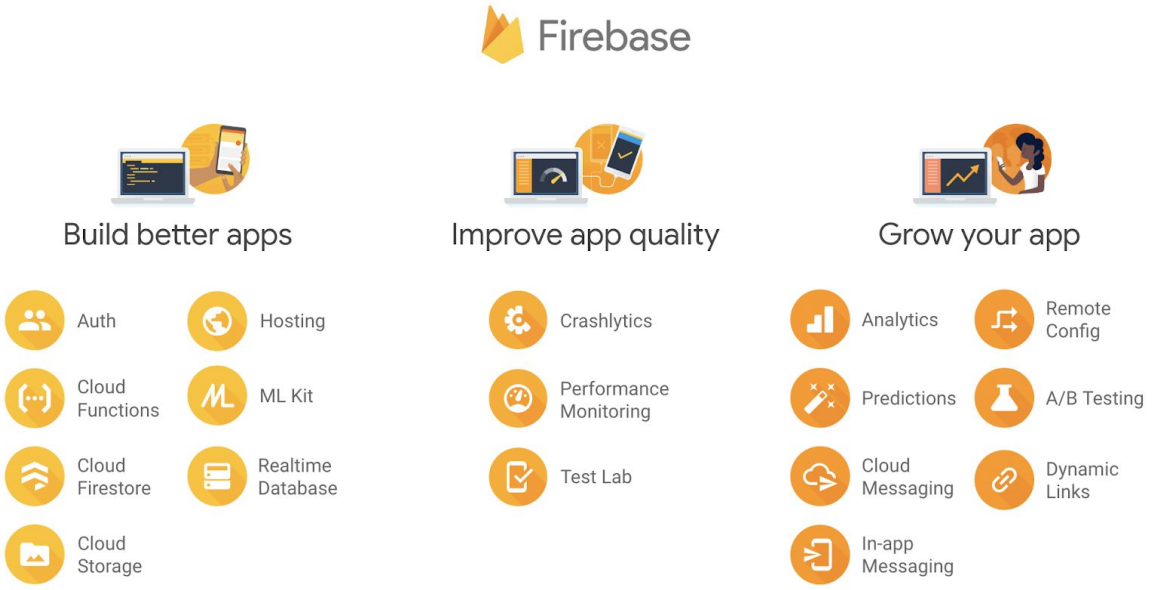
Intro to Firebase

We will explore Firebase and all the tools and services that it provides. Later on we are going to implement it, on our TodoApp, the one we created with ReactJs.

1.What is Firebase?

Firebase is a powerful platform for your mobile and web application. Firebase can power your app's backend, including data storage, user authentication, static hosting, and more. With Firebase, you can easily build mobile and web apps that scale from one user to one million.

2.Firebase Services



With Firebase you can do a lot of different things, so let's go through them one by one:

2.1 Build better apps

Firebase lets you build more powerful, secure and scalable apps, using world-class infrastructure:

1. *Cloud Firestore:* Is a flexible, scalable database for mobile, web, and server development from Firebase and GCP. It is a NoSQL

document database that lets you easily store, sync, and query data.It's supported for Android, iOS and Web Platform.

2. *ML Kit:* Is a mobile SDK that brings Google's machine learning expertise to Android and iOS apps in a powerful yet easy-to-use package. Whether you're new or experienced in machine learning, you can implement the functionality you need in just a few lines of code. There's no need to have deep knowledge of neural networks or model optimization to get started. On the other hand, if you are an experienced ML developer, ML Kit provides convenient APIs that help you use your custom TensorFlow Lite models in your mobile apps. It's supported for Android and iOS Platform.
3. *Cloud Functions:* Lets you automatically run backend code in response to events triggered by Firebase features and HTTPS requests. Your code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale your own servers. It's supported for Android, iOS, C++, Unity and Web Platform.
4. *Authentication:* Provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more. It's supported for Android, iOS and Web Platform.
5. *Hosting:* With a single command, you can quickly deploy web apps and serve both static and dynamic content to a global CDN (content delivery network). You can also pair Firebase Hosting with Cloud Functions to build and host microservices on Firebase. It's supporting only Web Platform.
6. *Cloud Storages:* The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content. On the server, you can use Google Cloud Storage, to access the same files. It's supported for Android, iOS, C++, Unity and Web Platform.
7. *Real-time Database:* Is a cloud-hosted NoSQL database that lets you store and sync between your users in real-time. The Real-time Database is really just one big JSON object that the developers can manage in real-time. It's supported for Android, iOS, C++, Unity and Web Platform.

2.2 Improve app quality

Firebase gives you insights into app performance and stability, so you can channel your resources effectively using:

1. *Crashlytics:* is a lightweight, real-time crash reporter that helps you track, prioritize, and fix stability issues that erode your app quality. Crashlytics saves you troubleshooting time by intelligently grouping crashes and highlighting the circumstances that lead up to them. It's supported for Android and iOS Platform.
2. *Performance Monitoring:* is a service that helps you to gain insight into the performance characteristics of your iOS and Android apps. You use the Performance Monitoring SDK to collect performance data from your app, and then review and analyze that data in the Firebase console. Performance Monitoring helps you to understand where and when the performance of your app can be improved so that you can use that information to fix performance issues. It's supported for Android and iOS Platform.
3. *Test Labs:* is a cloud-based app-testing infrastructure. It provides a large number of mobile test devices to help you test your apps. It's supported for Android and iOS Platform.

2.3 Grow your business

Firebase helps you grow to millions of users, simplifying user engagement and retention using:

1. *In-App Messaging:* helps you engage users who are actively using your app by sending them targeted and contextual messages — like beating a game level, buying an item, or subscribing to content. It's supported for Android and iOS Platform.
2. *Google Analytics:* for Firebase is a free app measurement solution that provides insight on app usage and user engagement. Analytics integrates across Firebase features and provides you with unlimited reporting for up to 500 distinct events that you can define using the Firebase SDK. Analytics reports help you understand clearly how your users behave, which enables you to make informed decisions regarding app

marketing and performance optimizations. It's supported for Android, iOS, C++ and Unity Platform.

