Ibukun Folay
Posted on 22 abr 2023 • Updated on 2 may 2023

Build a Nodejs Server Using Firebase/Firestore (CRUD)

#node #javascript #webdev #firebase

Introduction

In this article we would be looking at how to build a node js server using firebase and firestore.
We would also be building the crud (Create, Read, Update, Delete) functionalities into our server.

Topics to look out for in this article

- What is node js and firebase?
- Setting up our project folder
- Installing neccessary dependencies
- Es6 setup (.babelrc)

- Configuring firebase
- Setting up config files
- Create express server (index.js)
- Building our model
- Building crud functionalities (controllers)
- Routes
- Run server

What is node js and firebase?

Nodejs is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser. Nodejs is used for building asynchronous & event-driven model backend servers. If you don't have node js installed on your local machine, you can download it here.

Firebase is a Backend-as-a-Service (BaaS) app development platform provided by google that hosts backend services such as a realtime database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files.
The service we would be using today from firebase is Firestore, which is the database provided by firebase. It is a no-sql document oriented database.

Setting up our project folder

Firstly, we have to create a folder, in my case i'll call it node-firebase but feel free to name it whatever. This would be our root working directory.

Then in your terminal, run the following script:

```
Npm init -y
```

This should create a package.json file in the root of our folder.

Next, in the root of our project folder we need to create the following files:

- index.js - this is the main file for our server
- firebase.js - this is where the database connection is initiated
- config.js - this contains all the necessary configuration to help maintain a clean working environment. I refer to this as the brain box file in any server.

- .env - this contains all our environment variables

Also create the following folders in the root of your project folder:

- Controllers
- Models
- Routes

Installing neccessary dependencies

Run the scripts below to install the needed dependencies:

```
npm install express cors dotenv firebase
```

```
npm install nodemon –save-dev
```

Using –save-dev installs our *nodemon* as a dev dependency

Es6 setup (babel)

This will enable us to use the es6 syntax in writing our code.

In our terminal we run the following script to install our babel packages as dev dependencies:

```
npm install @babel/core @babel/node @babel/cli @babel/preset-env –save-dev
```

after the dependencies are installed successfully, create a file .babelrc in the root of your project folder and add the code below:

```
{
"presets": [["@babel/preset-env"]]
}
```

finally, in the package.json, add the line below:

```
"type": "module"
```

And we are good to go using the es6 syntax.

Setting up firebase

After our dependencies have been installed, we go to [firebase console] a follow the steps below:

- Click on add project
- Enter the name of the project
- We can turn off analytics for this project.
- Click create project
- Click on the code button
- Enter the name of the app and click on the register app button
- Click on cloud firestore (to create database)
- Click on setting and copy firebase configuration

Setting up config files

- .env file

After sucessfully executing the steps above, in the root of your project folder, create a .env file and add the following code.

```
PORT=5000
HOST=localhost
HOST_URL=http://localhost:5000
```

```
#firebase config
API_KEY=””
AUTH_DOMAIN=””
PROJECT_ID=””
STORAGE_BUCKET=””
MESSAGING_SENDER_ID=””
APP_ID=””
```

Replace the strings with the firebase config details generated on your firebase console.

Config.js

Once you are done with the above, you can copy the code below into your config.js file in the root of your folder.

```
import dotenv from 'dotenv';
import assert from 'assert';

dotenv.config();

const {
  PORT,
  HOST,
  HOST_URL,
  API_KEY,
  AUTH_DOMAIN,
  PROJECT_ID,
  STORAGE_BUCKET,
  MESSAGING_SENDER_ID,
  APP_ID,
} = process.env;

assert(PORT, 'Port is required');
assert(HOST, 'Host is required');

export default {
  port: PORT,
  host: HOST,
  hostUrl: HOST_URL,
  firebaseConfig: {
    apiKey: API_KEY,
    authDomain: AUTH_DOMAIN,
    projectId: PROJECT_ID,
    storageBucket: STORAGE_BUCKET,
    messagingSenderId: MESSAGING_SENDER_ID,
    appId: APP_ID,
  },
};
```

Firebase.js

In the firebase.js file add the following code to initialize our database:

```
import { initializeApp } from 'firebase/app';
import config from './config.js';

const firebase = initializeApp(config.firebaseConfig);
```

```
export default firebase;
```

Create express server (index.js)

In the index.js file, add the following code:

```
import express from 'express';
import cors from 'cors';

import config from './config.js';

const app = express();

app.use(cors());
app.use(express.json());


app.listen(config.port, () =>
  console.log(`Server is live @ ${config.hostUrl}`),
);
```

If we run `npm start` our server should be up and running.

