How to build an Angular 10 app from scratch in 11 easy steps (eBook by Techiediaries.com)

Angular is one of the three most popular frameworks for front-end development, alongside React and Vue.js. The latest version is Angular 10 at the time of this writing.

There are many new features and enhancements to both the command-line interface and the framework itself which result in a performance-boost and smaller production bundles.

In this tutorial, we'll take you step by step on a journey to build an example Angular 10 application from scratch that uses many Angular APIs such as HttpClient, and Material Design.

Here are a few things we'll learn:

- How to mock a REST API server that uses fake data from a JSON file
- How to consume the REST API from our Angular 10 application using `Httplient`
- How to handle HTTP errors using the RxJS `throwError()` and `catchError()` operators
- How to retry failed HTTP requests in poor network conditions and cancel pending requests using the RxJS `retry()` and `takeUntil()` operators
- How to create and make use of Angular components and services
- How to set up routing and Angular Material in our project and create a professional-looking UI with Material Design components
- And finally, we'll learn how to deploy the application to Firebase using the `ng deploy` command available in Angular 8.3+.

You'll also learn by example:

- How to quickly mock a REST API with real-world features, such as pagination, that you can consume from your app before you can switch to a real backend when it's ready.
- How to set up Angular CLI
- How to initialize your Angular 10 project
- How to set up Angular Material
- How to add Angular components and routing
- How to generate and use Angular services
- How to consume REST APIs with Angular HttpClient
- How to build and deploy your Angular application to Firebase.

This tutorial is divided into the following steps:

- Step 1 — Installing Angular CLI 10
- Step 2 — Creating your Angular 10 Project
- Step 3 — Adding Angular HttpClient
- Step 4 — Creating Components
- Step 5 — Adding Routing

- Step 6 — Building the UI with Angular Material Components
- Step 7 — Mocking a REST API
- Step 8 — Consuming the REST API with Angular HttpClient
- Step 9 — Handling HTTP Errors
- Step 10 — Adding Pagination
- Step 11 — Building and Deploying your Angular Application to Firebase

Now, let's get started with the prerequisites!

> **Note**: you can download our **Angular Book: Build your first web apps with Angular** for free.

Prerequisites

If you want to follow this tutorial, you'll need to have:

- Prior knowledge of TypeScript.
- A development machine with **Node 8.9+** and **NPM 5.5.1+** installed. Node is required by the Angular CLI. You can head to the official website and grab the binaries for your system. You can also use NVM — Node Version Manager — a POSIX-compliant bash script to install and manage multiple Node.js versions in your machine.

If you are ready, let's learn by example how to build an Angular 10 application that consumes a REST API using HttpClient. We'll implement real-world features like error handling and retrying failed HTTP requests.

Step 1 — Installing Angular CLI 10

Let's start with the first step, where we'll install the latest version of Angular CLI.

Angular CLI is the official tool for initializing and working with Angular projects. Head to a new terminal and execute the following command:

```
$ npm install -g @angular/cli
```

When writing this tutorial, **angular/cli v8.3.2** is installed on our system.

That's it, you are ready for the second step!

Step 2 — Creating your Angular 10 Project

In this step, we'll use Angular CLI to initialize our Angular project.

Go to your terminal and execute these commands:

```
$ cd ~
$ ng new angular-example
```

The CLI will prompt you and ask **whether you would like to add Angular routing.** Say Yes. It'll then ask **which stylesheet format you would like to use.** Choose **CSS**.

Angular CLI will generate the required files and folders, install the packages from npm, and even automatically set up routing in our project.

Now, go to your project's root folder and run the local development server using these commands:

```
$ cd angular-example
$ ng serve
```

Your Angular web application will be available from the `[http://localhost:4200/]` `(http://localhost:4200/)` address.

Open a web browser and go to the `http://localhost:4200/` address. You should see this beautiful page (Starting with Angular 8.3+):

You need to leave the development server running and open a new terminal for the next steps.

You are now ready for the third step!

Step 3 — Adding Angular HttpClient

In this step, we'll add `HttpClient` to our example project.

Open the `src/app/app.module.ts` file and make the following changes:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

We simply imported HttpClientModule and included it in the `imports` array.

That's all - now we can use the `HttpClient` service in our Angular project to consume our REST API.

You are ready for the fifth step!

Step 4 — Creating UI Components

Angular apps are made up of components. In this step, we'll learn how to create a couple of Angular components that compose our UI.

