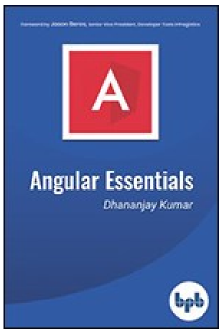


Angular Essentials: The Essential Guide to Learn Angular

by Dhananjay Kumar

BPB Publications. (c) 2019. Copying Prohibited.

Chapter 12: Working with API and $http

Angular provides $http service to work with REST API in an Angular application. You can use $http service with RxJS operators to perform various tasks with the API. You can also use $http services to execute various HTTP request from Angular application to an API to perform CRUD operations on the data residing on the server. In this chapter, you will learn:

- Angular in-memory web api
- Setting up Angular in-memory web api
- Create service to perform HTTP operations
- Read Data
- Create Data
- Update Data
- Delete Data

Angular in-memory Web Api

In real application, you will have data residing on the server, and an Angular application will perform the CRUD operations on the data of server through API. Standard is, you perform various HTTP operations to execute CRUD operations. To read data from the server you perform HTTP GET operation and to create data you perform HTTP POST operation. You perform various operations as shown in figure 12.1

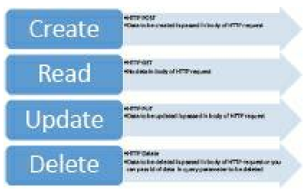


Figure 12.1

In real application, you will have real server and API. However, for development and testing purpose you can fake a backend server. There are three ways; you can fake a backend server in Angular.

1. Return hardcode data from a local file
2. Use a local JSON file
3. Use angular in-memory web api

Problem with hardcoded data is you do not have to make http request to read data from a text file. With local JSON file, you can make http request but again JSON file is not best suited for all CRUD operations. To overcome these issues, Angular comes up with in-memory web api to simulate a data server. You should use Angular in-memory web api for demos and tests that emulates CRUD operations over a REST API.

Angular in-memory web api is very useful for development and testing purpose. Once you have installed it in your application, it intercepts HTTP request and returns mock data. You do not have to hardcode data or need ready backend API for development or testing purposes. You can almost do everything with in-memory web API like backend API such as:

- Setting request header
- Setting response header
- Setting response type

Angular in-memory web API intercepts Angular `Http` and `HttpClient` requests that would otherwise go to the remote API and redirects them to an in-memory data store that you have created. There are various use cases when you should use Angular in-memory web api such as:

- To mock Server API for demo Apps without having a real server. You can mock persistence of Data of CRUD operation locally.
- To simulate operations against data, yet to be created on server.
- To unit test of services doing read and write of data using http.
- To perform end-to-end test without affecting real database at the server.
- To use in continuous integration (CI) builds in which you need to mock real database server.

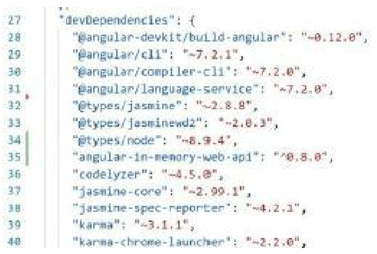
Setting up Angular In-Memory Web Api

You can install Angular in-memory web api in your project using `npm` as shown in *code listing 12.1*.

Code Listing 12.1

npm install angular-in-memory-web-api --save-dev

After successful installation, you can see dependency added for Angular in-memory web api in `package.json` file line number 35 as shown in *figure 12.2*.



```
27    "devDependencies": {
28      "@angular-devkit/build-angular": "~0.12.0",
29      "@angular/cli": "~7.2.1",
30      "@angular/compiler-cli": "~7.2.0",
31      "@angular/language-service": "~7.2.0",
32      "@types/jasmine": "~2.8.8",
33      "@types/jasminewd2": "~2.0.3",
34      "@types/node": "~8.9.4",
35      "angular-in-memory-web-api": "^0.8.0",
36      "codelyzer": "~4.5.0",
37      "jasmine-core": "~2.99.1",
38      "jasmine-spec-reporter": "~4.2.1",
39      "karma": "~3.1.1",
40      "karma-chrome-launcher": "~2.2.0",
```

Figure 12.2

We have added Angular in-memory web api dependencies in the project, now to simulate CRUDoperations, create an entity class for Car asshown in *code listing12.2.*

Code Listing 12.2

```
export class Car {
     constructor ( public id = 0,
                  public model = '',
                  public price= 0,
                  public color = '') {
     }
}
```

Once entity class is created, you need to override `createDb()` method of `inMemoryDbService` abstract class, to do that, create a new class and implement `InMemoryDbService` class as shown in the *code listing 12.3.*

Code Listing 12.3

```
import {InMemoryDbService} from 'angular-in-memory-
web-api';
import { Car } from './car';

export class CarInMemDataService implements
InMemoryDbService {
      createDb() {
           const cars: Car[] = [
            { id: 1, model: 'BMW', price: 40000, color:
'Red'  },
            { id: 2, model: 'Audi', price: 45000, color:
'Blue'  },
            { id: 3, model: 'Tata', price: 20000, color:
'White'  },
            { id: 4, model: 'Honda', price: 30000, color:
'Black' },
            { id: 5, model: 'GM', price: 30000, color:
```

```
'Red'
}
            ];
              return { cars};
      }
}
```

As you see, we are overriding created method, which returns collections. In this case, there is only one-collection car returns; however, you can return any number of collections. To use in-memory web api, you need to import in the required module. You can import it in `AppModule` as shown in *code listing 12.4.*

Code Listing 12.4

```
import { InMemoryWebApiModule } from 'angular-in-
memory-web-api';
import { CarInMemDataService } from './car-in-mem-data.
service';

@NgModule({
     imports: [
        BrowserModule,
        InMemoryWebApiModule.forRoot(CarInMemDataService)
     ],
```

We are passing `CarInMemDataService` class in which created method is implemented to `InMemoryWebApiModule`.