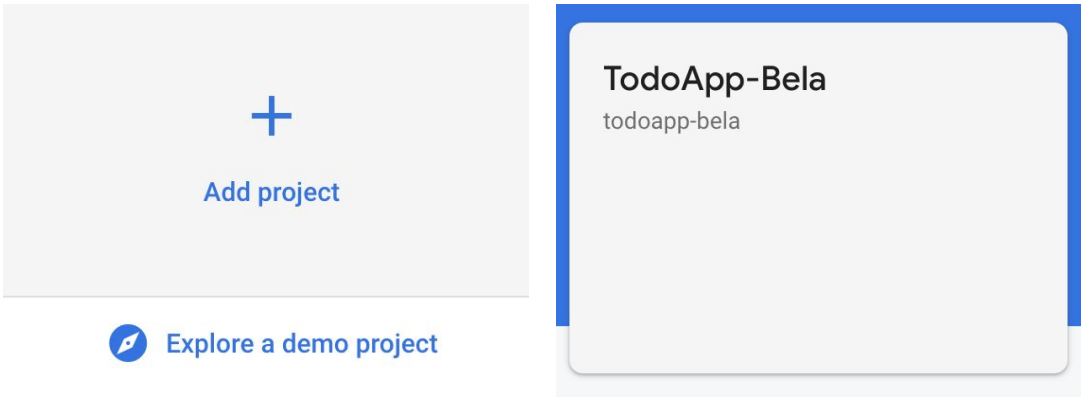
3. *Predictions:* applies machine learning to your analytics data to create dynamic user segments based on the predicted behavior of users in your app. These predictions are available for use with Firebase Remote Config, the Notifications composer, Firebase In-App Messaging, and A/B Testing. You can also link your app's Predictions data to BigQuery so you can get daily exports that you can further analyze or push to third party tools. It's supported for Android, iOS, C++ and Unity Platform.
4. *A/B Testing:* helps you optimize your app experience by making it easy to run, analyze, and scale product and marketing experiments. It gives you the power to test changes to your app's UI, features, or engagement campaigns to see if they actually move the needle on your key metrics (like revenue and retention) before you roll them out widely. It's supported for Android, iOS, C++ and Unity Platform.
5. *Cloud Messaging (FCM):* provides connection between your server and devices that allows you to deliver and receive messages and notifications on iOS, Android, and the web at no cost.
6. *Remote Config:* essentially allows us to publish updates to our users immediately. Whether we wish to change the color scheme for a screen, the layout for a particular section in our app or show promotional/seasonal options — this is completely doable using the server side parameters without the need to publish a new version. It's supported for Android, iOS, C++ and Unity Platform.
7. *Dynamic Links:* With Dynamic Links, your users get the best available experience for the platform they open your link on. If a user opens a Dynamic Link on iOS or Android, they can be taken directly to the linked content in your native app. If a user opens the same Dynamic Link in a desktop browser, they can be taken to the equivalent content on your website. It's supported for Android, iOS, Web, C++ and Unity Platform.
8. *App Indexing:* gets your app into Google Search. If users have your app installed, they can launch your app and go directly to the content they're searching for. App Indexing reengages your app users by helping them find both public and personal content, even offering query autocompletion to help them more quickly find what they need. If users don't yet have your app,

relevant queries trigger an install card for your app in Search results. It's supported for Android and iOS Platform.

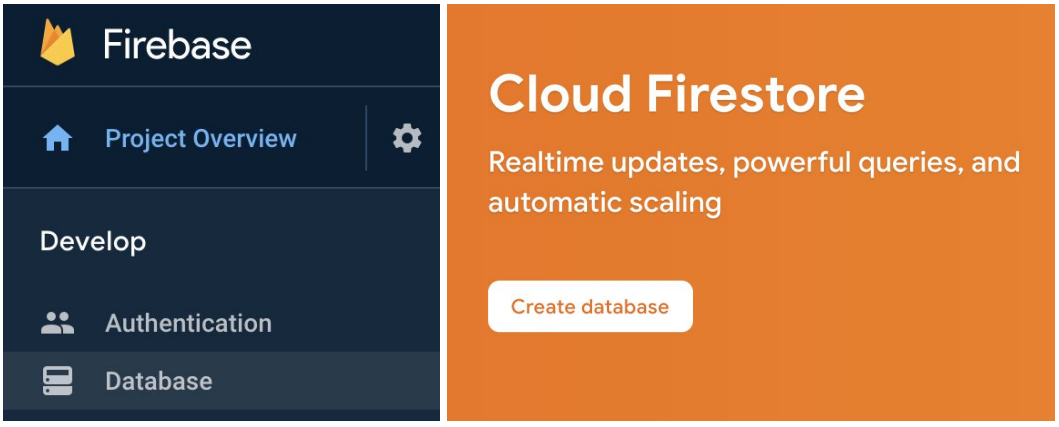
3. Cloud Firestore

We'll focus on their database solution called Cloud Firestore. Follow the steps below:

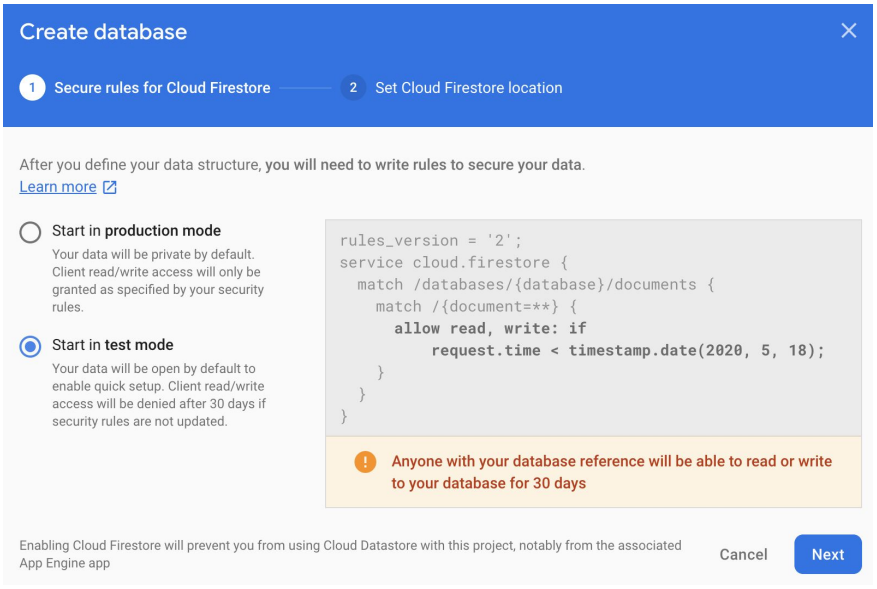
1. Go to the firebase's developer console.
2. Click *Add Project*.