Open a new terminal and run the following command:

```
$ cd ~/angular-example
$ ng g component home
```

You'll get the following output in your terminal:

```
CREATE src/app/home/home.component.html (19 bytes)
CREATE src/app/home/home.component.spec.ts (614 bytes)
CREATE src/app/home/home.component.ts (261 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (467 bytes)
```

We have four files, all required by our component.

Next, generate the about component:

```
$ ng g component about
```

Next, open the `src/app/about/about.component.html` file and add the following code:

```
<p style="padding: 15px;"> This is the about page that describes your app</p>
```

You are ready for the sixth step!

Step 5 — Adding Routing

In this step, we'll learn how to add routing to our example.

Go to the `src/app/app-routing.module.ts` file and add the following routes:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';


const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full'},
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

We imported the Angular components and we declared three routes.

The first route is for redirecting the empty path to the home component, so we'll be automatically redirected to the home page when we first visit the app.

That's it. Now that you have added routing, you are ready for the next step!

Step 6 — Adding Angular Material

In this tutorial step, we'll learn to set up Angular Material in our project and build our application UI using Material components.

Go to your terminal and run this command from the root of your project:

```
$ ng add @angular/material
```

You'll be prompted to choose the theme, so let's pick **Indigo/Pink**.

For the other questions - whether you want to **set up HammerJS for gesture recognition** and if you want to **set up browser animations for Angular Material** - press **Enter** to use the default answers.

Open the `src/app/app.module.ts` file and add the following imports:

```
import { MatToolbarModule,
  MatIconModule,
  MatCardModule,
  MatButtonModule,
  MatProgressSpinnerModule } from '@angular/material';
```

We imported the modules for these Material Design components:

- MatToolbar that provides a container for headers, titles, or actions.
- MatCard that provides a content container for text, photos, and actions in the context of a single subject.
- MatButton that provides a native `<button>` or `<a>` element enhanced with Material Design styling and ink ripples.
- MatProgressSpinner that provides a circular indicator of progress and activity.

Next, add these modules to the `imports` array:

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    AboutComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    BrowserAnimationsModule,
    MatToolbarModule,
    MatIconModule,
    MatButtonModule,
    MatCardModule,
    MatProgressSpinnerModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Next, open the `src/app/app.component.html` file and update it as follows:

```
<mat-toolbar color="primary">
<h1>
My Angular Store
</h1>
<button mat-button routerLink="/">Home</button>
<button mat-button routerLink="/about">About</button></mat-toolbar><router-outlet></router-out
```

We added a top navigation bar with two buttons that take us to the home and about pages, respectively.

Step 7 — Mocking a REST API

Go to a new command-line interface and start by installing `json-server` from npm in your project:

```
$ cd ~/angular-example
$ npm install --save json-server
```

Next, create a `server` folder in the root folder of your Angular project:

```
$ mkdir server
$ cd server
```

In the `server` folder, create a `database.json` file and add the following JSON object:

```
{
    "products": []
}
```

This JSON file will act as a database for your REST API server. You can simply add some data to be served by your REST API or use Faker.js for automatically generating massive amounts of realistic fake data.

Go back to your command-line, navigate back from the `server` folder, and install `Faker.js` from npm using the following command:

```
$ cd ..
$ npm install faker --save
```

At the time of creating this example, **faker v4.1.0** will be installed.

Now, create a `generate.js` file and add the following code:

```
var faker = require('faker');

var database = { products: []};

for (var i = 1; i<= 300; i++) {
  database.products.push({
    id: i,
    name: faker.commerce.productName(),
    description: faker.lorem.sentences(),
    price: faker.commerce.price(),
    imageUrl: "https://source.unsplash.com/1600x900/?product",
    quantity: faker.random.number()
  });
}

console.log(JSON.stringify(database));
```

We first imported faker, and next we defined an object with one empty array for products. Next, we entered a *for* loop to create *300* fake entries using faker methods like `faker.commerce.productName()` for

generating product names. Check all the available methods. Finally we converted the database object to a string and logged it to standard output.

Next, add the `generate` and `server` scripts to the `package.json` file:

```
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "generate": "node ./server/generate.js > ./server/database.json",
    "server": "json-server --watch ./server/database.json"
  },
```

Next, head back to your command-line interface and run the generate script using the following command:

```
$ npm run generate
```

Finally, run the REST API server by executing the following command:

```
$ npm run server
```

You can now send HTTP requests to the server just like any typical REST API server. Your server will be available from the `http://localhost:3000/` address.

