**React + Cloud Firestore: Step by step tutorial -** [jason ewins](https://medium.com/@jtradex?source=post_page-----c78f198d2364--------------------------------) Jul 2, 2018

In their own words, Firestore is Google’s flexible, scalable NoSQL cloud database to store and sync data for client- and server-side development.

So, if you’re looking to create a fully-fledged app, web or native, Google’s Firestore offers a ‘path of least resistance’ route to getting up and running with a real-time database, quickly.

Great for those of us working towards an MVP and scalable for those ideas which gain traction (shouldn’t we be so lucky).

In October last year, Google released the Firestore beta (a pretty stable beta). Prior to that you may be familiar the Firebase product ([Google acquired Firebase in 2014 — link to Techcrunch announcement](https://techcrunch.com/2014/10/21/google-acquires-firebase-to-help-developers-build-better-realtime-apps/)) Here’s a quick two minute intro to Firestore:

A quick Firestore intro — looking sweet!

**How to get Firestore up and running with your React app — super quick**

**Step 1:**Open up your preferred terminal and install ***firebase****with npm**(that’s****firebase****not firestore — bare with me here)*

$ npm install firebase

You should now have firebase listed in your package.json file — At time of writing, we’re currently on version 5.1.0

**Step 2: Lets create our database —**Head over to the Firebase website: [https://firebase.google.com](https://firebase.google.com/) and hit the ‘Get Started’ button — use a preferred gmail account to sign-up. When you’re done with the signup you’ll land on the welcome page with a prompt to create your first project.

Diagram

Description automatically generated

Firebase welcome page — let’s hit that ‘+ Add project’ card and get building!

Select the *‘Add project’* card to create a new project. You’ll be presented with a modal asking you to create a name for your project and identify the region + the obligatory ‘sign your analytics life away’ should you wish to disclose.

Graphical user interface, text, application

Description automatically generated

You can always change your ‘project name’ at a later date through the settings panel

Next — we want to get the tracking code for our web app, hit the ‘Add Firebase to your web app’ option…

Graphical user interface, application

Description automatically generated

No prizes for guessing which icon to select…

You’ll be presented with your config code — copy and paste the code within the *<script>* tags (script tags not needed)

Graphical user interface, text

Description automatically generated

When I was a young lad we used ‘var’….

**Step 3:** Head back over to your React project and let’s create a new *Firestore.js* component (or whatever you would like to name it). Import firebase into the component and then paste-in the config code. Don’t forget to export the component.

You should have something similar to this *don’t worry, the below keys are fictitious, I’m not that trusting!*

import firebase from ‘firebase’;const config = {  
 apiKey: "AIzfhrwkKhDBFHbntnKhTahHjkNK6FG875dSB6y",  
 authDomain: "typecode-76g33.firebaseapp.com",  
 databaseURL: "https://typecode-76g33.firebaseio.com",  
 projectId: "typecode-76g33",  
 storageBucket: "",  
 messagingSenderId: "547432956432"  
};  
firebase.initializeApp(config);export default firebase;

**Step 4:** Almost there with the Firestore setup, now we need to create our database. Back in Firebase, select the ‘Database’ option from the left-hand navigation and select ‘Create database’ on the Firestore cta.

Graphical user interface, website

Description automatically generated

Go create your database…

Whilst in development, we’ll relax the security preferences to allow *‘read’* & *‘writes’* to our database. *This will need to be addressed prior to any publishing of your app for obvious security reasons.*

Graphical user interface, text, application

Description automatically generated

We’ll use test mode whilst we’re in development

**Step 5:** Once enabled, you’ll be presented with your new database. Select the ‘*+ Add collection*’ option, and create a name for said collection. As highlighted in the tooltip, a collection is a set of documents which contain data. For our example, we’ll use the suggested ‘users’ for creating a database of user details.

Graphical user interface, text, application, chat or text message

Description automatically generated

After selecting ‘Next’ we have the option to add a *Document id* (unique reference) or we leave it blank for Firestore to create a random unique id per document. We’ll leave it blank for now.

For this example project, we’ll capture two pieces of data from our users, their full name and email address — both are set to string with blank values.

Graphical user interface, application

Description automatically generated

Once you’ve specified the data you’re looking to capture, click *‘save’.*You’ve now created your database, populated with an example document.

