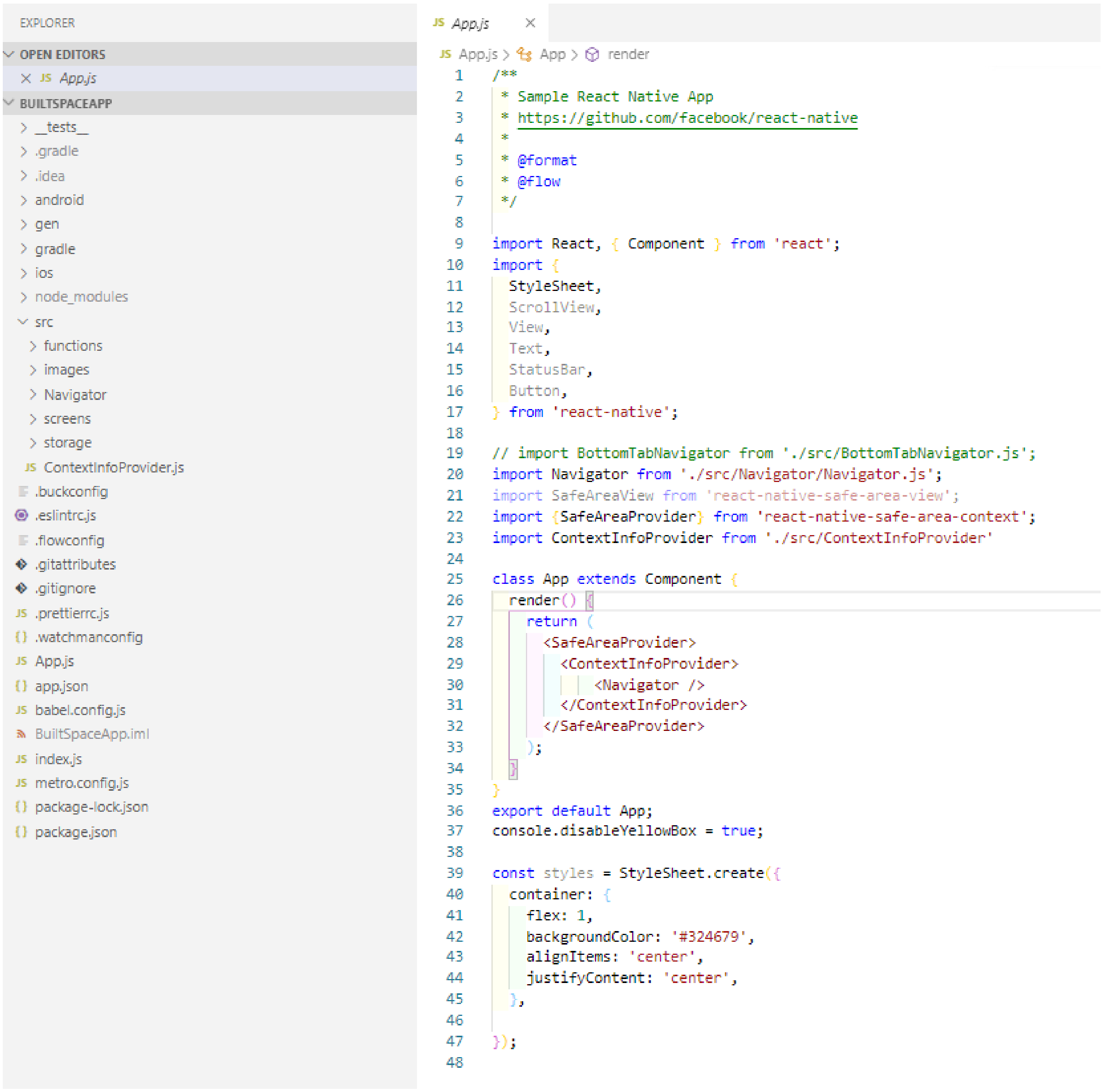
BUILTSPACE

May 19, 2020

Table of contents

Project File Structure

Once you have the git repository cloned and installed the node modules using npm install the project folder will look something like this in visual studio code.



```
EXPLORER
OPEN EDITORS
  × App.js
BUILTSPACEAPP
  > __tests__
  > .gradle
  > .idea
  > android
  > gen
  > gradle
  > ios
  > node_modules
  ∨ src
    > functions
    > images
    > Navigator
    > screens
    > storage
    ContextInfoProvider.js
  .buckconfig
  .eslintrc.js
  .flowconfig
  .gitattributes
  .gitignore
  .prettierrc.js
  .watchmanconfig
  App.js
  app.json
  babel.config.js
  BuiltSpaceApp.iml
  index.js
  metro.config.js
  package-lock.json
  package.json
```

```
App.js > App > render
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 * @flow
 */

import React, { Component } from 'react';
import {
  StyleSheet,
  ScrollView,
  View,
  Text,
  StatusBar,
  Button,
} from 'react-native';

// import BottomTabNavigator from './src/BottomTabNavigator.js';
import Navigator from './src/Navigator/Navigator.js';
import SafeAreaView from 'react-native-safe-area-view';
import {SafeAreaProvider} from 'react-native-safe-area-context';
import ContextInfoProvider from './src/ContextInfoProvider'

class App extends Component {
  render() {
    return (
      <SafeAreaProvider>
        <ContextInfoProvider>
          <Navigator />
        </ContextInfoProvider>
      </SafeAreaProvider>
    );
  }
}
export default App;
console.disableYellowBox = true;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#324679',
    alignItems: 'center',
    justifyContent: 'center',
  },

});

```

The App.js file is the root of the project. App.js contains the wrappers for the Navigation and all the screens for the BuiltSpaceApp.

What's in the /src/ directory

/src/functions contains a functions.js file that holds importable functions that are used in the BuiltSpaceApp.
/src/images contains images for the camera icons.
/src/Navigator/navigator.js contains the react-native navigation code.

May 19, 2020

/src/screens contains all the screens and components in the BuiltSpaceApp.

/src/storage contains the script to fetch the api data and the database functions.

/src/ContextInfoProvider.js
This file is the component for creating a context API for the application. The context api allows any screen wrapped in the context component to access a global variable. I used context to make a global variable for account details and the network status.

/src/storage/fetchAPI.js
This script contains all the http get requests from the BuiltSpace server.

/src/storage/schema/dbSchema.js
This file contains all the database schemas and database functions for the BuiltSpaceApp.

How the app works

