Firebase Storage for Angular

A reliable file upload solution for your applications

Abdelfattah Ragab

Firebase Storage for Angular

A reliable file upload solution for your applications

Abdelfattah Ragab

Introduction

Welcome to the book “Firebase Storage for Angular: A reliable file upload solution for your applications”.
In this book, I explain how you can integrate Firebase Storage into your Angular application.
One of the pain points in many applications is where to store the application files. Whether you want to upload the product images for your e-commerce store or allow users of your social media application to upload files, etc. You need a reliable file storage provider to manage your files.
Fortunately, Firebase Storage comes to the rescue.
By the end of this book, you will be able to manage files freely from your application and handle all kinds of scenarios.
Let us get started.

What is Cloud Storage for Firebase?

Cloud Storage for Firebase is a powerful and cost-effective object storage service designed for managing user-generated content such as images, audio, video and other files. It is based on the Google Cloud infrastructure and offers scalability and reliability for developers working on mobile, web and server applications.

Google account

You should have a google account.

1. Sign in to your account.
2. Open firebase website.
3. Go to the console.

Create a project

1. Start a new firebase project.
2. Give it a name and continue.

Enable Cloud Storage

In the firebase console, navigate to the storage section and click on Get Started.

You need to set up your Cloud Billing Account, if not already done.

Create a Storage Bucket

During setup, you will be prompted to create a cloud storage bucket. Choose a unique name for your bucket, select the appropriate storage class and specify the storage location that best suits your needs.

Configure Security Rules

Check and configure the security rules for the cloud storage. Firebase requires authentication for read and write operations by default. You can customize these rules depending on the requirements of your application, e.g. allow public access during development.

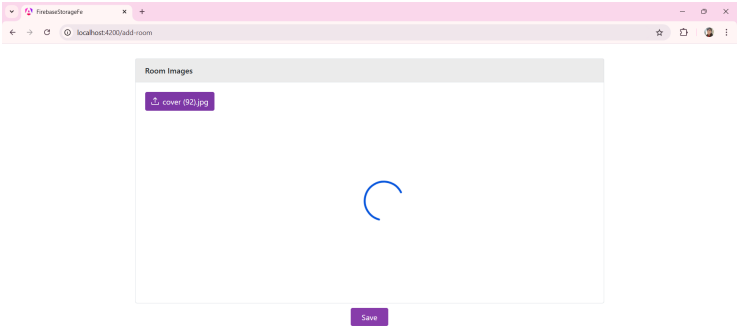
The Backend

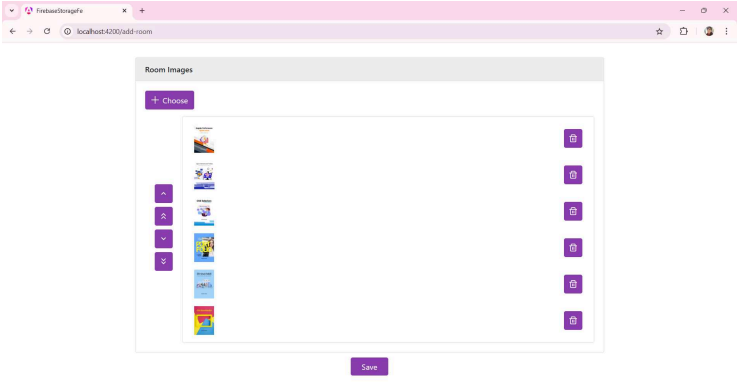
As with all of our major applications, I will set up a backend server in the middle between the Angular application and the Firebase server to protect the business logic from changes and hide our API keys.