Time for a sneaky

Graphical user interface, text, application

Description automatically generated

**Step 6:**Ok, with our database created let’s get some data in there. In our project we’ll create a new component called User.js which contains a simple form.

import React from 'react';  
import firestore from "./firestore";class User *extends* React.Component {  
 render() {  
 return (  
 <form>  
 <input  
 *type*="text"  
 name="fullname"  
 *placeholder*="Full name"  
 />  
 <input  
 *type*="email"  
 name="email"  
 *placeholder*="Full name"  
 />  
 <button *type*="submit">Submit</button>  
 </form>  
 );  
 }  
 }export default User;

**Step 7:**Looking good. Now we need to get the data from the form, into the [*state*](https://reactjs.org/docs/faq-state.html)*object*(If your not familiar with React’s component state, have a quick run-through the [React docs here](https://reactjs.org/docs/faq-state.html)).

First we’ll create our initial ‘empty’ state inside a constructor function:

class User *extends* React.Component {  
 **constructor() {  
 super();  
 this.state = {  
 email: “”,  
 fullname: “”  
 };  
 }** ...

**Step 8:**Next we’ll add a custom attribute to **both input fields**, which will call a function ‘*updateInput’* to set our component state, when a user enters data into the form.

...**updateInput = e => {  
 this.setState({  
 [e.target.name]: e.target.value  
 });  
}**render() {  
 return (  
 <form>  
 <input  
 t*ype*="text"  
 name="fullname"  
 *placeholder*="Full name"  
 **onChange={this.updateInput}**  
 />  
 ...

**Let’s test this:** open up the page in the browser and using React developer tools (Chrome) search for the *User* component. Now you’ll be able to see the *state* updated as you type in the form fields — great job!

**Step 9:** Now we’ll add another custom attribute to our input fields. This will assign the data (in *state*) to our object value which get’s passed into Firestore. (that was succinct…!) like so:

...<form>  
 <input  
 *type*="text"  
 name="fullname"  
 *placeholder*="Full name"  
 onChange={this.updateInput)  
 **value={this.state.fullname}** />  
 <input  
 *type*="email"  
 name="email"  
 *placeholder*="Full name"  
 onChange={this.updateInput}  
 **value={this.state.email}**  
 />  
<button ...

Almost there with the form, we just need to add an attribute to call a function which does the magic when the form is submitted.

<form **onSubmit={this.addUser}**>  
 <input ...

**Step 10:**Still with me…? Awesome, we’re nearly there.  
Let’s create the addUser function which will send data to our Firestore database. We’ll use an arrow function to avoid having to bind *this* to the function in the constructor — if that doesn’t make sense, [read this](https://reactjs.org/docs/faq-functions.html#why-is-binding-necessary-at-all).

addUser = e => {  
 e.preventDefault();  
 this.setState({  
 fullname: “”,  
 email: “”  
 });  
};

Ok, here we’re doing two simple things.

1. *preventDefault()*method stops page from refreshing (default behavior)

2. After the user submits the form, the state is updated to an empty string to remove the user input across both fields.

Simple enough, now let’s get that data into Firestore:

addUser = e => {  
 e.preventDefault();  
 **const db = firebase.firestore();  
 db.settings({  
 timestampsInSnapshots: true  
 });**  
 **const userRef = db.collection(“users”).add({  
 fullname: this.state.fullname,  
 email: this.state.email  
 });**   
 this.setState({  
 fullname: “”,  
 email: “”  
 });  
};

Ok, let’s run through what we’re doing here:

1. With *firebase.firestore()* we’re initialising Firestore through firebase and saving to a variable.
2. *timestampsinSnapshots: true*without this, we get a warning in the console, which tells us the timestamps stored in Firestore will be read back as timestamp objects, as opposed to system date objects.
3. *db.collection(“users”)*is simply pointing to our database; the *collection* we created called *users.*
4. finally the*.add()* method is submitting our data object with the users full name and email taken from our updated state.

Hit *save,*open up the browser and complete the form, hit submit and…

Magic — you did it!

Graphical user interface, text, application, email

Description automatically generated

Great work — you’re now capturing user data in your Firestore db

As always, this is by no means the *only way, right way, best way* to get Firestore working with React. This is just one simple route to get things up and running, break down the learning barriers and either move on to another challenge or dive-in further. The [Firestore documentation](https://firebase.google.com/docs/firestore/quickstart) is fantastic, so definitely take a peek.