These are the API endpoints we'll be able to use via our JSON REST API server:

- `GET /products` for getting the products
- `GET /products/<id>` for getting a single product by id
- `POST /products` for creating a new product
- `PUT /products/<id>` for updating a product by id
- `PATCH /products/<id>` for partially updating a product by id
- `DELETE /products/<id>` for deleting a product by id

You can use `_page` and `_limit` parameters to get paginated data. In the `Link` header you'll get `first`, `prev`, `next` and `last` links.

Leave the JSON REST API server running and open a new command-line interface for typing the commands of the next steps.

Step 8 — Creating a Service for Consuming the REST API with Angular HttpClient

In this step, we'll learn how to consume our REST API from Angular using HttpClient.

We'll need to create an Angular service for encapsulating the code that allows us to consume data from our REST API server.

Go to your terminal and run the following command:

```
$ ng g service api
```

Next, go to the `src/app/api.service.ts` file, import and inject `HttpClient`:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  private SERVER_URL = "http://localhost:3000";

  constructor(private httpClient: HttpClient) { }
}
```

We imported and injected the `HttpClient` service and defined the `SERVER_URL` variable that contains the address of our REST API server.

Next, define a `get()` method that sends a GET request to the REST API endpoint:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
 providedIn: 'root'
})
export class ApiService {

 private SERVER_URL = "http://localhost:3000";
 constructor(private httpClient: HttpClient) { }

 public get(){
  return this.httpClient.get(this.SERVER_URL);
 }
}
```

The method simply invokes the `get()` method of `HttpClient` to send GET requests to the REST API server.

Next, we now need to use this service in our home component. Open the `src/app/home/home.component.ts` file, and import and inject the data service as follows:

```
import { Component, OnInit } from '@angular/core';
import { ApiService } from '../api.service';

@Component({
 selector: 'app-home',
 templateUrl: './home.component.html',
 styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
 products = [];
 constructor(private apiService: ApiService) { }
 ngOnInit() {
  this.apiService.get().subscribe((data: any[])=>{
   console.log(data);
   this.products = data;
  })
 }
}
```

We imported and injected `ApiService.` Next, we defined a `products` variable and called the `get()` method of the service for fetching data from the JSON REST API server.

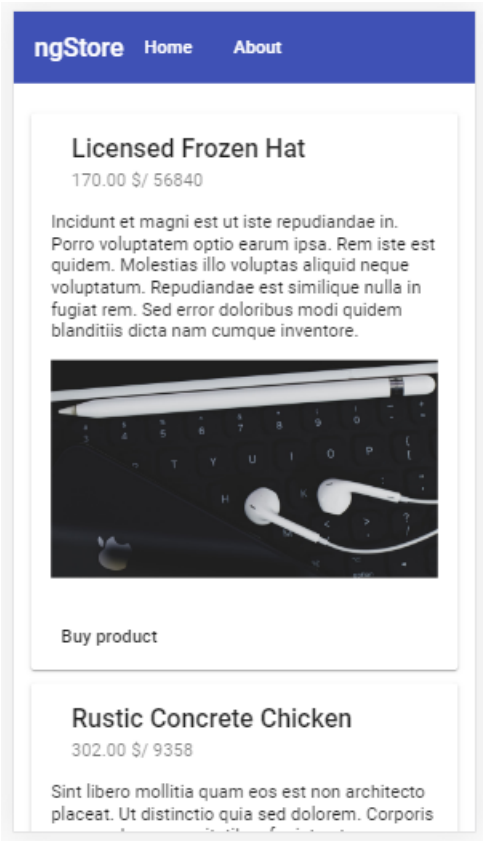
Next, open the `src/app/home/home.component.html` file and update it as follows:

```
<div style="padding: 13px;">
    <mat-spinner *ngIf="products.length === 0"></mat-spinner>

    <mat-card *ngFor="let product of products" style="margin-top:10px;">
        <mat-card-header>
            <mat-card-title>{{product.name}}</mat-card-title>
            <mat-card-subtitle>{{product.price}} $/ {{product.quantity}}
            </mat-card-subtitle>
        </mat-card-header>
        <mat-card-content>
            <p>
                {{product.description}}
            </p>
            <img style="height:100%; width: 100%;" src="{{ product.imageUrl }}" />
        </mat-card-content>
        <mat-card-actions>
      <button mat-button> Buy product</button>
    </mat-card-actions>
    </mat-card>
</div>
```

We used the `<mat-spinner>` component for showing a loading spinner when the length of the `products` array equals zero, that is before any data is received from the REST API server.

Next, we iterated over the `products` array and used a Material card to display the `name`, `price`, `quantity`, `description` and `image` of each product.