When the app first loads, it renders the App.js root file. The App.js file wraps all the /src/Navigator/Navigator.js under a SafeAreaView, the context API. So once the App.js loads it renders the Navigator.js, and the Navigator.js contains all the screens in the BuiltSpaceApp.

The Navigator.js will first load /src/screens/AuthLoadingScreen.js for an authentication listener (which is now implemented yet) and navigates to the login.js by default. We did not create any authentication and hard coded the account details into the context api within the LoginForm.js.

Loading data in certain screens
The app is structured so it will always be fetching data from the database as its priority. But if it is the user's first time using the app this is how the flow works.

/src/screens/LoginScreen/Login.js
There is not authentication implementation in the app so the login button goes straight into the HomeScreen and assigns the a context variable called accountContext with the hard coded values in loginForm.js's state variable.

/src/screens/HomeScreen/*HomeScreen.js*
Once the user logs in for the first time, the app checks to see if a database exists and creates one if it does not. Then the app will check if the account details of the user logged in exists and creates the account. When the account is being created it will also do a http get request to fetch the list of organizations and its buildings that belong to the user and adds it into the database along with the account.

In the home screen the user can delete inspections that have been saved in the device (submit is not implemented yet). The user can log out, and the user can continue in the app by clicking the select an organization button. When we goto the SelectOrganizationScreen we will also pass in the organization list as a prop to reuse.

/src/screens/operations/SelectOrganizationScreen.js
This screen will render a list of organizations using the react-native component Flatlist. The user can select an organization which will goto the SelectBuildingScreen for that organization. A prop orgName will be passed to the SelectBuildingScreen. We will be using orgName in the api to fetch the building data.

/src/screens/operations/*SelectBuildingScreen.js*
When the SelectBuildingScreen loads it will be similar to the HomeScreen. There will be checkers to see if there is building data for this organization and checks if a 1 hour interval has passed. If 1 hour has passed, the data will automatically fetch the newest data from the BuiltSpace API and updates realm db. In this screen the user can select a from a Flatlist of buildings to do inspections on. When the user selects a building the app will navigate to the BuildingDetailsScreen, passing in the organization data and building information as navigation parameters.