Building our model

Create a file named productModel.js in your model folder and add the code below:

```
class Product {
  constructor(id, name, price, retailer, amountInStock) {
    (this.id = id),
      (this.name = name),
      (this.price = price),
      (this.retailer = retailer),
      (this.amountInStock = amountInStock);
  }
}

export default Product;
```

Building crud functionalities (controllers)

Crud stands for Create, Read, Update and Delete. These are the foundations on which an api is built.

In the controllers folder, create a file named productControllers.js and add the following blocks of code respectively:

At the top :

```
import firebase from '../firebase.js';
import Product from '../models/productModel.js';
import {
  getFirestore,
  collection,
  doc,
  addDoc,
  getDoc,
  getDocs,
  updateDoc,
  deleteDoc,
} from 'firebase/firestore';

const db = getFirestore(firebase);
```

Create product function:

```
export const createProduct = async (req, res, next) => {
  try {
    const data = req.body;
    await addDoc(collection(db, 'products'), data);
    res.status(200).send('product created successfully');
  } catch (error) {
    res.status(400).send(error.message);
  }
};
```

Get all products function:

```
export const getProducts = async (req, res, next) => {
  try {
    const products = await getDocs(collection(db, 'products'));
    const productArray = [];
```

```
      if (products.empty) {
        res.status(400).send('No Products found');
      } else {
        products.forEach((doc) => {
          const product = new Product(
            doc.id,
            doc.data().name,
            doc.data().price,
            doc.data().retailer,
            doc.data().amountInStock,
          );
          productArray.push(product);
        });

        res.status(200).send(productArray);
      }
    } catch (error) {
      res.status(400).send(error.message);
    }
  };
```

Get product by id:

```
export const getProduct = async (req, res, next) => {
  try {
    const id = req.params.id;
    const product = doc(db, 'products', id);
    const data = await getDoc(product);
    if (data.exists()) {
      res.status(200).send(data.data());
    } else {
      res.status(404).send('product not found');
    }
  } catch (error) {
    res.status(400).send(error.message);
  }
};
```

Update product by id:

```
export const updateProduct = async (req, res, next) => {
  try {
    const id = req.params.id;
    const data = req.body;
```

```
    const product = doc(db, 'products', id);
    await updateDoc(product, data);
    res.status(200).send('product updated successfully');
  } catch (error) {
    res.status(400).send(error.message);
  }
};
```

And finally, the delete product function:

```
export const deleteProduct = async (req, res, next) => {
  try {
    const id = req.params.id;
    await deleteDoc(doc(db, 'products', id));
    res.status(200).send('product deleted successfully');
  } catch (error) {
    res.status(400).send(error.message);
  }
};
```

Routes

In our routes folder, create a file named productRoute.js and code the file below to it:

```
import express from 'express';

import {
  createProduct,
  getProduct,
  getProducts,
  updateProduct,
  deleteProduct,
} from '../controllers/productController.js';

const router = express.Router();

router.get('/', getProducts);
router.post('/new', createProduct);
router.get('/product/:id', getProduct);
router.put('/update/:id', updateProduct);
router.delete('/delete/:id', deleteProduct);
```

```
export default router;
```

In order to get access to this route, we have to update our index.js file:

Import the product route file:

```
import productRoute from './routes/productRoute.js';
```

Initialize the route:

```
app.use('/api', productRoute);
```

Our final index.js should look like this:

```
import express from 'express';
import cors from 'cors';

import config from './config.js';
import productRoute from './routes/productRoute.js';

const app = express();

app.use(cors());
app.use(express.json());

//routes
app.use('/api', productRoute);

app.listen(config.port, () =>
  console.log(`Server is live @ ${config.hostUrl}`),
);
```

Run server

In order to run the our nodejs-firebase server;

We have to add the start command to our scripts in package.json:

```
"start": "nodemon index.js"
```

After this, in our terminal we can run the server command using:

```
npm start
```

You can now use postman to ensure your api's are working fine and also see the created files in your firestore database.

Conclusion

At the end of this article, you should be able to create a working Nodejs server with firebase.

Here's a link to the source code on github.

Happy coding.

Top comments (1)

Mahesh Sharma • 27 mar

this is awesome for learn firebase

Code of Conduct • Report abuse

Ibukun Folay

Software Developer

JOINED
8 abr 2023

More from Ibukun Folay

How to Build a Chrome Extension using React and Tailwindcss
#react #javascript #tailwindcss #webdev