This is a screenshot of the home page after JSON data is fetched:



Next, we'll see how to add error handling to our service.

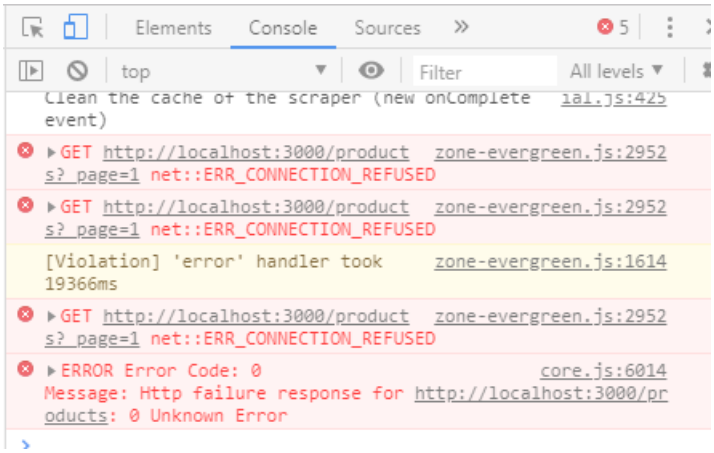
Step 9 — Adding Error Handling

In this step, we'll learn to add error handling in our example.

Go to the `src/app/api.service.ts` file and update it as follows:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from "@angular/common/http";

import {  throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';


@Injectable({
  providedIn: 'root'
})
export class ApiService {

  private SERVER_URL = "http://localhost:3000/products";

  constructor(private httpClient: HttpClient) { }

  handleError(error: HttpErrorResponse) {
    let errorMessage = 'Unknown error!';
    if (error.error instanceof ErrorEvent) {
      // Client-side errors
      errorMessage = `Error: ${error.error.message}`;
    } else {
      // Server-side errors
      errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;
    }
    window.alert(errorMessage);
    return throwError(errorMessage);
  }

  public sendGetRequest(){
    return this.httpClient.get(this.SERVER_URL).pipe(catchError(this.handleError));
  }
}
```

This is a screenshot of an example error on the browser console:



In the next step, we'll see how to add pagination to our application

Step 10 — Adding Pagination

In this step, we'll learn to add support for data pagination using the Link header of the HTTP response received from the REST API server.

By default, HttpClient provides only the response body. But in our app we need to parse the Link header for extracting pagination links. So we need to instruct `HttpClient` to give us the full HttpResponse using the `observe` option.

The Link header in HTTP allows the server to point an interested client to another resource containing metadata about the requested resource. Wikipedia

Open the `src/app/data.service.ts` file and import the RxJS `tap()` operator:

```
import { retry, catchError, tap } from 'rxjs/operators';
```

Next, add these variables:

```
public first: string = "";
public prev: string = "";
public next: string = "";
public last: string = "";
```

Next, add the `parseLinkHeader()` method that will be used to parse the Link header and populates the previous variables:

```
parseLinkHeader(header) {
  if (header.length == 0) {
    return ;
  }

  let parts = header.split(',');
  var links = {};
  parts.forEach( p => {
    let section = p.split(';');
    var url = section[0].replace(/<(.*)>/, '$1').trim();
    var name = section[1].replace(/rel="(.*)"/, '$1').trim();
    links[name] = url;

  });

  this.first  = links["first"];
  this.last   = links["last"];
  this.prev   = links["prev"];
  this.next   = links["next"];
}
```

Next, update the `sendGetRequest()` as follows:

```
public sendGetRequest(){
  // Add safe, URL encoded _page and _limit parameters

  return this.httpClient.get(this.SERVER_URL, {  params: new HttpParams({fromString: "_page=
    console.log(res.headers.get('Link'));
    this.parseLinkHeader(res.headers.get('Link'));
  }));
}
```

We added the `observe` option with the `response` value in the options parameter of the `get()` method so we can have the full HTTP response with headers. Next, we use the RxJS `tap()` operator for parsing the Link header before returning the final Observable.

Since the `sendGetRequest()` is now returning an Observable with a full HTTP response, we need to update the home component so open the `src/app/home/home.component.ts` file and import `HttpResponse` as follows:

```
import { HttpResponse } from '@angular/common/http';
```

Next, update the `subscribe()` method as follows:

```
ngOnInit(){

this.apiService.sendGetRequest().pipe(takeUntil(this.destroy$)).subscribe((res: HttpResponse<a
 console.log(res);
 this.products = res.body;
})
}
```

We can now access the data from the `body` object of the received HTTP response.

Next, go back to the src/app/data.service.ts file and add the following method:

```
public sendGetRequestToUrl(url: string){
 return this.httpClient.get(url, { observe: "response"}).pipe(retry(3),
 catchError(this.handleError), tap(res => {
  console.log(res.headers.get('Link'));
  this.parseLinkHeader(res.headers.get('Link'));
 }));
}
```

This method is similar to `sendGetRequest()` except that it takes the URL to which we need to send an HTTP GET request.

Go back to the `src/app/home/home.component.ts` file and add define the following methods:

```
public firstPage() {
   this.products = [];
   this.apiService.sendGetRequestToUrl(this.apiService.first).pipe(takeUntil(this.destroy$)).
     console.log(res);
     this.products = res.body;
   })
 }
 public previousPage() {

   if (this.apiService.prev !== undefined && this.apiService.prev !== '') {
     this.products = [];
     this.apiService.sendGetRequestToUrl(this.apiService.prev).pipe(takeUntil(this.destroy$))
       console.log(res);
       this.products = res.body;
     })
   }

 }
 public nextPage() {
   if (this.apiService.next !== undefined && this.apiService.next !== '') {
     this.products = [];
     this.apiService.sendGetRequestToUrl(this.apiService.next).pipe(takeUntil(this.destroy$))
       console.log(res);
       this.products = res.body;
     })
   }
 }
 public lastPage() {
   this.products = [];
   this.apiService.sendGetRequestToUrl(this.apiService.last).pipe(takeUntil(this.destroy$)).s
     console.log(res);
     this.products = res.body;
   })
 }
```

Finally, open the `src/app/home/home.component.html` file and update the template as follows:

```
<div style="padding: 13px;">
    <mat-spinner *ngIf="products.length === 0"></mat-spinner>

    <mat-card *ngFor="let product of products" style="margin-top:10px;">
        <mat-card-header>
            <mat-card-title>#{{product.id}} {{product.name}}</mat-card-title>
            <mat-card-subtitle>{{product.price}} $/ {{product.quantity}}
            </mat-card-subtitle>
        </mat-card-header>
        <mat-card-content>
            <p>
                {{product.description}}
            </p>
            <img style="height:100%; width: 100%;" src="{{ product.imageUrl }}" />
        </mat-card-content>
        <mat-card-actions>
      <button mat-button> Buy product</button>
    </mat-card-actions>
    </mat-card>

</div>
<div>
    <button (click) ="firstPage()" mat-button> First</button>
    <button (click) ="previousPage()" mat-button> Previous</button>
    <button (click) ="nextPage()" mat-button> Next</button>
    <button (click) ="lastPage()" mat-button> Last</button>
</div>
```

This is a screenshot of our application:

Step 11 — Building and Deploying your Angular Application to Firebase

Head back to your command-line interface. Make sure you are inside the root folder of your Angular project and run the following command:

```
$ ng add @angular/fire
```

This will add the Firebase deployment capability to your project.

As the time of writing this tutorial, **@angular/fire v5.2.1** will be installed.

The command will also update the `package.json` of our project by adding this section:

```
    "deploy": {
      "builder": "@angular/fire:deploy",
      "options": {}
    }
```

The CLI will prompt you to **Paste authorization code here:** and will open your default web browser and ask you to give Firebase CLI permissions to administer your Firebase account.

After you sign in with the Google account associated with your Firebase account, you'll be given the authorization code.

Next, you'll be prompted to **Please select a project: (Use arrow keys or type to search)**. You should have created a Firebase project before.

The CLI will create the `firebase.json` and `.firebaserc` files and update the `angular.json` file accordingly.

Next, deploy your application to Firebase using the following command:

```
$ ng deploy
```

The command will produce an optimized build of your application (equivalent to the `ng deploy --prod` command). It will upload the production assets to Firebase hosting.

Make sure to visit us at **Techiediaries** for more tutorials and PDF ebooks about modern web development.

Conclusion

Throughout this step by step tutorial, you learned to build an Angular application from scratch using the latest Angular 8.3+ version.

You learned to mock a REST API backend for your Angular application with nearly zero lines of code.

You learned how to create a project using Angular CLI, add `HttpClient` and Angular Material for sending HTTP requests to your mocked REST API backend, and style the UI with Material Design components.

Finally, you learned to deploy your Angular application to Firebase using the `ng deploy` command available starting from Angular 8.3+.

Check out our other tutorials from https://www.techiediaries.com/angular/.