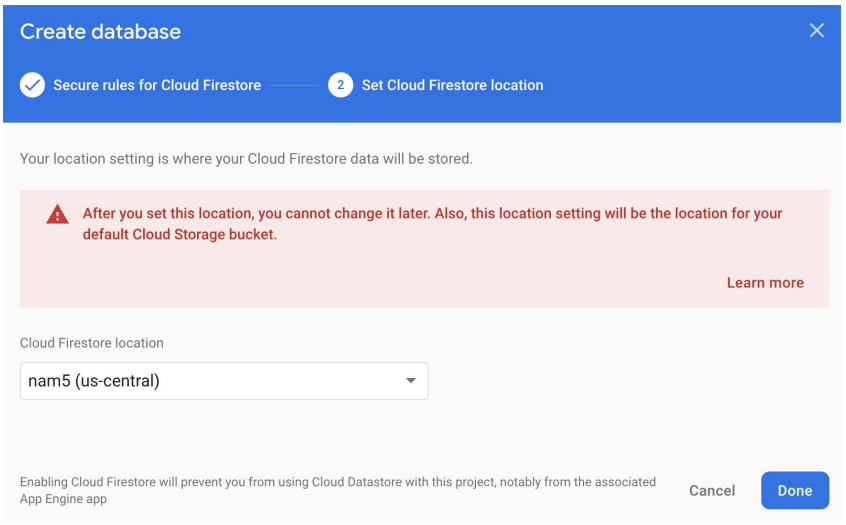


3. After creating it, go to your project and click Database -> Create Database

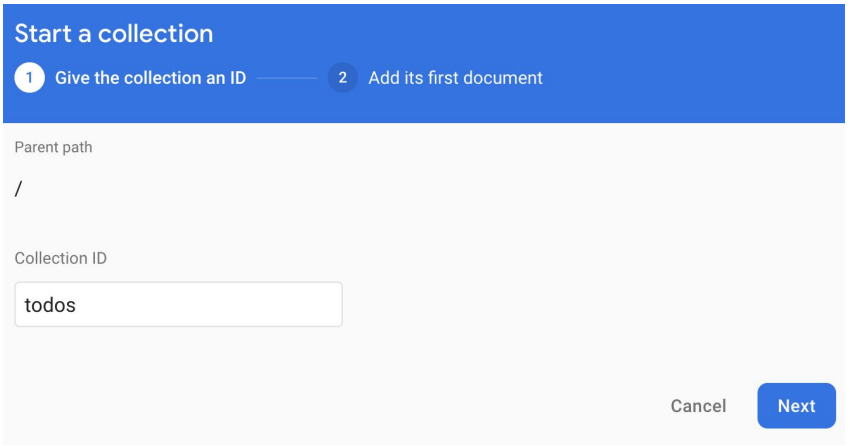


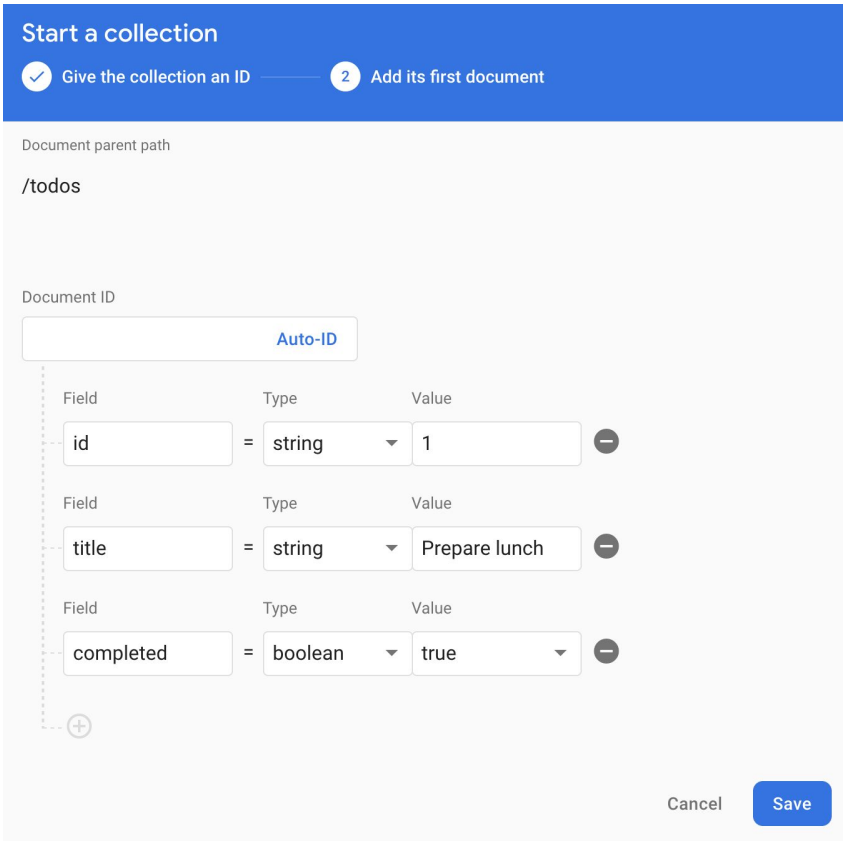
To be able to create the database you have to first start in test mode and then select the location (the location can not be changed later). After these steps the database is set up and ready to create documents and collections.



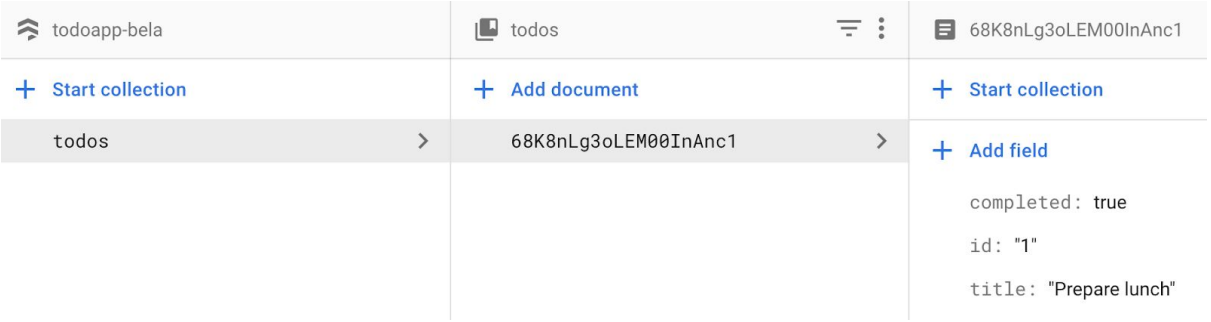


4. Add collection (Table in SQL) and your first Document (A row in SQL)





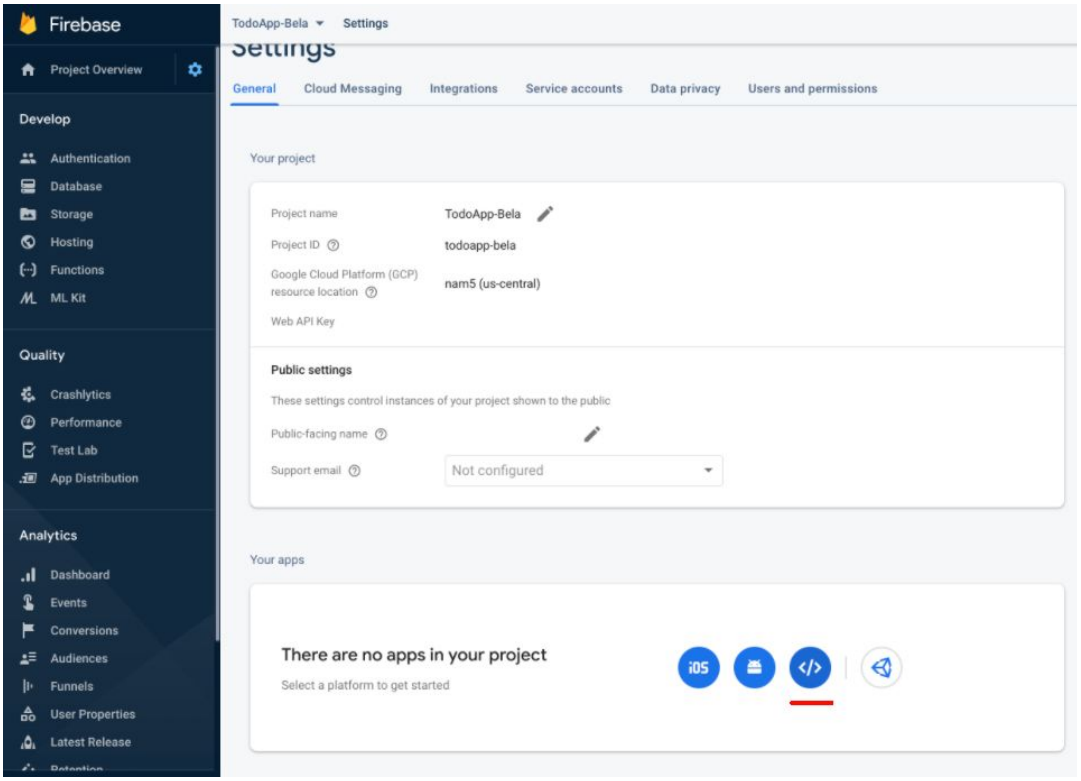
Cool, we've created a collection with a single entry (document). Now we're going to query this.



First, we need the credentials for the database which you can find under Project Overview.

We need to add a web app at the red underline in the picture below. After that a window will pop up with the firebase configuration. I also screenshotted this step, but I hid my keys.

! Remember it's good to keep some keys private :)



Add Firebase to your web app

Register app

2 Add Firebase SDK

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/7.14.1/firebase-app.js"></scrip

<!-- TODO: Add SDKs for Firebase products that you want to use
     https://firebase.google.com/docs/web/setup#available-libraries -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey:
    authDomain:
    databaseURL:
    projectId:
    storageBucket:
    messagingSenderId:
    appId:
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

Learn more about Firebase for web: Get Started, Web SDK API Reference, Samples

Continue to console

4. Init Database

We use Firebase module to access the Firebase Firestore Database in React. The Firebase module is available as an npm module, so we just need to type this command to install the module.

npm install --save firebase

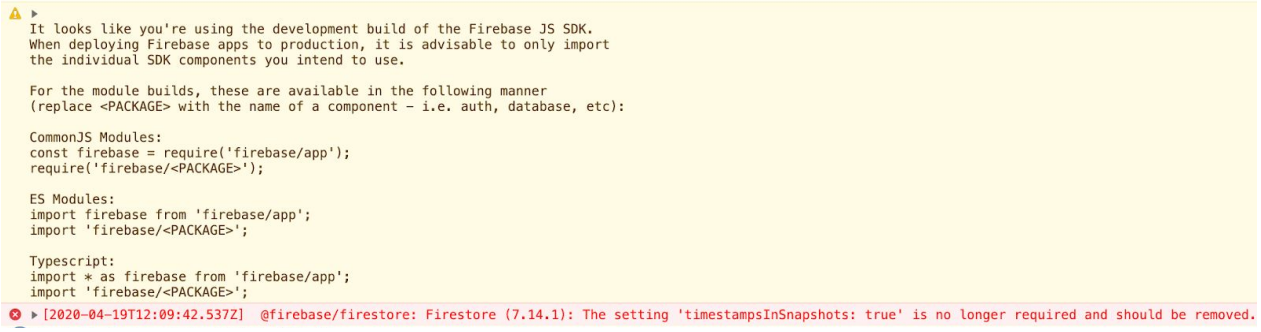
Next, we create a new file called Firebase.js in the root of the project folder. This file will contain the Firebase configuration. Please fill them based on **your** configuration.

```
import * as firebase from 'firebase';
import firestore from 'firebase/firestore'

const settings = {timestampsInSnapshots: true};

const config = {
 apiKey: "YOUR_API_KEY",
 authDomain: "YOUR_AUTH_DOMAIN",
 databaseURL: "YOUR_DATABASE_URL",
 projectId: "YOUR_PROJECT_ID",
 storageBucket: "YOUR_STORAGE_BUCKET",
 messagingSenderId: "YOUR_MESSAGING_ID"
};
firebase.initializeApp(config);

firebase.firestore().settings(settings);

export default firebase;
```

Now i need to import it at App.js like this:

```
import firebase from "./Firebase";
```

You will probably get warnings like the ones below:



The first warning we get is not breaking the code, but is more like an advice on how to improve the efficiency. You do not need to import all Firebase, as we said that it has a lot of packages, but only the one package you need. We can fix it by changing:

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

The second one occurred because the setting “timestampsInSnapshots” is no longer required. In most of the examples online you will find it because it was necessary before. Lets remove the line with settings and also change this line:

```
firebase.firestore().settings({});
```

With this step we finished setting up the database. Now let's add some functionalities on them.