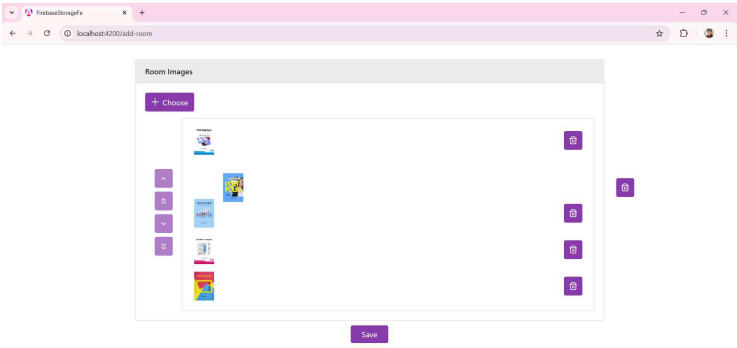
I will create three modules: db, rooms and storage. The upload logic is implemented in the storage module, but I have created two more modules to give you a broad idea of how to use it in real life.

I assume a hotel booking application, and you as the owner want to upload images for the hotel room.
I have also created the db module to serve as a replacement for the database. For the sake of simplicity, let us just push the newly added room into the room array.

Preview

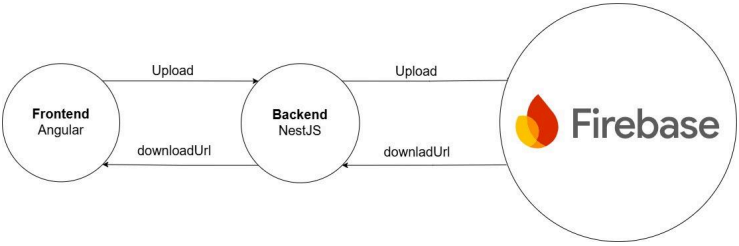






We will build a multi file upload solution and I will use the primeng orderlist component to allow ordering the files.

How does it work!



1. You select files on your computer that will be sent to the NestJS backend server.
2. The NestJS backend server sends the files to the Firebase server.
3. The Firebase server uploads the files and responds with the DownloadUrl.
4. NestJS stores the details of the uploaded files in the database (in our case in the db array). Note

that we store the files in the database after they have been returned from Firebase, as they now have the DownloadUrl required to preview and download the files.

5. The final step is to return the saved file array to the Angular application, which will read the DownloadUrl and display the files in the order list.

Source code

You can find the code example at `https://books.abdelfattah-ragab.com` Download, unzip and run. Don't forget to `npm install`.

Debug Enabled

In the Nest application, you can start troubleshooting by clicking on the "Run" menu“ → “Start Debugging”. This works because I added a **launch.json** file in the **.vscode** folder. It contains the configuration for NestJS debugging. Simply add this file to any of your NestJS applications to enable debugging.

Install Dependencies

We need to install Firebase, uuid and dotenv:.
`npm i firebase uuid dotenv`

I will use `uuid` to generate unique names for uploaded files and `dotenv` to read the keys from the **.env** file.

Install `@types/multer` as a dev dependency.
`npm i @types/multer --save-dev`

.eslintrc.js

Customize the configuration of the **.eslintrc.js** file, otherwise you will get a lot of warnings. Since I don't need it for this application, I will delete its content and replace it with an empty object as follows:
`module.exports = {};`

Firebase Keys

Add the firebase keys to the **.env** file

```
FIREBASE_APIKEY=AIz..
FIREBASE_AUTHDOMAIN=fb-sto..
FIREBASE_PROJECTID=fb-sto..
FIREBASE_STORAGEBUCKET=fb-sto..
FIREBASE_MESSAGINGSENDERID=415..
FIREBASE_APPID=1:4150..
```

DB Module

For simplicity I will create a db module with arrays to save files and room details instead of connecting to the database.
Create a new module for db as follows:

```
nest generate module modules/db
```

DB Service

Create a new service for db as follows:

```
nest generate service modules/db
```

It will have the following arrays:

```
private rooms: Room[] = [];
private files: File[] = [];
private roomFiles: RoomFile[] = [];
```

Similar to the database, we have an array for rooms and another for files, with the roomFiles array taking on the role of the relational tables in the database. It simply defines which files belong to which rooms.

The dbService will have methods to add and delete entries.

Storage Module

Create a new module for storage as follows: