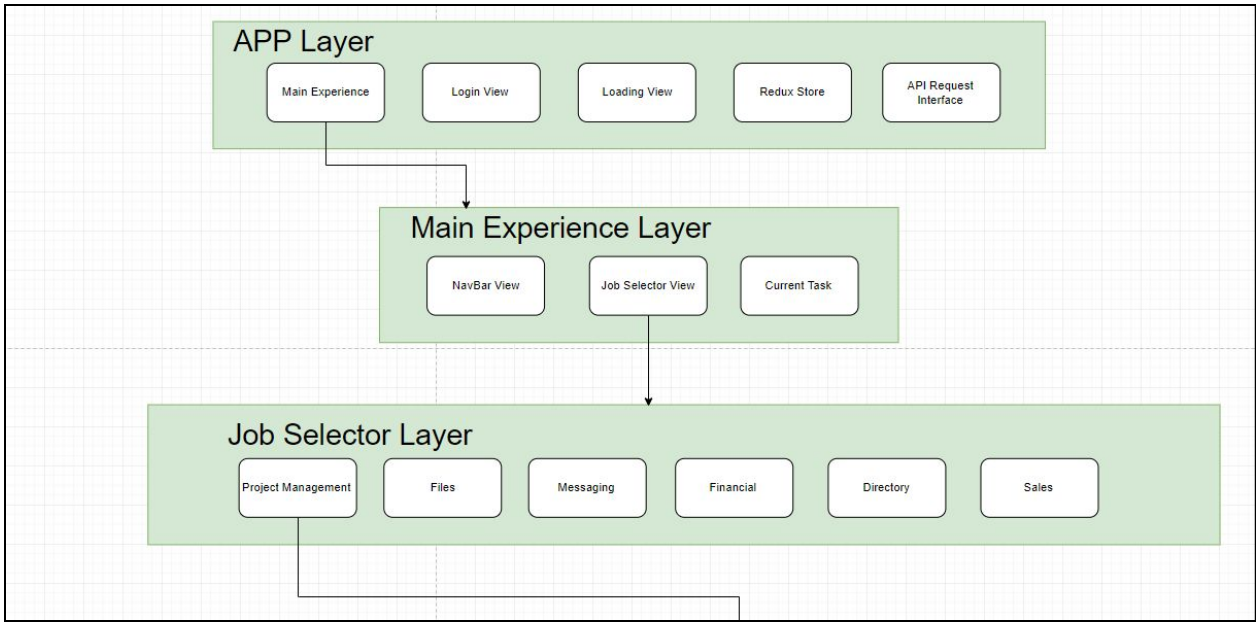

Figure 9.1: Implementation Diagram - Top Layer

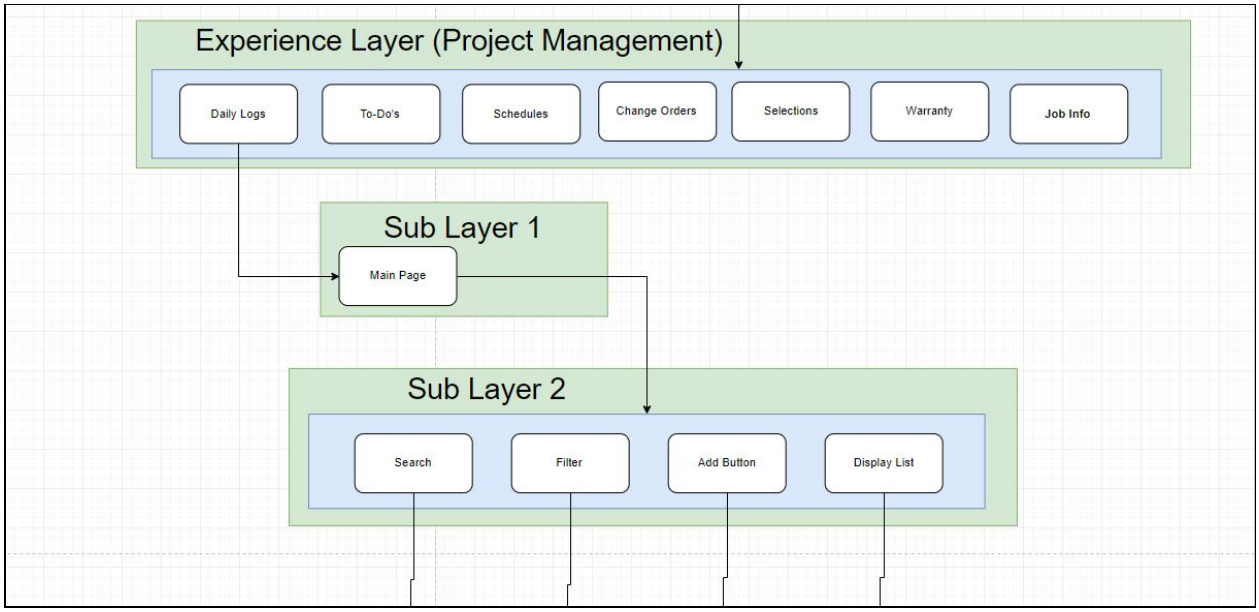

Figure 9.2: Implementation Diagram - Sub Layers

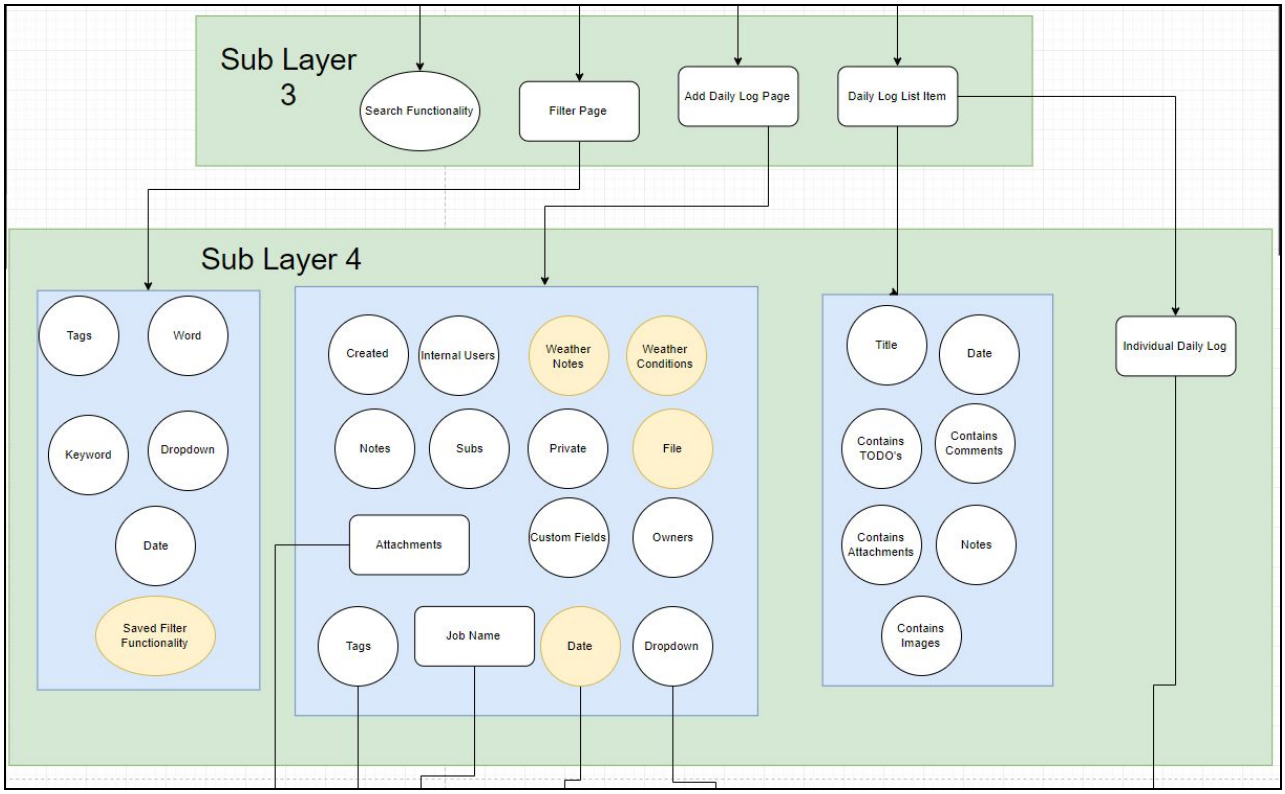

Figure: 9.3: Implementation Diagram - Daily Logs

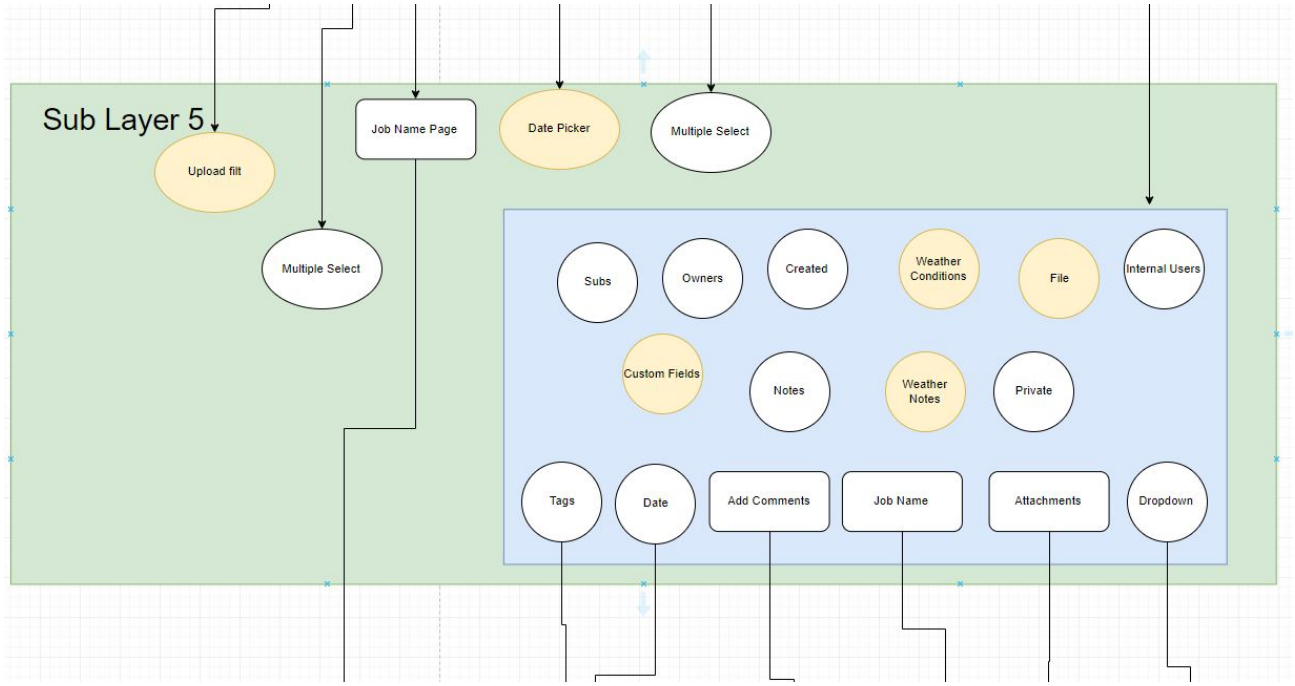

Figure: 9.4: Implementation Diagram - Daily Log Info Layer

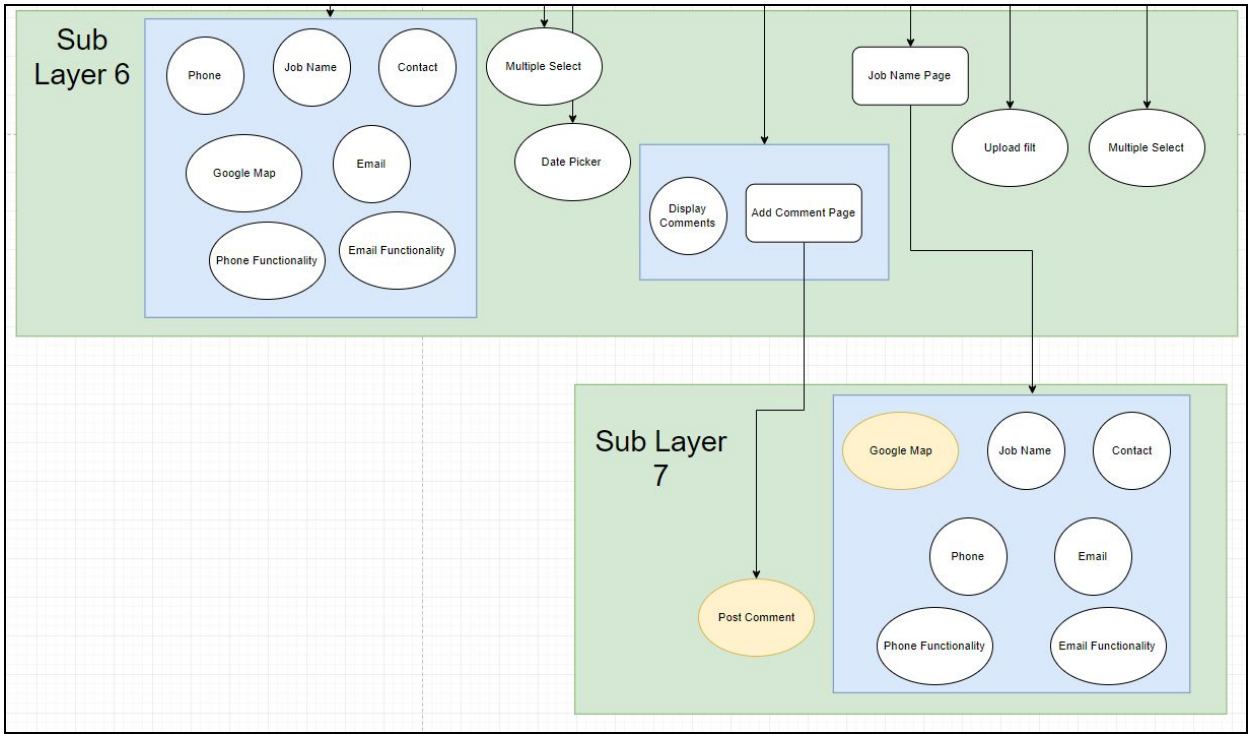

Figure 9.5: Implementation Diagram - Bottom Layer

4.2 Technologies used

- Charles Proxy - We are using Charles to get the HTTP requests used by the existing mobile application. Some of the alternatives were Postman and Fiddler, but we went with Charles because it was the most user friendly and easily allowed for HTTPS interception with its own SSL certificate.
- Expo - Expo is a tool for mobile development. React Native does not have a development tool like Expo available, which requires us to use both for efficient development.
- GitLab - We considered using another remote source control distribution, namely, GitHub because most of our team was familiar with it, but we decided to go with GitLab because it was already set up for us, it offered better issue tracking and easier code review and enforcement than GitHub.