5. Read Document (R)

Till now we get the todos from a json placeholder. This step should read the data from the database, so it should show us every document that is inside the collection todos into our UI.

We first are going to deconstruct the database reference, so we don't need to write it always. We add this before the state:

```
db = firebase.firestore().collection('todos')
```

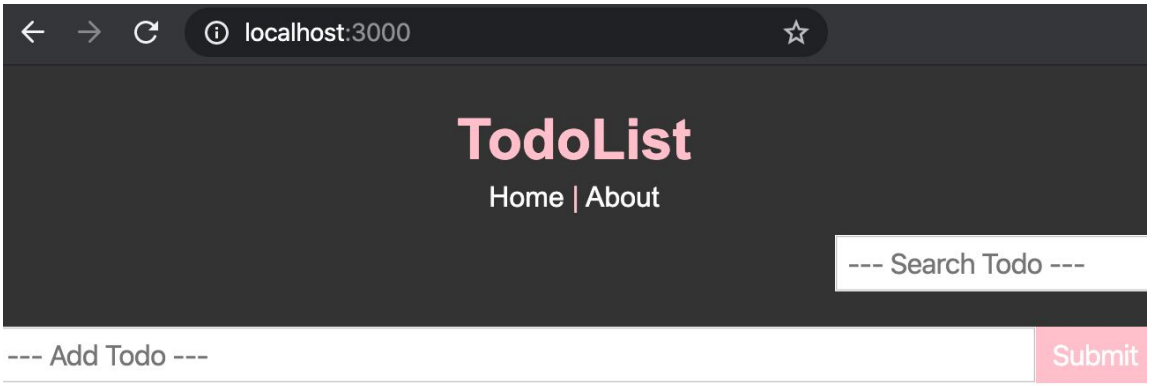
Later we need that every time we render our app, the data comes from the database, so in the componentDidMound we snapshot our documents with the function onCollectionUpdate.

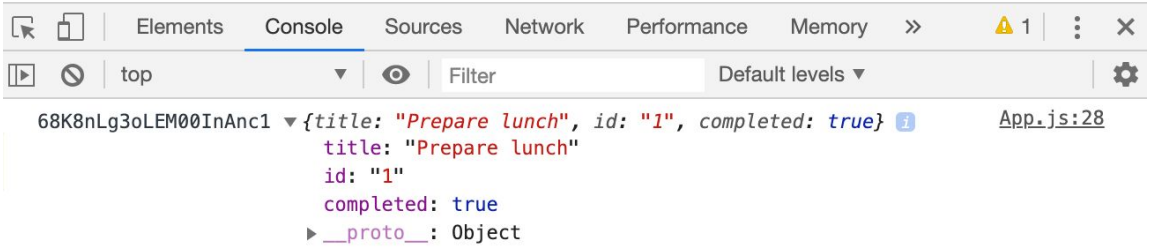
```
componentDidMount() {
this.db.onSnapshot(this.onCollectionUpdate);
}
```

This function will return just a console for what we have in the database. It will not show anything on the UI. First we need to prove that we connected the database.

```
onCollectionUpdate = (docSnapshot) => {
  docSnapshot.forEach((doc) => {
    console.log(doc.id, doc.data());
  })}
```

This is what we will get so far:

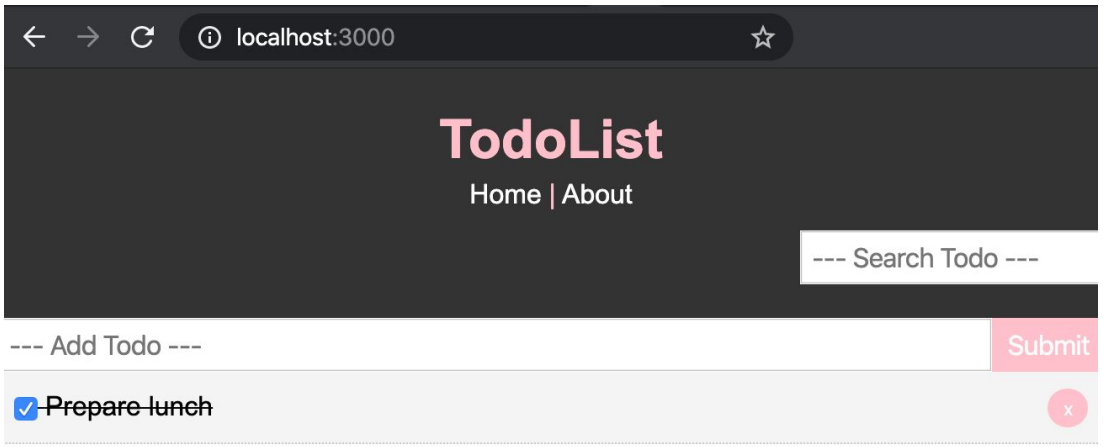


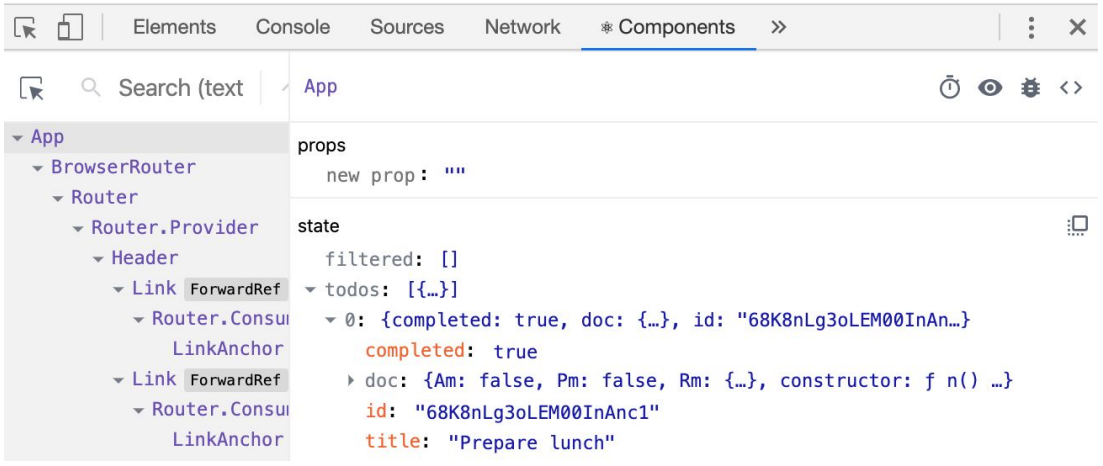


Now we need to update the UI with the data from the database. In order to do so, we need to push the data into the todos state.

```
onCollectionUpdate = (docSnapshot) => {
  const todos = [];
  docSnapshot.forEach((doc) => {
    const { title, completed } = doc.data();
    todos.push({
      id: doc.id,
      title,
      completed,
      doc // DocumentSnapshot
    });
  });
  this.setState({
    todos: todos
  });
}
```

We are done reading the data from the database, as you can see below:



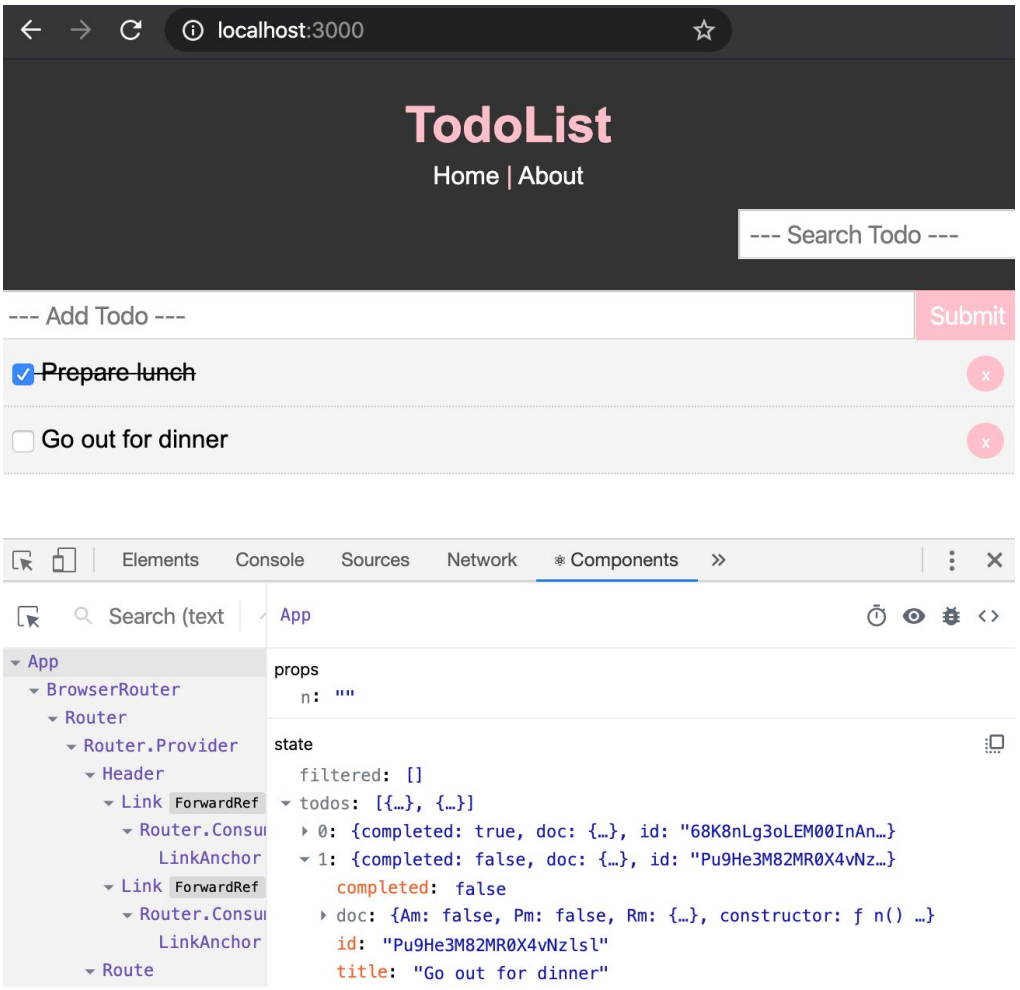


6. Add Document (C)

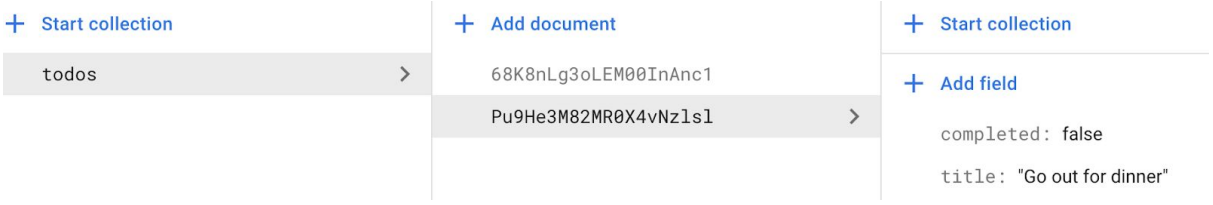
The idea is that when a new item is added into the Todos, it should also be added on the database. We need to change the addTodo function. We do not need anymore the uuidv4 because now every document can be unique with the firestore id:

```
addTodo = (title) => {
  this.db.add({
    title,
    completed: false
  }).then(() => {
    console.log("The todo is added");
  })
    .catch((error) => {
      console.error("Error adding document: ", error);
    });
};
```

This is what we will get at the React Developer Tool when we submit a new todo:



The picture below shows that this todo is also saved in the database:

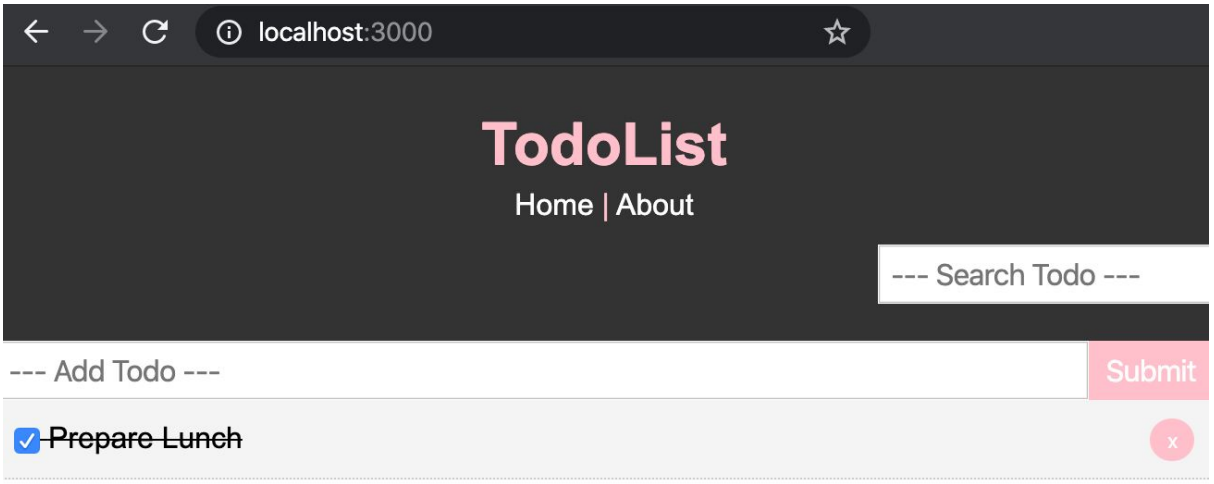


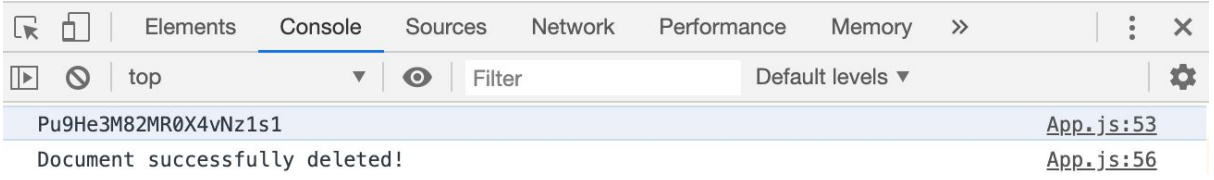
7. Delete Document (D)

The idea is that when a todo is deleted, it should also be removed from the database. After this step we do not need to use axios anymore, so we can remove its import. We need to change the deleteTodo function. The id that will be passed to this function is the doc.id from the firestore.

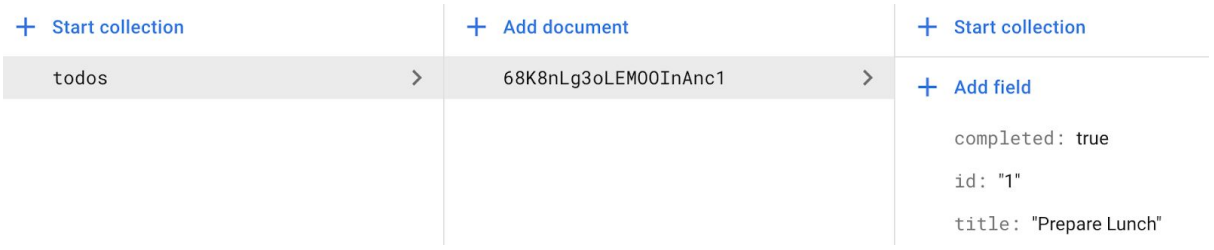
```
deleteTodo = (id) => {
  console.log(id)
  this.db.doc(id).delete()
    .then(() => {
      console.log("Document successfully deleted!")
    })
    .catch((error) => {
      console.error("Error removing document: ", error);
    });
};
```

The picture below shows the console log after deleting the “Go out for dinner”. I logged the todo’s id to show that is actually the auto-created id from firestore.





As you can see the todo was also deleted from the database.

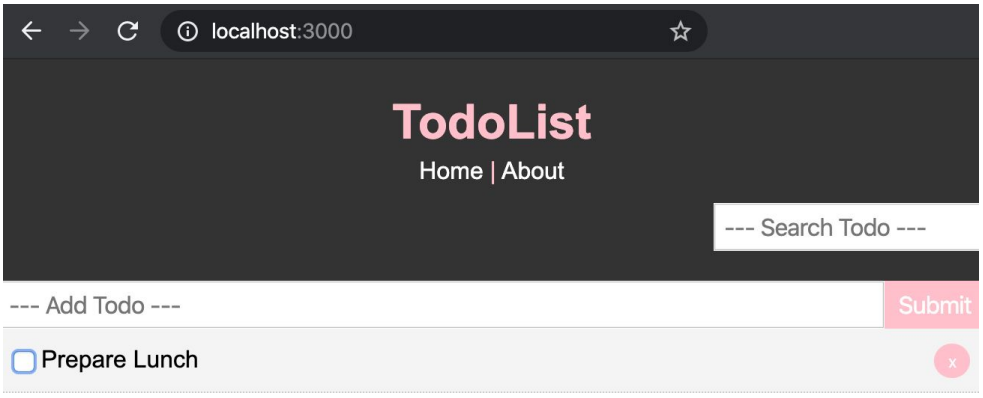


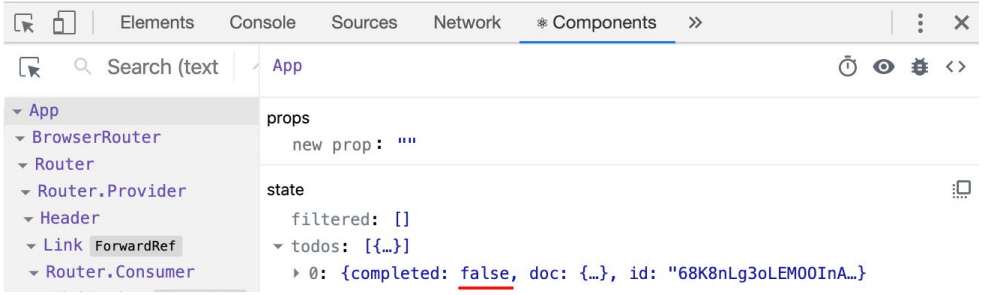
8. Update Document (U)

We need to update a todo, from not completed to completed. For this reason we used the markComplete function, but now also this function needs to include the database. Why? Because every time I want to change the boolean value of completed I also want to save this in the database.

