1. **HOME SCREEN**
   * **handleRegisterPress** and **handleLoginPress** are event handler functions called when the corresponding buttons are pressed. They navigate to the 'Register' and 'LogIn' screens, respectively.
   * **handleAnonymousLogin** is an event handler function that performs anonymous login using Firebase Authentication's **signInAnonymously** function. Upon successful login, it navigates to the 'MainPage' screen.
   * **handleGoogleSignIn** is an event handler function that performs Google sign-in using Firebase Authentication's **signInWithPopup** function. Upon successful sign-in, it navigates to the 'MainPage' screen.
   * **handleLanguageChange** is an event handler function that takes a language object as a parameter and sets the **language** state variable accordingly.
2. useEffect Hook:
   * The **useEffect** hook is used to subscribe to the **onAuthStateChanged** event provided by Firebase Authentication.
   * It listens for changes in the user's authentication state. If the user is already logged in anonymously, it navigates directly to the 'MainPage' screen.
   * The effect runs only once when the component is mounted, thanks to an empty dependency array **[]**.

Overall

It sets up the home screen with buttons for various actions, handles user authentication using Firebase, and allows language selection.

**LOGIN SCREEN**

1. State and Constants:
   * The **useState** hook is used to define state variables **email** and **password**.
   * The **email** and **password** variables are initially set to empty strings.
2. Navigation:
   * The **useNavigation** hook is used to access the navigation object, which allows navigation between screens.
   * The **navigation** object is used to navigate back to the previous screen when the back button is pressed.
3. useEffect Hook:
   * The **useEffect** hook is used to subscribe to the **onAuthStateChanged** event provided by Firebase Authentication.
   * It listens for changes in the user's authentication state. If a user is already logged in, it navigates directly to the 'MainPage' screen.
   * The effect runs only once when the component is mounted, thanks to an empty dependency array **[]**.
4. Event Handler:
   * The **handleLogin** function is called when the login button is pressed.
   * It uses the **signInWithEmailAndPassword** function from Firebase Authentication to authenticate the user.
   * If the login is successful, the logged-in user's email is logged to the console.
   * If there is an error, an alert is shown with the error message.

Overall

In summary, this code sets up a login screen where users can enter their email and password to log in. It uses Firebase Authentication's **signInWithEmailAndPassword** function to authenticate the user, and upon successful login, it navigates to the 'MainPage' screen.

**REGISTER SCREEN**

* + The **useState** hook is used to define state variables **email** and **password**, and their corresponding setter functions, **setEmail** and **setPassword**.
  + The **navigation** object is obtained using the **useNavigation** hook.

1. Event Handler:
   * The **handleSignUp** function is defined. It is called when the user presses the "Register" button.
   * It uses the **createUserWithEmailAndPassword** function from the Firebase **auth** object to create a new user with the provided email and password.
   * If the user creation is successful, the user's email is logged to the console, and the screen is navigated to the "MainPage" using the **navigation.replace** method.
   * If there is an error, an alert is displayed with the error message.
2. UI Elements:
   * The JSX defines the UI elements to be rendered on the screen.
   * It includes a **KeyboardAvoidingView** component that ensures the keyboard does not overlap with the input fields.
   * There is a "REGISTER" title, email and password input fields, a "Register" button, and a back button.
   * The **onChangeText** prop of the email and password inputs updates the corresponding state variables.
   * The **value** prop of the email and password inputs is set to the corresponding state variables.
   * The "Register" button calls the **handleSignUp** function when pressed.
   * The back button allows the user to navigate back to the previous screen.

Overall, this code provides a user interface for registration, where users can enter their email and password and register as new users in the application.

**PROFILE PAGE**

1. State Variables:
   * The component uses several state variables:
     + **modalVisible**: Controls the visibility of a modal.
     + **selectedEvent**: Stores the selected event.
     + **userEmail**: Stores the user's email obtained from Firebase authentication.
     + **todoData**: Stores an array of todo items fetched from the Firebase Realtime Database.
     + **lang**: Stores the current language, with an initial value of **en**.
2. useEffect Hook:
   * The **useEffect** hook is used to perform side effects in the component.
   * The first **useEffect** fetches the user's email from Firebase authentication and sets it in the **userEmail** state variable.
   * The second **useEffect** updates the **lang** state whenever the **props.route.params.language** changes. This is likely used for language translation purposes.
3. Event Handler:
   * The **handleSignOut** function is defined. It is called when the user presses the "Sign Out" button.
   * It uses the **signOut** method from the Firebase **auth** object to sign out the current user.
   * If the sign out is successful, the screen is navigated to the "HomeScreen" using the **navigation.replace** method.
   * If there is an error, an alert is displayed with the error message.
4. useEffect with Firebase Realtime Database:
   * The **useEffect** hook is used to fetch todo items from the Firebase Realtime Database.
   * It sets up a listener on the "posts/" reference in the database.
   * Whenever the value of the "posts/" reference changes, the callback function is called with a snapshot of the data.
   * The data is then transformed into an array of objects and stored in the **todoData** state variable.

Overall, this code represents a profile screen in a React Native app. It displays the user's profile information, including their email and profile picture. It also fetches and displays the user's events from a Firebase Realtime Database. The user can sign out from the app by pressing the "Sign Out" button.

**EVENTS**

* + The component uses two state variables:
    - **todoData**: Stores an array of todo items fetched from the Firebase Realtime Database.
    - **lang**: Stores the current language, with an initial value of **en**.

1. useEffect Hook:
   * The **useEffect** hook is used to fetch todo items from the Firebase Realtime Database.
   * It sets up a listener on the "posts/" reference in the database.
   * Whenever the value of the "posts/" reference changes, the callback function is called with a snapshot of the data.
   * The data is then transformed into an array of objects and stored in the **todoData** state variable.
2. useEffect with Language Update:
   * The second **useEffect** hook updates the **lang** state whenever the **props.route.params.language** changes. This is likely used for language translation purposes.
3. Event Handler:
   * The **handleLikeButtonPress** function is defined. It is called when the user presses the "Like" button.
   * It toggles the **pressed** property of the corresponding todo item in the **todoData** state array.
4. UI Elements:
   * The JSX defines the UI elements to be rendered on the screen.
   * It includes a **ScrollView** component to display a scrollable list of events.
   * For each todo item in the **todoData** array, a **TouchableOpacity** component is rendered.
   * Inside each **TouchableOpacity**, there are nested views to display the event details, including the publisher's name, event title, and event description.
   * There is also a "Like" button rendered as a **TouchableOpacity** with an icon from the **AntDesign** library.

Overall, this code represents a component that renders a scrollable list of events. Each event is displayed in a box-like layout, including the publisher's name, event details, and a "Like" button. The component fetches the event data from a Firebase Realtime Database and allows the user to toggle the "Like" status of each event. The component also supports language translation, updating the UI based on the selected language.

**Create Event**

1. State Variables:
   * The component uses several state variables:
     + **title**: Stores the value of the event title entered by the user.
     + **description**: Stores the value of the event description entered by the user.
     + **userEmail**: Stores the current user's email obtained from the **auth.currentUser** object.
     + **lang**: Stores the current language, with an initial value of **en**.
2. **addPost** Function:
   * The **addPost** function is defined to add a new post to the Firebase Firestore database.
   * It uses the **addDoc** function from the Firestore library to add a document to the 'posts' collection.
   * The document includes the **title** and **description** properties, which are obtained from the component's state.
   * It also logs a custom event using Firebase Analytics.
   * If an error occurs during the database operation, an error message is logged to the console.
3. **useEffect** Hooks:
   * The first **useEffect** hook is used to retrieve the current user's email and set it to the **userEmail** state variable. This is done when the component mounts.
   * The second **useEffect** hook updates the **lang** state whenever the **props.route.params.language** changes. This is likely used for language translation purposes.
4. **addData** Function:
   * The **addData** function is called when the user clicks the "Create Event" button.
   * It uses the **set** function from the Firebase Realtime Database to add the event data to the 'posts/' reference.
   * The event data includes the **title**, **description**, and **userEmail** properties.
   * After adding the data, the **setTitle** and **setDescription** functions are called to reset the input fields.
   * The **addPost** function is also called, passing the **title** and **description** as parameters.

Overall, this code represents a component that allows the user to create a new event by entering a title and description. The event data is then added to both the Firebase Firestore and Realtime Database. The component also retrieves the user's email and supports language translation.