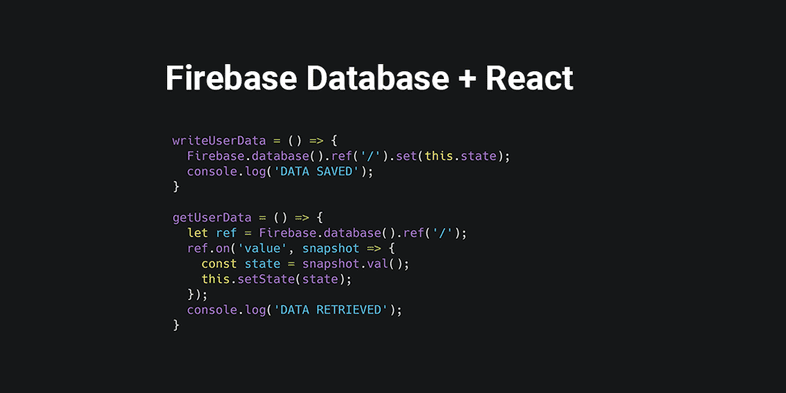
Firebase as simple database to React app

Posted on February 12, 2019



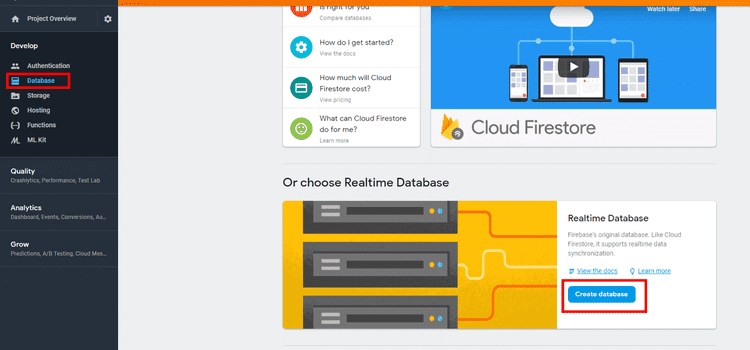
Firebase is an all-in-one backend as a service provider (BaaS) that provides database, authentication, cloud storage among their many services. In this tutorial you're going to learn how to use Firebase Real Time Database service in React application.

You're going to build a simple team list application, where users can add, delete and edit team member information.

Create your database application

First, you need to create your application in Firebase console.

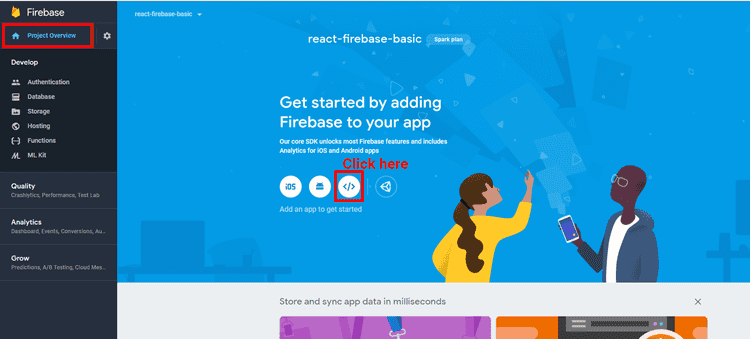
Then head over to the *Database* menu and scroll a bit down into *Choose Real Time Database* section.



Set the security rules to start in **test mode**.

(this makes your database insecure, but it's okay for the purpose of this tutorial.)

Finally, get the configuration needed to integrate Firebase into your web app.



Grab the source code from this codesandbox and fork it:

And put your credential in the config file:

const config = {

apiKey: "{YOUR KEY}",

authDomain: "{YOUR KEY}",

databaseURL: "{YOUR KEY}",

}

export default config;

Setting up your React application

Start your React application with create-react-app

npx create-react-app react-firebase-basic

Then install firebase and Bootstrap (so you can skip writing your own css.)

npm i firebase bootstrap

Then you can remove everything from src/ since you don't need most of the boilerplates

Creating config.js file

Let's write Firebase configuration in a separate config.js file:

const config = {

apiKey: "{YOUR KEY}",

authDomain: "{YOUR KEY}",

databaseURL: "{YOUR KEY}",

}

export default config;

You'll import this config into your App.js later.

Write the index.js file

This file will serve as React entry point:

import React from 'react';

import ReactDOM from 'react-dom';

import 'bootstrap/dist/css/bootstrap.css';

import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));

Creating your application

It's time to write your App.js file. Let's initialize our Firebase App in the constructor:

import React from 'react';

import Firebase from 'firebase';

import config from './config';

class App extends React.Component {

constructor(props){

super(props);

Firebase.initializeApp(config.firebase);

this.state = {

developers: []

}

}

//...

Then you can write the logic for getting and saving data: writeUserdata will write our state into the database, while getUserData will create a listener on / path, and on value changes, we will assign snapshot value as state.

writeUserData = () => {

Firebase.database().ref('/').set(this.state);

console.log('DATA SAVED');

}

getUserData = () => {

let ref = Firebase.database().ref('/');

ref.on('value', snapshot => {

const state = snapshot.val();

this.setState(state);

});

console.log('DATA RETRIEVED');

}

Put these writeUserData and getUserData in componentDidUpdate and componentDidMount respectively.

componentDidMount() {

this.getUserData();

}

componentDidUpdate(prevProps, prevState) {

// check on previous state

// only write when it's different with the new state

if (prevState !== this.state) {

this.writeUserData();

}

}

All that's left is to write the render and handle submit form logic: We will map our developers array from state and put each item in a card component. Each card will have a delete and update button. On delete clicked, we will filter out the specific item, while on update clicked, we will get the item data into the form.

handleSubmit will insert data when uid value is false and update data when it is true. We're using refs to get data from form inputs.

// ...

render() {

const { developers } = this.state;

return(

<div className="container">

<div className="row">

<div className='col-xl-12'>

<h1>Firebase Development Team</h1>

</div>

</div>

<div className='row'>

<div className='col-xl-12'>

{

developers

.map(developer =>

<div key={developer.uid} className="card float-left" style={{width: '18rem', marginRight: '1rem'}}>

<div className="card-body">

<h5 className="card-title">{ developer.name }</h5>

<p className="card-text">{ developer.role }</p>

<button onClick={ () => this.removeData(developer) } className="btn btn-link">Delete</button>

<button onClick={ () => this.updateData(developer) } className="btn btn-link">Edit</button>

</div>

</div>

)

}

</div>

</div>

<div className='row'>

<div className='col-xl-12'>

<h1>Add new team member here</h1>

<form onSubmit={ this.handleSubmit }>

<div className="form-row">

<input type='hidden' ref='uid' />

<div className="form-group col-md-6">

<label>Name</label>

<input type="text" ref='name' className="form-control" placeholder="Name" />

</div>

<div className="form-group col-md-6">

<label>Role</label>

<input type="text" ref='role' className="form-control" placeholder="Role" />

</div>

</div>

<button type="submit" className="btn btn-primary">Save</button>

</form>

</div>

</div>

</div>

)

}

handleSubmit = (event) => {

event.preventDefault();

let name = this.refs.name.value;

let role = this.refs.role.value;

let uid = this.refs.uid.value;

if (uid && name && role){

const { developers } = this.state;

const devIndex = developers.findIndex(data => {

return data.uid === uid

});

developers[devIndex].name = name;

developers[devIndex].role = role;

this.setState({ developers });

}

else if (name && role ) {

const uid = new Date().getTime().toString();

const { developers } = this.state;

developers.push({ uid, name, role })

this.setState({ developers });

}

this.refs.name.value = '';

this.refs.role.value = '';

this.refs.uid.value = '';

}

removeData = (developer) => {

const { developers } = this.state;

const newState = developers.filter(data => {

return data.uid !== developer.uid;

});

this.setState({ developers: newState });

}

updateData = (developer) => {

this.refs.uid.value = developer.uid;

this.refs.name.value = developer.name;

this.refs.role.value = developer.role;

}

Now your React application is ready to read and set data into your Firebase database. Here is the final demo again:

You might wonder is it safe to put Firebase Api key in the config where experienced programmers can easily get it. Actually, it’s okay to put it there because Firebase has security rules that ensures only authenticated users can access your database. I just haven’t setup the security rules in this tutorial. So don't put the code in production environment just yet.