- Linting - We are using TSLint to ensure code consistency in our project. Other alternatives include JSLint and ESLint. We picked TSLint because it is made specifically for TypeScript while the rest of them are geared towards JavaScript.
- NPM - We are using NPM and Yarn as our package managers. We use Yarn primarily for package managing because it has a lock file that will help us manage dependencies across

all our systems. NPM does offer a lock file now but we decided to go with Yarn because of its reputation for speed.

- React-Devtools Debugger - We use this for debugging React Components. Alternatives include the IDE debugger and the mobile's inspector. We chose React-Devtools because it was made specifically for React/React Native so it is geared towards what we are developing and has some specific features to React Native.

- React Native - The client requested that we use React Native. Alternatives to this are Flutter, Iconic, and Xamarin. The client already chose the technology we should use so it wasn't considered by us. One of the strengths is the popularity and community.

- React Navigation - We are using React Navigation for navigation within the application. One of the strengths of this is the efficient management of navigation which doesn't just help us do the work of navigation but also in optimal time. Some alternatives were React Router, and React Native Navigation. React Navigation seemed like a more mobile-based library compared to the other two.

- Redux - We are using Redux for state management within the mobile application. Other alternatives were Flux and MobX. We chose Redux because it has more community support and user-friendly. Flux offered almost no advantages. Some of our team members were also acquainted with Redux so we picked it over MobX. One of the weaknesses/trade-offs with Redux is the boilerplate that has to be done before the project can be started.

- TypeScript - We picked TypeScript as the language used for development instead of JavaScript. One reason we picked TypeScript is that it is a typed language which will help us avoid tricky bugs; it also has other feature languages that don't need a transpiler like JavaScript(besides the TypeScript transpiler itself). One weakness of TypeScript is that there might be some time spent in learning the language instead of actual development.

4.3 Application of Best Practices

Managing a software application of this size requires best practices to ensure the maintainability, scalability, and overall quality of code. The best practices we used include code reviews, code format standards, ticket tracking, and merge requests. We created a workflow that consists of these best practices and a system of voting on merge requests.

For any code to be included in the master branch, a merge request had to be created by the submitting developer. For each merge request, at least three team members had to approve after they did code reviews. Approval was only given after any comments on the merge request were resolved. Here one can see in figure 10 that this merge request was approved by three other people and there are 54 comments also. In figure 11 one can see the merge requests that were submitted by teammates.

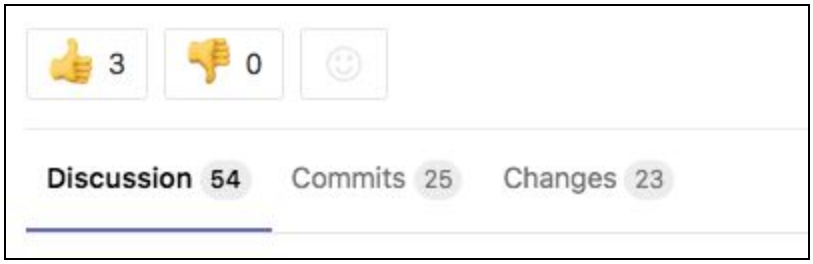

Figure 10: Merge Request Approval

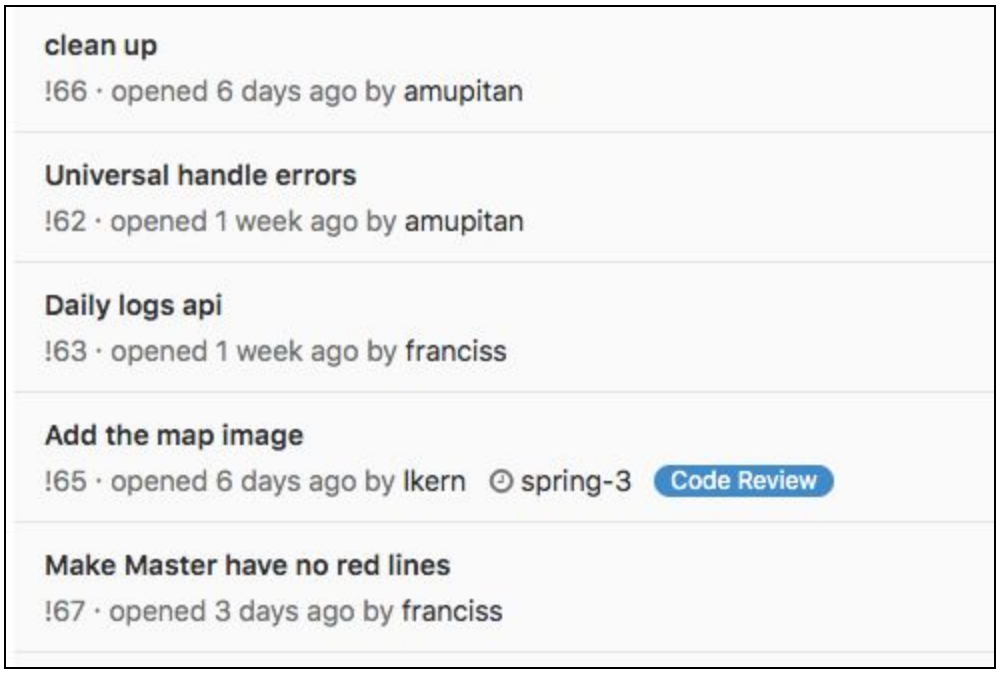

Figure 11: Merge Requests

Also, we tracked progress and divided tasks by creating tasks in GitLab. In figure 12 one can see how tasks are created and assigned to team members. Each task was also connected to a merge request and linked to a completion date goal (e.g. spring-3 indicates the last part of the spring semester).

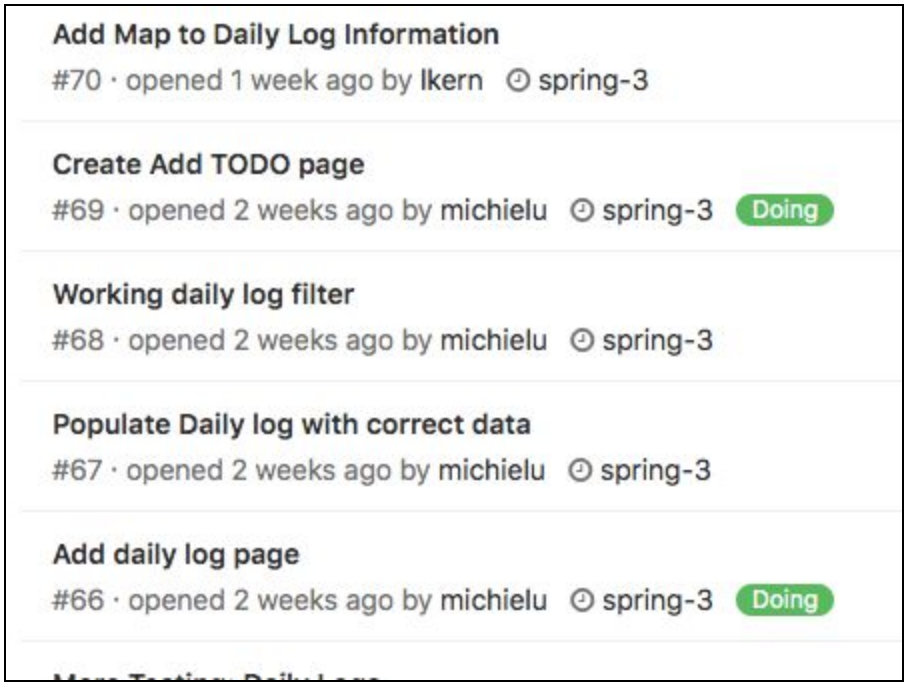

Figure 12: Git Repo Tasks

To maintain the quality of the code we adhered to a formatting style of the code. To do this we used an auto-formatting tool that can be installed as a plugin to VSCode, the text editor used for development. Also, we used a linter, a tool that automatically checks for style errors, bugs and performance issues. We ensured that all merge requests passed the linter checks before merging.

Our best practices were modeled after workflows and practices that we learned while working in the industry. We made sure that the quality of the project was the highest it could be even if the speed of development was hindered. If Buildertrend intends to use our project in the future, we had to make sure that our code base could be used as the foundation of an even larger and more complex project.

5 Testing, Validation, and Evaluation

5.1 Test Plan

We went with a continuous integration approach in the development process which included testing. Through this, we tested every addition to the code. This included automated and manual testing. Whenever a new feature was added, automated tests were added alongside and the then it was manually tested for correctness and regression.

Our automated tests mainly focused on unit testing for components. This was easy to accomplish because we used a component-based architecture - we wrote unit tests for each component that was created. This included components like a slider, search bar, page scroller, and other UI components. We had some integration testing for major components that interacted with others. This was for components like screen components and parts of the code that interacted with the API service.

Manual testing was also a part of the continuous integration process. Whenever a new feature was added or a change was made to the project, the application was manually tested by a team member that was not a direct part of that change. This testing involved testing for the correct behavior of the new feature or change, and testing to ensure there were no regressions in the application as a result of the change. Whenever a regression or incorrect behavior was found, the changes would have to be revised.

5.2 Hardware and Software Used

Jest - Jest is a JavaScript framework that makes it easy to create JavaScript tests. 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 either Node Package Manager or yarn 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 UI state as the last time the component was tested. The snapshots can be used to test UI state changes with interactions like clicking, scrolling, and other mobile events.

React Testing Renderer - React Testing Renderer (RTR) is a library also developed by the React team at FaceBook for testing react component behavior. Since React uses components, RTR can create actually test rendering a component. It has the ability to shallow render i.e. rendering the tested component without any child components and deep rendering which renders the child components as well. It is also used to test how the components respond to events (clicking, scrolling, swiping, etc). It also helps in testing React specific properties.

Since our project does not use any hardware besides a phone that runs our application, we did not focus on hardware testing.

5.3 Functional Testing

Unit Testing - We create individual test files for each component. We achieved 90% to 100% test coverage for most components. This includes statements and branches with component

functionality. Snapshot tests helped us make sure that each component worked correctly throughout the continuous development process.

Integration Testing - We tested larger components like screen components against each other. This focused on the component interactions.

Manual Testing - We manually tested each feature that was added, and also tested for regressions at the end of the development process. The manual testing focused on the functional and non-functional requirements of the application. Since Buildertrend gave us functional and nonfunctional requirements for the process, they will be able to easily test these requirements.

5.4 Non-Functional Testing

Testing the non-functional requirements was an important part of our project. Some non-functional requirements that we highlighted are performance, security, usability, and compatibility.

Performance - We used react devtools to profile the application. We also used Expo to test the performance of the application. We tested the application to be reasonably responsive to which it was.

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 was tested to be user-friendly and easy to use without a detailed explanation of how the application works. Since we did not have much control over how the application is formatted, our usability testing came 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. We also tested that long running requests had user friendly loaders to the user.

Compatibility - Compatibility comes down to how our application integrates well with BT's backend API services. We set up a test mock to simulate the calls we send to the backend service and compared it to the calls the old application were sending out.

6 Project and Risk Management

6.1 Task Decomposition and Roles and Responsibilities

Throughout the development process, we each took roles that fit our individual strengths. We split up tasks so we would all spend around the same amount of time working each week. Using Gitlab, we tracked issues to communicate the tasks we were working on. In our weekly meetings, we communicated what we worked on in the past week and what we planned to work on in the next week. To document these tasks, we wrote about our completed tasks in the weekly reports. Each individual completed tasks based on the following roles:

Victor: Project Designer

Frank: Backend Service Coordinator

Lucas: UI Developer

Walter: UI Developer

Kyle: Communication Lead/UI Developer

Michielu: UI Developer

All members of the team participated in code reviews. Every merge request was approved by three other team members before the code could be merged into the master branch. All individuals also wrote test cases for the components they were working on.

6.2 Schedule

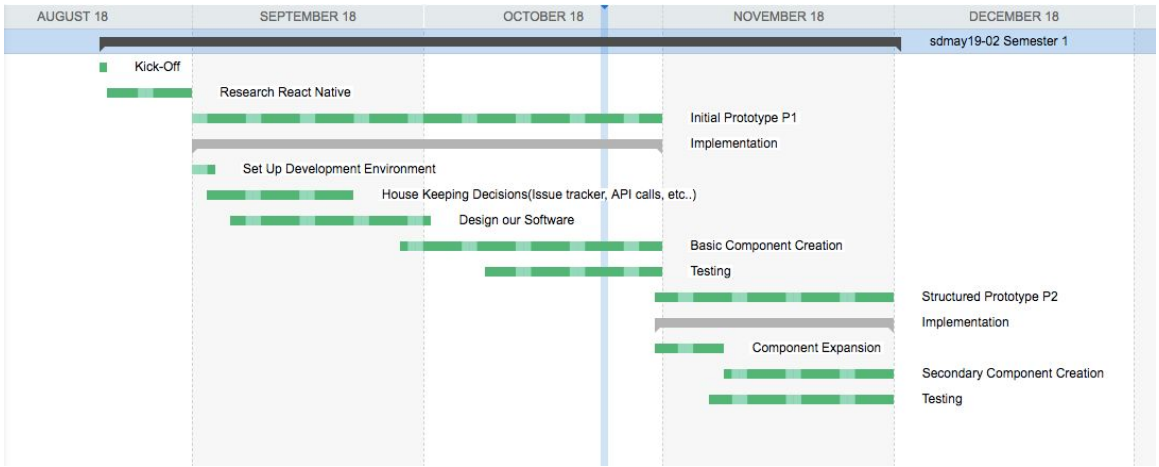

Figure 13: Semester 1 Gantt Chart

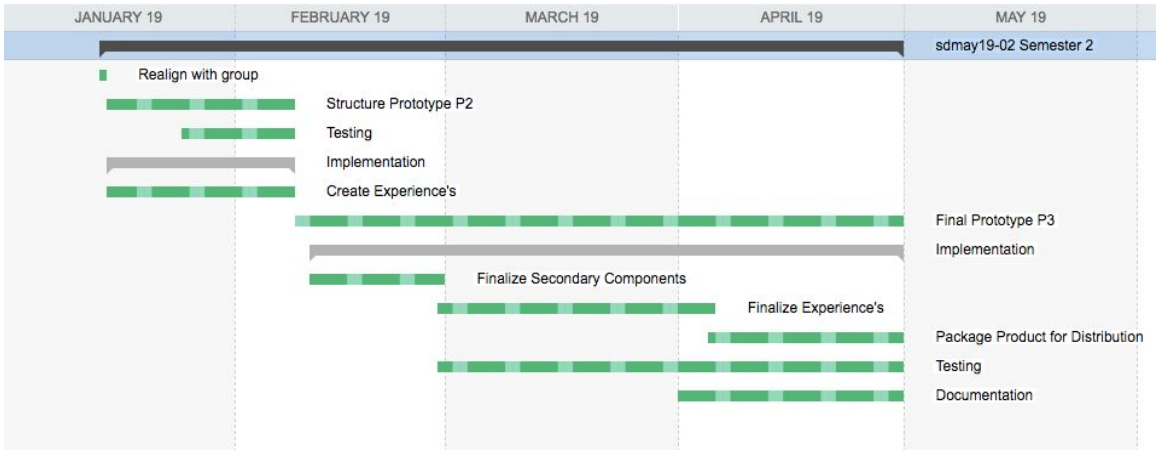

Figure 14: Semester 2 Gantt Chart

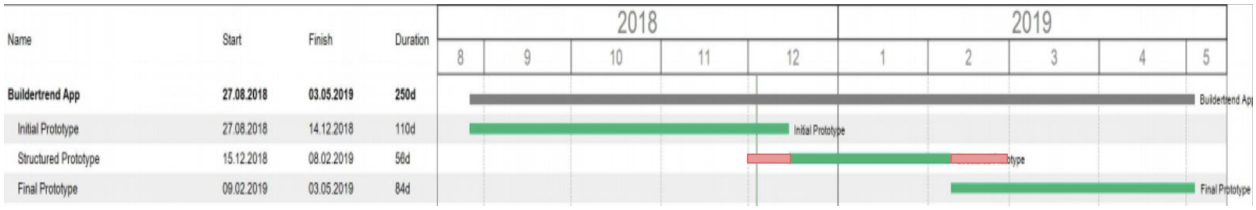

Figure 15: Simplified Final Gantt Chart

Our project timeline has been broken into three phases. These phases are based off our milestones which are an initial prototype, a structured prototype, and a final product. These phases take up a duration of the Fall 2018 semester as well as the Spring 2019 semester. The phases involve the implementation of that prototype, testing, and other steps that our team will take to complete the finished product.

The phases are broken up into two to three-month segments with the structured prototype being the shortest one. This structured prototype is built off of the initial prototype and the finished product is completing the structured prototype to be packaged and ready to present to the client. Please reference our Gantt charts represented in figure 13, 14, and 15 to view the timeline described below.

Phase 1: Initial Prototype (9/1-10/31)

- Building the skeleton code
- Basic app functionality
    - Basic components
    - Homepage
    - Login page
- Outline of components

- Having the app work on iOS and Android

Phase 2: Structured Prototype (10/31-11/30 & 1/14-2/8)

- Increased app functionality
- API Services for some components
- Testing
    - App functionality
    - Compatibility with BT services
    - Regression testing
- Implementation of shared components

Phase 3: Final Product (2/9-4/30)

- Finalize app functionality
- Deliverables completed
    - Project Report
    - Project Plan
    - Design Document
- Functional and Non-functional requirements completed
- Testing
    - Functionality
    - Regression testing
    - Requirements
- Packaged and ready to deliver
- Stylized Components

The simplified final Gantt chart shows how our actual development process turned out compared to our original plan. For the most part, we stayed on pace with the three major prototype phases. The only difference is that the structured prototype phase started earlier and ran a bit longer than we expected. Fortunately, this did not halt the process of working on our final prototype. We were able to have some people continue implementing BT functionality while others were working on completing functional and non-functional requirements.

6.4 Risks and Mitigation

Title: Lacking Understanding of React Native Framework

Strategy: Avoid

Premise: Coming into this project, most of the team members had not used the React Native framework. Lacking understanding of React may cause slips in schedule and loss in quality as members take time to grasp React. Estimating the required amount of time for tasks will be difficult because we have no understanding of our efficiency with React.

Action Plan: Getting as much experience with the framework will give us the best results if we are able to make more informed guesses and utilize aspects of React that improve quality. We plan on doing tutorials and reading documentation to understand React better.

Actual: By the end of our project, everyone was comfortable with the React Native framework. Every single member of the team wrote at least some code using React Native. React Native was used in nearly every merge request, and there is plenty of documentation online to help people who are not comfortable with the framework. With time, this risk was completely avoided.

Title: Relying on Phones

Strategy: Mitigate

Premise: React Native compiles native code for Android and iOS phones. In order to run the application, team members need to use their phones. Relying on phones in the workflow introduces risks such as not having a working phone, not having a charged phone and compatibility between the phone and the application. It is possible to create virtual devices to run the application but this may prove to be difficult.

Action Plan: We will work in groups so that phones can be shared if need be. There will probably be at least one person with a working phone. Also, we will look into the option of having virtual devices and the feasibility of this option.

Actual: Some members decided to download IOS or Android simulators on their computer so they did not need to depend on their own phones. Other members were able to develop comfortably on their own phones. There were times where issues persisted on Android but not IOS and vice versa. In these times we were able to use another member's phone for a short period of time to solve the problem.

Title: Malformed HTTP Requests

Strategy: Mitigate

Premise: The application we are developing uses the Buildertrend backend functionality through their API. However, we do not have any documentation on the API and therefore must intercept requests on their app to find out how it is used. Not having access to the API documentation will cause a lack of understanding and therefore may cause incorrect or malformed HTTP requests.

Action Plan: Having one person dig deep into the HTTP calls and having a map of what each does will provide other team members with the knowledge to use them correctly. When Buildertrend's API doesn't return what we expected, we can rely on the person who researched it more closely. Also, we can use our contacts at Buildertrend to ask questions about what might be going wrong.

Actual: Although this was a challenge, we were able to intercept the API calls made on Buildertrend's original application and use that information to hook our application into the backend. Using Charles, we could see what requests looked like and how the data needed to be formatted. This was also important so we could see responses to these API calls.

Title: Lack of knowledge of HTTPS interception for mobile applications

Strategy: Avoid

Premise: Android applications prevents the interception of HTTPS traffic by rejecting SSL certificates provided by Charles. This makes it impossible to intercept SSL traffic on an Android phone or emulator without having access to the code base. Also, it is impossible for us to run Buildertrend's app on an iOS simulator because we would need the code base. The only way to intercept traffic is with an iPhone.

Action Plan: Have a team member lend an old iPhone to the person who is intercepting HTTP calls. That way the person intercepting traffic will always have an iPhone available to use.

Actual: This risk was an issue for us because we thought it was possible to intercept traffic on an Android phone. We put off getting the web endpoints because we did not need them at that moment. We devised a plan to have Buildertrend give us some endpoints to get going but they only gave us the login endpoint. After some research and some trial and error, we finally realized we needed an iPhone. We spent a lot of time dealing with this issue which pushed back the timeline.

6.5 Lessons Learned

A very valuable lesson that was learned was how having a consistent codebase helps with the overall quality of the code, which mitigates many misunderstandings and problems in the future. Even though it could be very tedious and uncomfortable to criticize constructively other member's code, the overall quality and clearness of our code allow it to be scalable. As our code stands, we will easily be able to give what we have to Buildertrend and they should not have problems continuing development.

We also learned that having documentation for a codebase is very important. We may have been able to accomplish more or finish our tasks more efficiently If we had documentation or examples of the existing API Buildertrend had. Instead, we had to go through a process of tracking the API through a different application and figure out what we needed that way. Although creating documentation seems daunting at times, it saves time and effort for future development.

Lastly, we learned that preparation and planning are quite important for long-term projects. Having a schedule from the beginning allowed us to have a much better understanding of what tasks needed to be done by what time. It also allowed us to make sure everyone had something to

work on at all times. Because we took the time to create schedules and milestones, we all were aware of what each individual needed to do for the overall success of the project.

7 Conclusion

Creating a React Native mobile application that mirrors Buildertrend's has come with many challenges and successes. Our idea of the end product was constantly evolving as we learned more about the technologies we were using. Through interactions with our clients and the deliverables of the class, we have developed our ability to work in professional settings and produce high quality work. We have decided that our project is a success because we were able to closely replicate Buildertrend's application and show its viability as a React Native app.

Since our prototype for creating one single code base for both iOS and Android has been a success, the rest is up to how Buildertrend would like to go with it. They could build off of what we have, since we have a considerable amount of investment and stable infrastructure, or they could start developing their own React Native Buildertrend Application. Regardless of how Buildertrend would like to take this, our project showed them how it is possible to navigate through the mobile platforms and provide a comprehensible application for all.

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].

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].

Team Information

Victor Amupitan: amupitan@iastate.edu

Lucas Kern: lkern@iastate.edu

Francis San Filippo: franciss@iastate.edu

Michielu Menning: michielu@iastate.edu

Kyle Nordstrom: kjnords@iastate.edu

Walter Seymour: wseymour@iastate.edu