# ProConnect Maintainer Guide

[GitHub Repository Link](https://github.com/hadiDanial/ProConnect)

### About

ProConnect is a full stack application with multiple different layers, utilizing different technologies:

1. A front end application/website created using React Native+Expo, which can be deployed to a hosted website or a mobile application (Android APK)
2. A back end server created using Spring Boot and Java, hosted on Heroku, along with a PostgreSQL database
3. A Firebase project to handle Authentication, Cloud Storage, Firestore Database and Hosting

### Front End

The front end is developed using React Native, using Expo SDK 49 and TypeScript, React Native Web, React Native UI Library (by Wix) for theming and styling, React Native Navigation, React Hook Form for form validation, Redux Toolkit for state management, and RTK Query for API calls.

##### Running the project

In order to run and update the front end project:

1. Clone the repository.
2. Navigate to the ProConnect\_Frontend folder in your code editor of choice.
3. In the terminal, run the command npm install to install all dependencies.
4. Update the .env file to have the correct URL for the server (REACT\_APP\_BASE\_URL), which can point to a local server or the Heroku server and the Firebase configuration (apiKey, authDomain, projectId, storageBucket, messagingSenderId, appId, measurementId).
5. In the terminal, run the command npx expo start to run the project on a local development server using Expo. You can then press a or w after the project loads to run on an Android emulator or on the web (or alternatively scan the QR code on your Android phone to run on your device).

##### Project Structure

The project starts in the ProConnect\_Frontend/App.tsx component, which wraps the entire application. The theme gets initialized here, and a Redux Provider wraps the rest of the app so that we can access the global state from everywhere in the app. Inside the App component is the ProConnect\_Frontend/src/Index.tsx, which contains the Navigation Container, which consists of a Navigation Stack for the authentication pages, and a Drawer Navigator which contains the rest of the application flow, including an inner tab navigator as well.

The global style definitions, including theme colors and default component stylings, can be found in the ProConnect\_Frontend/Styles.tsx file.

If the user is not authenticated, they start off in the authentication navigation stack, where they can login, sign up, or reset their password. Otherwise, they get automatically navigated to the main navigation stack.

###### Folders in the ProConnect\_Frontend/src directory:

* All the base components that get reused throughout the project can be found in the Components folder, where they are divided into two categories: Controls and Layout. Controls are the components that perform certain actions or act as inputs, while the Layout components are those that are only responsible for displaying content in a certain way.
* In the Features folder are the components responsible for certain features of the app, including certain behaviors such as displaying jobs or users etc...
* The Hooks folder, there are some hooks which implement generic functionality that can be useful in various different places, but don't require a component, such as the useImagePicker hook responsible for picking images from a gallery or camera.
* The Navigation folder contains all the components that handle navigation.
* Screens contains all the screens that are included in the navigation.
* Services has Firebase and Redux configuration files and services, as well as the API.

##### Updates

###### API

The server's REST API is documented using Swagger and OpenAPI, and that documentation is used to generate the API calls in the front end application using RTK Query's codegen-openapi functionality. To update the API calls, you can update the schema file parameter in the ProConnect\_Frontend/openapi-config.ts file to match the server's address:

schemaFile:'http://SERVER ADDRESS/v3/api-docs'

And then run the following command in the terminal (assuming the server is running):

npx @rtk-query/codegen-openapi openapi-config.ts

Afterwards, you'll have the updated API, along with updated entities/models, and type definitions in the ProConnect\_Frontend/src/Services/Redux/Api.ts file.

##### Deploying

To deploy the application:

1. **Web version:** The web version is hosted on Firebase hosting. Ensure that you have a .firebaserc file in the ProConnect\_Frontend/ folder, and then run the following commands to build and deploy the React Native Web application on Firebase hosting:
   1. npx expo export --platform web
   2. firebase deploy --only hosting
2. **Android version:** To build an Android APK, you need to ensure that you are logged into the EAS platform using the EAS CLI, and then run the following command to generate an APK file you can download to an emulator or a physical device:
   * eas build --platform android --profile preview

### Back End

The back end was developed using the Spring Boot framework in Java, using Hibernate, Spring Security, Firebase Admin SDK, Lombok, and a MySQL (local) or PostgresSQL (on Heroku) database.

##### Running the project

In order to run and update the back end project:

1. Clone the repository
2. Navigate to the ProConnect\_Backend folder in your code editor of choice
3. Wait for Maven to download all the dependencies from the pom.xml file
4. Run the main function in src/main/java/com.braude.ProConnect/ProConnectApplication.java

##### Architecture

The project follows the standard Spring Boot application architecture:

1. Models/Entities that define the data and the database tables, defined in the com.braude.ProConnect.models.entities package.
2. Repositories (interfaces) that handle database queries for any given Entity, which work behind the scenes with any database, in the com.braude.ProConnect.repositories package.
3. Services that handle the business logic of the application, using the repositories as needed to use or update the data in the database, found in the com.braude.ProConnect.services package.
4. Controllers that accept HTTP requests and communicate with the services to perform actions and return the responses, in the com.braude.ProConnect.controllers package.
5. Security configuration, including providers, authentications and filters for the Spring Boot Security chain, found in the com.braude.ProConnect.security package.
6. Configuration files in the com.braude.ProConnect.config package.

##### Swagger/OpenAPI

The server has Swagger documentation you can access by going to http://server-address.com/swagger-ui.html, where you can see the entire REST API, including what each endpoint accepts and returns. In addition, you can try each endpoint, which performs real actions on the server. This is secured using Spring Security, and you can only activate the functions if you are authenticated (username: admin, password: proconnect).

The OpenAPI configuration is defined in com.braude.ProConnect.config.OpenAPIConfig class, while the API documentation itself is built automatically from the REST controllers.

The OpenAPI config generated by Swagger is used by the front end to generate the API calls using RTK Query's codegen-openapi.

##### Deploying

The server is hosted on Heroku, using a Heroku PostgreSQL database

To deploy the server, there are two ways:

1. Log in to the Heroku dashboard, select the application, and then deploy from branch.
2. Use the following commands in the terminal to create a .jar file and deploy to Heroku using the Heroku CLI:
   1. mvn install -e -DskipTests=true
   2. heroku deploy:jar target/ProConnect-0.0.1-SNAPSHOT.jar --app proconnect