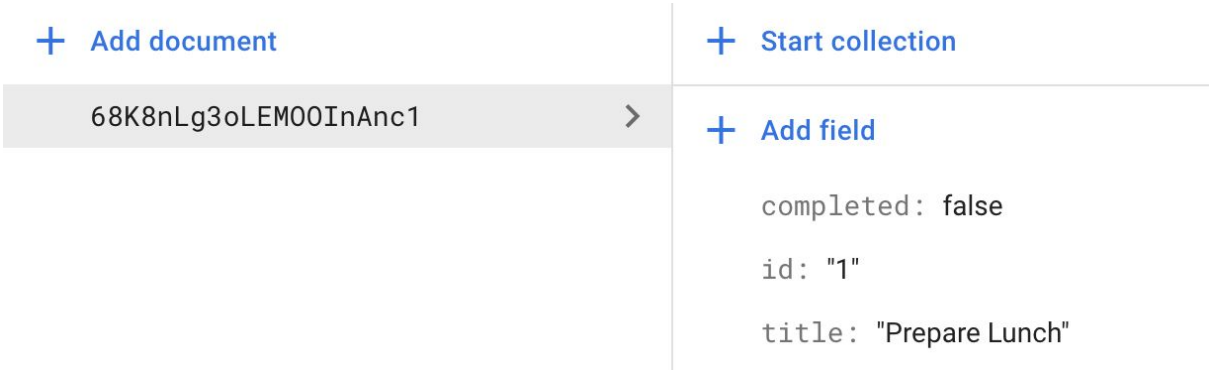
```
markComplete = (id) => {
  this.db.doc(id).get()

    .then((doc) => {
      if (doc.exists) {
        const { completed } = doc.data();
        this.db.doc(id).update({
          completed: !completed
        })
      } else {console.log("No such document!");}
    })
    .catch((error) => {
      console.error("Error updating document: ", error);
    });
};
```

The best approach is to do it with componentDidMount and onSnapshot, but i want to show that we can also use get to take a snapshot of the document and later change a field of it with the update function.





As you can see the completed field was also changed in the database.



! For the last functionality Search, it is almost the same as with only ReactJs, you just need to add the line below at onCollectionUpdate:

```
filtered: todos
```

We went through so far: **Create, Read, Update and Delete** (**CRUD**), that are the four basic functions of every action with a database.

9. Save you credentials at .env

You can save different environment variables in an .env file. You probably have seen this before, but why we want to keep them secret?

1. We do not want to push them into github and make them public for everyone, because they are consuming sensitive data that lives outside of version control. So if we push a code with a sensitive key, someone can use it and take advantage of our mistake.

2. We want to customize our app based on different variables whether the project is in development or deployment mode and if it is staging or production.

You can read more about env here. Now lets implement it on the firebase.js:

1. Install this package

   npm install dotenv --save

2. Add the following line into firebase.js:

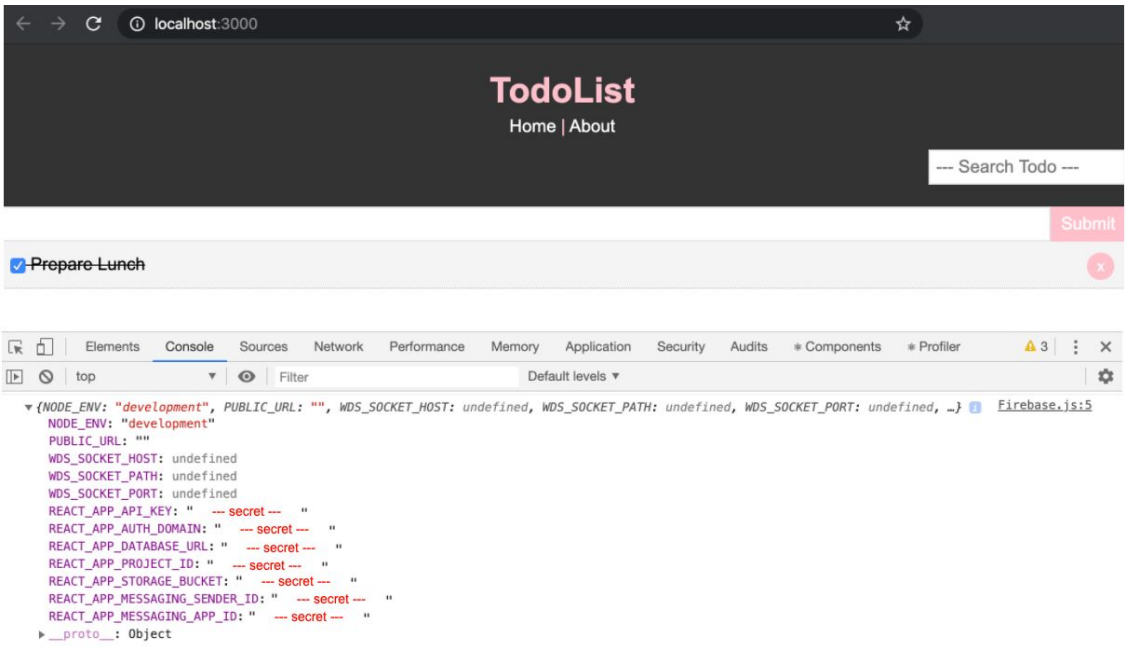
```
require('dotenv').config()
```

3. Change the config part like below:

```
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,
 aapId: process.env.REACT_APP_MESSAGING_APP_ID
 };
```

4. Then create a .env file at the root directory of your application and add the variables to it.

```
REACT_APP_API_KEY="..."
REACT_APP_AUTH_DOMAIN=...
REACT_APP_DATABASE_URL=...
REACT_APP_PROJECT_ID=...
REACT_APP_STORAGE_BUCKET=...
REACT_APP_MESSAGING_SENDER_ID="..."
REACT_APP_MESSAGING_APP_ID="..."
```



I console.log (process.env) and I can see all the variables I passed into .env. In this way we can check if everything is working.

5. The .env file can also be added to your .gitignore file, so your Firebase credentials are not exposed publicly on a platform like GitHub.

10. Homework

Implement *the homework of the previous ReactJS tutorial* in Firestore, so that everytime you update the title it should also update the corresponding document in the database.

Github Link: https://github.com/bsinoimeri/TodoApp-Firebase

Demo Link: https://todoapp-bela.firebaseapp.com