Initializing Firebase

In this lesson, we'll initialize Firebase using a separate class.

WE'LL COVER THE FOLLOWING

Importing Firebase

Let's import Firebase from the library we installed earlier, and then use it within a new Firebase class to initialize Firebase with the configuration we saw in the previous lesson:

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

const config = {
  apiKey: process.env.REACT_APP_API_KEY,
  authDomain: process.env.REACT_APP_AUTH_DOMAIN,
  databaseURL: process.env.REACT_APP_DATABASE_URL,
  projectId: process.env.REACT_APP_PROJECT_ID,
  storageBucket: process.env.REACT_APP_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_MESSAGING_SENDER_ID,
};

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

export default Firebase;
```

Firebase/firebase.js

That's all we really need for a Firebase configuration in our application.

Production vs. Environment

Optionally, we can create a second Firebase project on the Firebase website

Optionally, we can create a second Firebase project on the Firebase website. One project can serve as our development environment while the other can be used as the production environment.

That way, the data in the development mode does not get mixed with the data from our deployed application (production mode).

If we decide to create projects for both environments, we'd use the two configuration objects in our Firebase setup and decide which one to take depending on the development/production environment:

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

const prodConfig = {
  apiKey: process.env.REACT_APP_PROD_API_KEY,
  authDomain: process.env.REACT_APP_PROD_AUTH_DOMAIN,
  databaseURL: process.env.REACT_APP_PROD_DATABASE_URL,
  projectId: process.env.REACT_APP_PROD_PROJECT_ID,
  storageBucket: process.env.REACT_APP_PROD_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_PROD_MESSAGING_SENDER_ID,
};

const devConfig = {
  apiKey: process.env.REACT_APP_DEV_API_KEY,
  authDomain: process.env.REACT_APP_DEV_AUTH_DOMAIN,
  databaseURL: process.env.REACT_APP_DEV_DATABASE_URL,
  projectId: process.env.REACT_APP_DEV_PROJECT_ID,
  storageBucket: process.env.REACT_APP_DEV_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_DEV_MESSAGING_SENDER_ID,
};

const config =
  process.env.NODE_ENV === 'production' ? prodConfig : devConfig;

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

export default Firebase;
```

Firebase/firebase.js

An alternate way to implement this is to specify a dedicated `.env.development` and `.env.production` file for both kinds of environment variables in our project.

Each file is used to define environment variables for the matching environment. Defining a configuration becomes straightforward again, because we don't have to select the correct configuration ourselves

because we don't have to select the correct configuration ourselves.

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

const config = {
  apiKey: process.env.REACT_APP_API_KEY,
  authDomain: process.env.REACT_APP_AUTH_DOMAIN,
  databaseURL: process.env.REACT_APP_DATABASE_URL,
  projectId: process.env.REACT_APP_PROJECT_ID,
  storageBucket: process.env.REACT_APP_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_MESSAGING_SENDER_ID,
};

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

export default Firebase;
```

Firebase/firebase.js

Whether we use environment variables, define the configuration inline, use only one Firebase project, or multiple projects for each environment, we have finally configured Firebase for our React application!

In the next lesson, you can run and verify if your app is working after adding the Firebase configuration on our live terminal!