Firebase Real-time Database in React

In this section, we'll be introduced to the Firebase Realtime Database and its functionality.

WE'LL COVER THE FOLLOWING

Storing User Data

So far, only Firebase knows about our users. There is no way to retrieve a single user or a list of users for our application from Firebase's authentication database.

The users are stored internally by Firebase to keep the authentication secure. That's good because we don't need to be involved in storing sensitive data like passwords. However, we can introduce the Firebase Real-time Database to keep track of user entities ourselves.

It makes sense because then we can associate users with other domain entities (e.g. a message, a book, an invoice) created by them.

We should maintain a degree of control over our users, even if Firebase takes care of all the sensitive data.

This section will explain how we can store users in our Firebase Real-time Database. First, we must initialize the Real-time Database API for our Firebase class.

Initializing the Real-time Database API

This technique is identical to the one we used earlier for the authentication API:

```
import app from 'firebase/app';
import 'firebase/auth';

import 'firebase/database';

const config = { ... };

class Firebase {
  constructor() {
    app.initializeApp(config);

    this.auth = app.auth();
    this.db = app.database();
  }

  // *** Auth API ***

  ...
}

export default Firebase;
```

Firebase/firebase.js

The second step is to extend the interface for our Firebase class for the user entity. It defines two new functions:

- One to get a reference to a user by the identifier (`uid`)
- One to get a reference to all users

```
import app from 'firebase/app';
import 'firebase/auth';
import 'firebase/database';

const config = { ... };

class Firebase {
  constructor() {
    app.initializeApp(config);

    this.auth = app.auth();
    this.db = app.database();
  }

  // *** Auth API ***

  doCreateUserWithEmailAndPassword = (email, password) =>
    this.auth.createUserWithEmailAndPassword(email, password);

  doSignInWithEmailAndPassword = (email, password) =>
    this.auth.signInWithEmailAndPassword(email, password);

  doSignOut = () => this.auth.signOut();

  doPasswordReset = email => this.auth.sendPasswordResetEmail(email);
```

```
  doPasswordUpdate = password =>
    this.auth.currentUser.updatePassword(password);


  // *** User API ***

  user = uid => this.db.ref(`users/${uid}`);

  users = () => this.db.ref('users');
}

export default Firebase;
```

Firebase/firebase.js

The paths in the `ref()` method match the location where our entities (users) will be stored in Firebase's Real-time Database API.

If we delete a user at "users/5", the user with the identifier `uid = 5` will be removed from the database.

If we create a new user in "users", Firebase creates the identifier for us and assigns to it all the information we passed.

The paths follow the REST philosophy where every entity (e.g. user, message, book, author) is associated with a URI, and HTTP methods are used to create, update, delete and get entities.

In Firebase, the RESTful URI becomes a simple path, and the HTTP methods become Firebase's API.

Next, we will run and verify our app after the integration of Real-time Database in our app.