Techiediaries

Author: Techiediaries Team 05 Aug 2020

In this tutorial, you'll be using Angular 10/9 with Firebase and Firestore database to create an app that implements the common CRUD operations.

You can also read how to use your own self-hosted database with Node and TypeORM.

We'll see step by step how to set up Firebase in our Angular 10 project, and create a service for implementing Firebase CRUD operations using the Firestore realtime database.

Angular 10 CRUD with Firebase and Firestore Database

These are the steps of our Angular 10 Firebase CRUD tutorial:

Step 1 - Creating your Angular 10 Project
Step 2 - Creating a Firebase Project and a Firestore Database
Step 3 - Installing and Adding Firebase to your Angular 10 Project.
Step 4 - Create an Angular 10 Model
Step 5 - Creating an Angular 10 Service
Step 6 - Creating a Component for Making CRUD Operations

CRUD stands for Create, Read, Update and Delete and refer to the operations that we run against a database to create, retrieve, update and delete data. In this example, the database is a Firestore database that exists on the cloud.

> ***Note**: This tutorial works with Angular 9.*

Prerequisites

Before starting this tutorial, you first need to make sure, you have:

If you have the prerequisites, you are ready to start creating your project!

Step 1 - Creating your Angular 10 Project

The first step in this tutorial is creating a new Angular 10 project using the CLI.

Head over to your terminal and run the following command:

```
$ ng new angular-firebase-crud
```

The CLI will ask if you want to add routing to your project (you can choose any option you want) and which style sheet format you want to use (You can select CSS).

After that, your project files will be generated and your project's dependencies will be installed.

Step 2 - Creating a Firebase Project and a Firestore Database

Now that your project is generated, you need to proceed with creating a Firebase
need to go to the Firebase Console and create a new Firebase project.

Next head to the `Project Overview >Develop >Database` page and create a new Fi
so you don't need any permissions to access the database.

Step 3 - Installing and Adding Fi[illegible] your Angular

re database you next need to install the `firebase` and `@angular/fire` packages as follows:

on *web* and copy the *config* data.

/our Angular 10 project and add the `firebaseConfig` object inside the `environment` object.

```
export const environment = {
  production: false,
  firebaseConfig : {
    apiKey: "YOUR_API_KEY",
    authDomain: "YOUR_AUTH_DOMAIN",
    databaseURL: "YOUR_DATABASE_URL",
    projectId: "YOUR_PROJECT_ID",
    storageBucket: "YOUR_STORAGE_BUCKET",
    messagingSenderId: "YOUR_MESSAGING_SENDER_ID"
  }
```

```
import { AngularFireModule } from '@angular/fire';
import { AngularFireDatabaseModule } from '@angular/fire/database';
import { environment } from '../environments/environment';

@NgModule({
        // [...]
    imports: [
        // [...]
        AngularFireModule.initializeApp(environment.firebaseConfig),
        AngularFireDatabaseModule
    ],
```

You simply import `AngularFireModule` and `AngularFireDatabaseModule` and you add them to the `imports` array of the main application module.

You also call the `initializeApp()` method of `AngularFireModule` to pass the configuration object that you added earlier to the `environments/environment.ts` file.

That's it, you now have added Firebase and Firestore to your Angular 10 project.

Step 4 - Create an Angular 10 Model

After setting up Firestore in your project, you can proceed with creating a model creating an insurance app where we need to manage a set of policies.

An insurance application will often contain more that one type of data like clients focus on the policy entity.

Let's create a model for our insurance policy entity as follows:

```
$ ng g class policy --type=model
```

t as follows:

This is an example of an insurance policy with many fields and relationships with other entities omitted for the sake of simplicity.

Step 5 - Creating an Angular 10 Service

An Angular service allows you to encapsulate the code that could be repeated in many places in your project. Using the Angular CLI, run the following command to generate a service:

```
$ ng g service policy
```

```
import { AngularFirestore } from '@angular/fire/firestore';
import { Policy } from 'src/app/policy.model';
```

Next, inject `AngularFirestore` in your service via its constructor:

```
export class PolicyService {
  constructor(private firestore: AngularFirestore) { }
}
```

Next, add the `getPolicies()` method to retrieve the available policies from the Firestore collection:

```
getPolicies() {
    return this.firestore.collection('policies').snapshotChanges();
}
```

You also need to add the `createPolicy()` method to persist an insurance policy in the Firestore database:

```
createPolicy(policy: Policy){
    return this.firestore.collection('policies').add(policy);
}
```

Next, you need to add the `updatePolicy()` method to update an insurance policy b

```
updatePolicy(policy: Policy){
    delete policy.id;
    this.firestore.doc('policies/' + policy.id).update(policy);
}
```

Finally, you can add the `deletePolicy()` method to delete an insurance policy by

```
.delete();
```

:omponent for Making CRUD

ding, updating and deleting insurance policies, you now need to create the component

Using Angular CLI 10 run the following command to generate a component:

```
$ ng g c policy-list
```

Now, open the `src/app/policy-list/policy-list.component.ts` file and update it accordingly:

```
import { Component, OnInit } from '@angular/core';
import { PolicyService } from 'src/app/policy.service';
import { Policy } from 'src/app/policy.model';
```

```
export class PolicyListComponent implements OnInit {

  policies: Policy[];
  constructor(private policyService: PolicyService) { }

  ngOnInit() {
    this.policyService.getPolicies().subscribe(data => {
      this.policies = data.map(e => {
        return {
          id: e.payload.doc.id,
          ...e.payload.doc.data()
        } as Policy;
      })
    });
  }

  create(policy: Policy){
      this.policyService.createPolicy(policy);
  }

  update(policy: Policy) {
    this.policyService.updatePolicy(policy);
  }

  delete(id: string) {
    this.policyService.deletePolicy(id);
  }
}
```

Updating the Component Template

Now let's update the component's template to display the insurance policies and a
delete policies:

Open the `src/app/policy-list.component.html` file and add the following HTMI

```
      <td>{{policy.expireDate}}</td>
      <td>{{policy.policyAmount}}</td>
      <td>
          <button (click)="delete(policy.id)">Delete</button>
      </td>
    </tr>
  </tbody>
</table>
```

Below the `<table>` markup, you can also add a form to create an insurance policy.

Conclusion

ure of Angular

3+ Ways to Add Bootstrap 4 to Angular 10/9 With Example & Tutorial
Angular 11 CRUD REST API
Angular 11 Tutorial with HttpClient
Routing and Navigation with Angular 11 Router
Full Vue 3 Tutorial by Example
CSS Centering
CSS Grid Layout Tutorial
Adding Comments to JSON
How to Delete Local/Remote Git Branches
Webpack Tutorial for Angular Developers
GraphQL Tutorial for Angular Developers
Git and GitHub for Angular Developers
Bootstrap 5 with Sass and Gulp 4 Tutorial by Example

Angular Newsletter

Get our Angular book for free!