HomePairs Startup Guide

Table of Contents

Introduction:

This document will briefly overview the structure of the current programmatic progress of the HomePairs application. This introduction assumes that the users have an in depth understanding of design patterns and Software Engineering principles. We also assume that the user has some experience with TypeScript outside of the react environment. The purpose of this document is to explain the structure of the app as well as the technologies used.

Prerequisites:

1. Previous Knowledge:
   a. TypeScript experience:
      i. https://www.tutorialspoint.com/typescript/index.htm
   b. Other Helpful Resources:
      i. 7 tips to develop React-Native Apps for all Screens
      ii. Simple Navigation Between Pages
      iii. Complex Navigation
      iv. Building an Application with Authentication Flow
      v. Passing Data Between Parent, Child, and Sibling Components
      vi. Present Content over another View (Popup/Modal)

        vii. Useful VS Code Extensions for React-Native Projects
        viii. Loading Custom Fonts (and other Assets)
        ix. React-Native Components Documentation

2. Installations Needed
    a. Libraries Used:
        i. React-native-elements
        ii. React-navigation
        iii. Expo-font
    b. All users will also need:
        i. NodeJS
            1. Most recent version available at: https://nodejs.org/en/
            2. Can be installed with
                a. ‘Brew install node’
        ii. Yarn
            1. Can be installed with
                a. ‘Brew install yarn’
        iii. Expo-CLI
            1. Can be installed with
                a. Preferred Method (if yarn is already installed)
                    i. ‘Yarn global add expo-cli’
                b. Other Method
                    i. ‘Npm I -g expo-cli’

Notes

Only react-native-elements need to be manually installed. However, if you find yourself needing to install a library this is the format:*yarn add [NAME_OF_LIBRARY]*

Creating the project:

Now to create our working directory, run:
expo init

You will be asked a few questions including what to name the project. Make sure to create the project as a typescript project. Now navigate into the newly created directory. You should have a templated project.

Running the project:

To test run the app you can run it through a simulator manually or use the expo support to get faster updates. I'm currently using expo so this we will go that route today by running:

yarn start

You can choose to run on IOS or Android device.
However, expo also allows us to run on the web. The command is simple:

yarn run web

These commands can also be found package.json file in "scripts"

Defining Types for React Components:

The fundamental advantage of using react and react-native is the ability to divide up our project in smaller chunks known as components. Since we are using Typescript, we also have some additional information that we must use to help define our components as opposed to JavaScript. For example, let's examine the LoginScreen.tsx file.

```
import { StyleSheet, Text, View, Linking, ScrollView } from 'react-native';
import {Card} from 'react-native-elements';
import React from 'react'; //**For every file that uses jsx, YOU MUST IMPORT REACT  */
import InputForm from '../../UIComponents/InputForm';
import ThinButton from '../../UIComponents/Buttons/ThinButton';
import AuthStyles from './AuthStyles'
import {NavigationProps} from '../../../utility/NavigationProps'
import { SafeAreaView } from 'react-navigation';

/**
 * StyleSheet for all components in this directory and children directories
 * */
const parentStyles = AuthStyles.AuthScreenStyles

/**
 * These define the types for the prop and state attributes for the component. Notice
 * how the LoginScreenProps extends from NavigationProps. This is so all screens that are
 * in a NavigationStack (check App.tsx) are defined along with ant properties specific
 * to this component.
*/
interface LoginScreenProps extends NavigationProps {}
interface LoginScreenState {
            username? : String,
            password? : String,
}
```

```
export default class LoginCard extends React.Component<LoginScreenProps, LoginScreenState>{
        constructor(props){
        super(props)
        this.state = {
        username : '',
        password : '',
        }
        }

        /**
        * Optional navigation options. Here is where will will define animations, headers, and other
        * settings for navigationg from this screen.
        * */
        static navigationOptions = {}

        /**
        * Callback functions for child components. These functions are intended to passed into a childs components
        * parentCallBack attribute.
        * EX:
        * <InputForm name='EMAIL' parentCallBack={this.getFormUsername} />
        */
        getFormUsername = (childData : String) => {
        this.setState({username : childData})
        }
        getFormPassword = (childData : String) => {
        this.setState({password : childData})
        }

        /**
        * Event functions for when something occurs on this component.
        */
        _clickSignIn = () => {
        alert(this.state.username + " " + this.state.password)
        //TODO: Make authentication request to BackEnd
        }
        _clickSignUp = () => {
        this.props.navigation.navigate('SignUp')
        };

        /**
        * NOTE: If you want your ScrollView to actually scroll, you must set the style of the ScrollView {{flex:1}}. Do
not
        * have the contentContainerStyle have a flex. This will effectively make your ScrollView unable to scroll!
        */
        render() {
        return(
        <SafeAreaView style={parentStyles.pallet}>

        <ScrollView style={{flex: 1}} contentContainerStyle={styles.assetLoadedContainer}>
        <Card
        containerStyle={parentStyles.authCardContainer}
        title="HomePairs"
        titleStyle={parentStyles.homePairsTitle}>
        <View style={parentStyles.container}>
                <Text style={parentStyles.subTitleText}>Sign in to your Account</Text>
                <InputForm name='EMAIL' parentCallBack={this.getFormUsername} />
                <InputForm
                name='PASSWORD'
                parentCallBack ={this.getFormPassword}
                secureTextEntry={true}/>
                <View style={parentStyles.submitSection}>
                <ThinButton
                name='Sign In'
             onClick={this._clickSignIn}
                buttonStyle={parentStyles.thinButton}
                buttonTextStyle={parentStyles.thinButtonText}/>
                </View>
                <View style={parentStyles.signUpSection}>
                <Text style={parentStyles.standardText}>New to HomePairs?{" "}
                <Text
                     style={{color: '#0098CD'}}
                onPress={this._clickSignUp}>
                  Sign Up
                </Text>
                </Text>
                </View>
        </View>
        </Card>
        </ScrollView>
        </SafeAreaView>
        );
        }
}

/**
 * Individual styleSheet specific for this component. Most components in this directory
 * will use the AuthStyles stylesheet but this one had a specific case for it and it only.
 * Therefore a styleSheet was defined for specific this.
 */
const styles= StyleSheet.create({
        assetLoadedContainer: {//This container will only set default items to center of the pallet. It will always have
a blue pallet behind it as well.
        maxWidth: 380,
        backgroundColor: '#1177B0',
        alignItems: 'center',
        justifyContent: 'center',
        alignSelf: 'center',
        width: '100%',
        aspectRatio: 1/1.5,
        minHeight: 750,
        },
});
```

React Components have two types of values that must be defined: **props** and **state**. Since we are using TypeScript we must simply define an interface for what values these will hold. **Recall that props is immutable and the only way to change anything in state must be within the* ***setState()*** *function.*

The Root of the Project:

A majority of React-Native projects will have a default file known as App.tsx. This is the file that the React Compiler looks for when it is compiling the code into html/native code. I believe it can also be named index.tsx but I would not focus on this. For now, we will keep the name as App.tsx.

A. Project Structure
    a. Screens
        i. These are components that are the pages of an application. They represent the large-scale combination of smaller components.
    b. UIComponents
        i. These are smaller parts of a page that will be used by the Screens. For now, even Screen specific components are stored here. It is possible that these components may be divided into a more flexible class later down the road.
    c. Utility
        i. These are files that are simply helpful. **NavigationProps** allows us to have all screen props inherit the values that a NavigationScreen will require. This prevents us from needing to define these props in every single screen. Also, I think putting our enums here will also be very helpful in the future.
    d. Navigation
        i. The expo projects for will include a default library known as the react-navigation. This library eases the problem of navigating between different screens and pages of our app. It's a very good practice to divide our application into different pages. These pages will be divided up into smaller components. *If you ever need to add a library, go into the project's directory and input the following command: *yarn add [NAME_OF_LIBRARY]*

App.tsx

```
import {createAppContainer, createSwitchNavigator, } from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack';
import HomeScreen from './Components/Screens/Main/HomeScreen';
import LoginScreen from './Components/Screens/Auth/LoginScreen';
import LoadingScreen from './Components/Screens/LoadingScreen';
import SignUpScreen from './Components/Screens/Auth/SignUpScreen';


const authStackConfig = {
 defaultNavigationOptions: {
        headerTintColor: '#fff',
        headerStyle: {
        backgroundColor: '#000',
        },
        headerShown: false,
 }
}

const MainStack = createStackNavigator({ Home: HomeScreen});
const AuthStack = createStackNavigator({ Login: LoginScreen, SignUp: SignUpScreen},  authStackConfig);

export default createAppContainer(createSwitchNavigator(
 {
        Main: MainStack,
        Auth: AuthStack,
        Loading: LoadingScreen,
 },
 {
        initialRouteName: 'Loading',
 }
));
```

Let's focus our attention to what is actually happening. We have this type called an **AppContainer**. This is where all the screens of our application will be referenced for later usage. This app container has what is called a **SwitchNavigator**. This is a navigation class that allows the application to switch from component/Screen/Navigator to the next. This is often used for applications that have authentication.
Notice how we also have two attributes: **MainStack** and **AuthStack**. Both of these are **StackNavigators**. **StackNavigators** essentially have the same functionality as the **SwitchNavigator** but with bonus functionality. We will go into bonus functionality when we head into our first Screen.

*You can literally add a screen to **NavigationStack** with the push function. I believe it preserves the content of the previous page (like in Android). I'm not exactly sure on this one.

Notice the attribute **initialRouteName**. This is the page the **AppContainer** will route to by default. Most applications with authentication will route the page to a loading screen.

Now is a good time to show the purpose and functionality of the LoadingScreen. Below is the source code:

LoadingScreen.tsx

```
import React from "react";
import { AsyncStorage, View, ActivityIndicator, StatusBar, StyleSheet } from "react-native";
import * as Font from 'expo-font';
import {NavigationProps} from '../../utility/NavigationProps'


export default class AuthLoadingScreen extends React.Component<NavigationProps> {
        constructor(props) {
        super(props);
        this.state = {
        assetsLoaded: false,
        };
        this.loadAssets();
        }

        async loadAssets(){
        await Font.loadAsync({ //Every Assest not found in native devices, load them here!!
        'nunito-regular' : require('../../assets/fonts/Nunito-Regular.ttf')
        });
        this.setState({assetsLoaded: true});
        this._bootstrapAsync()
        }

        // Fetch the token from storage then navigate to our appropriate place
        _bootstrapAsync = async () => {
        //TODO: Implement UserTOKEN in STORAGE ITEM const userToken = await
AsyncStorage.getItem('userToken');

        // This will switch to the App screen or Auth screen and this loading
        // screen will be unmounted and thrown away.
        this.props.navigation.navigate('Auth')//userToken ? 'App' : 'Auth');
        };

        // Render any loading content that you like here
        render() {
        return (
        <View style={styles.container}>
        <ActivityIndicator />
        <StatusBar barStyle="default" />
        </View>
        );
        }
}

const styles = StyleSheet.create({
  container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
  },
});
```

NavigationProps.tsx:

```
interface NavigationProps {
        navigation: any
}

export {NavigationProps};
```

There are some things we must go over. First, LoadingScreen is a React Native component. It inherits all the functionality from the React.Component class. In this case, one of the properties will be known as a navigation. This is a property passed to our LoadingScreen from the NavigationStack or NavigationSwitch.

The main purpose of this LoadingScreen is to load the proper assets for the application and to check for a valid session. When all assets have been loaded, this screen will check for the status of the current session. If a session is expired or has not begun, then this screen will route the user to the Auth NavigationStack. Otherwise the user will be navigated to the main application.