/src/screens/operations/BuildingDetailsSreen.js
This screen will have three buttons: Start Visit, Browse and Scan QR Code. Start visit is not implemented yet. The browse button will let the users see all the assets, spaces and checklists in the building. When we click the browse button we will navigate to the ExploreBuildingScreen where the bulk of the operation exists.

/src/screens/operations/ExploreBuildingScreen.js
This screen will load 6 components from the */src/screens/operations/**components**/* directory. The initial screen contains three buttons that will trigger a modal on click. The buttons are **spaces, assets, checklist**. Each of the modals will render a Flatlist of selectable items that will update the state variable. When a space is selected, it will filter all assets in that space. When an asset is selected, it will filter the checklists that belong to the asset. When a checklist is selected the app will render all the questions in that checklist.

The questions are rendered by a Flatlist as well. When loading the flatlist will check the type of question whether it is ‘labour’, ‘materials’ or a ‘general’ question. Each question can be answered by filing the text input and selecting the formatted buttons. The user can save the current questions into the account as savedInspections.

The code

/src/screens/LoginScreen/Login.js
The login.js file has some basic components for styling and a <LoginForm/> component that passes in the navigation as props.

/src/screens/LoginScreen/LoginForm.js
This screen holds the form for logging into the application. The state variable holds the account details. The account details should be set when authentication succeeds. We did not implement this feature. This is where the accountContext is assigned. It is assigned when the user clicks login. When the user clicks log in it will trigger two functions:

1) Runs a function in the ContextInfoProvider.js to set the account details. We pass in the state.account object. This should be changed to an authentication logic.
2) Uses the navigation to switch to the Homestack.

/src/screens/HomeScreen/*HomeScreen.js*
The componentDidMount function will create a listener to refresh the page whenever we go back to this screen. The listener will run/rerun the loadData function on didFocus.

The loadData function will call checkDBExists in the dbSchema.js
Then the checkAccountExists and getAccountOrgs in the dbSchema.js
If the account does not exist it will fetchOrgs from the fetchAPI.js and passes in the organizatoins into the dbSchema's insertNewAccount function.

loadInspections will call the getInspections function from dbSchema.js and sets the checkboxes for each inspection saved in the account.

submitInspections is not implemented.

deleteInspection runs the delInspections function in dbSchema.js

/src/screens/operations/*SelectBuildingScreen.js*
Uses the navigation parameters to render a list of organization names.

/src/screens/operations/BuildingDetailsSreen.js
Uses the navigation params to display the buildings details such as city, address, postalcode.

/src/screens/operations/ExploreBuildingScreen.js

spacesFilter will run in the spacesModal. Whenever a space is selected, it will run this function to filter a list of assets that match the space's floor.

assetsFilter will run in the assetsModal. Whenever a space is selected, it will run this function to filter a list of checklists that match the assets' assetCategory.

May 19, 2020

loadQuestions will run in the checklistModal. Whenever a checklist is selected, it will run this function to assign a state variable called setQuestions. This variable holds all the questions that belong to the selected checklist.

updateQuestion will run in the three question type components (GeneralType, LabourType, MaterialsType). Each textinput and button in a question is assigned to this function. This function takes in 3 mandatory arguments and two optional arguments, measurement_label and measurement_unit. updateQuestions will check the type of input that it wants to process and updates the index of the question in the state setQuestions.

resetState is run when the refresh button is pressed. It set all the selected values to the default values.

onChange will run when a space, asset or checklist is selected. This function will change the state of the selected type to render in the modal.

loadData will load existing data form realm db or updates the database if an hour has passed.

updateBuildingData is called in loadData and if the refresh button is pressed.

saveAlert is called if the button Save to device is pressed. If ok is pressed this function will run saveToDevice().

QRCODE - openLink, onQRCodeScanDone

saveToDevice runes if save ok is pressed in the saveAlert. This function will save all the questions, general comments into the account.savedInspections schema.

CAMERA FUNCTION - cameraOnPress, onBottomButtonPressed

addQuestion will run if one of the buttons in the flatlist footer is pressed (add labour, add materials, add issue). The function checks the type and pushes a new question into setQuestions.

/src/functions/functions.js

getStartTime will create the current date and formats it into a readable time string and returns it.

calculateDurationInspection will take two arguments, starttime, and endtime. The function converts the two into a time object and calculates how long an inspection took to finish and returns the calculated value.

formatInspectionObject is run in ExploreBuildingScreen.js. When the saveToDevice function is called it will also run this function to format a checklistObject to be saved into realm database.

formatAddQuestion is called when the addQuestion function is called in ExploreBuildingScreen.js. This function checks the type of question and creates the object that will be pushed into setQuestions and returns the new list of question.

Realm v3.6.3 JavaScript Docs Here

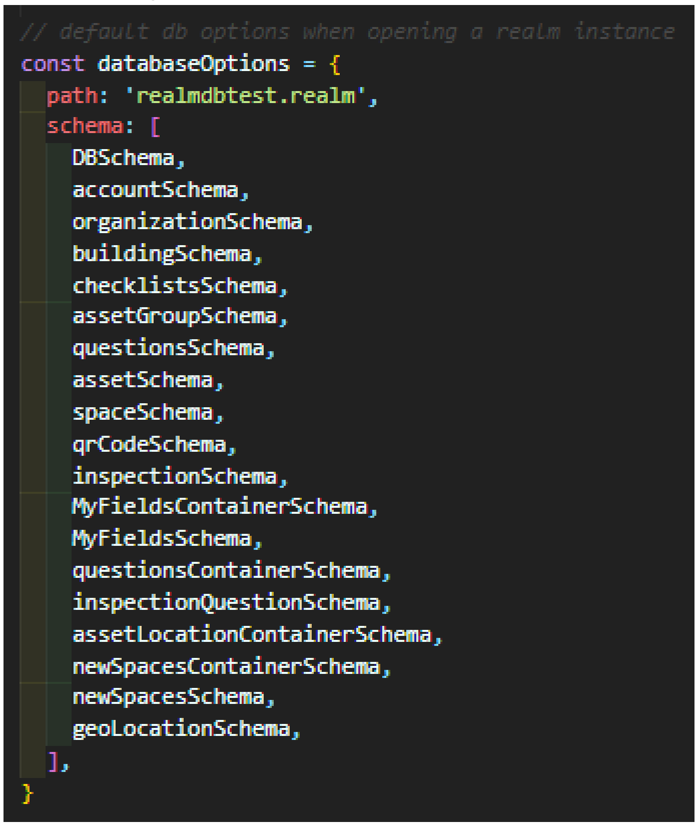
Realm v3.6.3 does not support recursive deletion of nested objects therefore we have to delete from bottom up.

Realm functions used in the BuiltSpaceApp:

Realm.exists() ● Take in a configuration option as an argument. ● Returns true or false ● I used this to check if the db exists before inserting new account ● I passed in the databaseOptions variable in dbSchema.js. ○ Consists of all the schemas	Realm.open() ● Takes in a configuration option as an argument. ● I passed in the databaseOption variable in dbSchema.js. ● Returns a callback with a realm object. ● We can then query the realm object.
realm.object(<schema>) ● Once the realm is open we can use .object on the callback to fetch all data within a schema.	realm.object(<schema>).filtered() ● Filters a schema with a string argument. example `realm.object(‘Accounts’).filtered(‘id == 200’)` .isEmpty() ● Checks if filtered content is empty. .isValid() ● Checks if filtered content is not deleted (exists)

May 19, 2020

Realm Schemas:
Schemas represent an object and its attributes. In builtspace the schemas are placed in the dbSchema.js file.



```
// default db options when opening a realm instance
const databaseOptions = {
  path: 'realmdbtest.realm',
  schema: [
    DBSchema,
    accountSchema,
    organizationSchema,
    buildingSchema,
    checklistsSchema,
    assetGroupSchema,
    questionsSchema,
    assetSchema,
    spaceSchema,
    qrCodeSchema,
    inspectionSchema,
    MyFieldsContainerSchema,
    MyFieldsSchema,
    questionsContainerSchema,
    inspectionQuestionSchema,
    assetLocationContainerSchema,
    newSpacesContainerSchema,
    newSpacesSchema,
    geoLocationSchema,
  ],
}
```

May 19, 2020