So far, we have added dependency of Angular in-memory web api, created entity class, and implemented created method. Now Angular in-memory web api is ready to be used for CRUD operations. You can access created API at URL - ... **/api/cars.** Various HTTP operations can be performed as shown in the following code:

```
http.post(api/cars, undefined);
http.get(api/cars);
http.post('commands/cars,   '{"delay":1000}');
```

Now we have Angular in-memory web api to perform CRUD operations on Car data. In next section, we will use to perform operations.

Create Service to Perform HTTP Operations

Let us create an Angular service, in which we will perform HTTP operations. We will inject this service in the required components. You can add service in the project using CLI command:

***`ng g`* Service App**

You will have `AppService` created after successful running of command. In created service import items as shown in *code listing 12.5* to perform HTTP operations:

Code Listing 12.5

```
import { HttpClient, HttpHeaders, HttpParams } from '@
angular/common/http';
import { Observable, throwError } from 'rxjs';
import { tap, catchError, map } from 'rxjs/operators';
import { Car } from './car';
```

After importing required files, we need to inject `HttpClient` service in `AppService` as shown in *code listing 12.6.*

Code Listing 12.6

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

     apiurl = 'api/cars';
     headers = new HttpHeaders().set('Content-Type',
'application/json').set('Accept', 'application/json');
```

```
      httpOptions = {
         headers: this.headers
      };

      constructor(private http: HttpClient) {

      }
      private handleError(error: any) {
         console.error(error);
         return throwError(error);
      }

      getCars(): Observable<Car[]> {
         return this.http.get<Car[]>(this.apiurl).pipe(
            tap(data => console.log(data)),
            catchError(this.handleError)
         );
      }
}
```

We have also added **`handleError`** method to log any error encounters while making HTTP calls. In real applications, you should make sure to log error to the server. In addition, we have created a **`httpOptions`** variable to be used while making http operations.

Next, you need to import **`HttpClientModule`** in application module to work with **`HttpClient`** service. You can import **`HttpClientModule`** with other required module as shown in *code listing 12.7.*

Code Listing 12.7

```
import { InMemoryWebApiModule } from 'angular-in-
memory-web-api';
import { CarInMemDataService } from './car-in-mem-data.
service';
import { HttpClientModule } from '@angular/common/
http';

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

As of now, we have created service class and injected **`HttpClientModule`** in that to perform HTTP operations.

Read Data

To read data from API, you need to use **`http.get`**O method. In the method, pass **apiurl** as shown in *code listing 12.8.*

Code Listing 12.8

```
   getCars(): Observable<Car[]> {
      return this.http.get<Car[]>(this.apiurl).pipe(
         tap(data => console.log(data)),
         catchError(this.handleError)
      );
   }
```

We are using pipe and tap operators to read data from observable. If there is any error in HTTP get method request, Angular will call **`catchError`** function.

You can use **`getCars`** method in a component as shown in *code listing 12.9.*

Code Listing 12.9

```
export class AppComponent implements OnInit { cars:

      Car[]  =   [];
      constructor(private appservice: AppService)   {
   }
   ngOnInit() {
      this.getCars();
   }

   getCars()   {
      this.appservice.getCars().subscribe(data => {
         this.cars = data;
         });
   }
}
```

Let us talk though code:

- We are injecting AppService in AppComponent
- Calling getCars method and subscribing to returned observable
- Assigning returned data to local variable cars
- Calling the getCars method in onInit life cycle hook

On the template, you can simply use cars array that holds data from Angular in-memory web api to display data as shown in *code listing 12.10.*

Code Listing 12.10

```
<table>
   <tr *ngFor="let c of cars">
      <td>
         {{cid}}
      </td>
      <td>
         {{c.model}}
      </td>
      <td>
         {{c.color}}
      </td>
      <td>
         {{c.price}}
      </td>
   </tr>
</table>
```

You can read a particular car with its id property. You can read data with ‘id' as shown in *code listing 12.11.* Add method **`getCar`** in **`AppService.`**

Code Listing 12.11

```
getCar(id:  number):  Observable<Car>  {
      const url =  "${this.apiurl}/${id}";
      return this.http.get<Car>(url).pipe(
         catchError(this.handleError)
      );
   }
```

You can use **getCar** method in component as shown in *code listing 12.12.*

Code Listing 12.12

```
idtofetch = 1;
   getCar() {
```

```
            this.appservice.getCar(this.idtofetch).
subscribe(data => {
         this.car = data;
      });
   }
```

We are subscribing to returned observable from service and assigning returned data to local variable car. Also while calling `getCar` method of service, you need to pass id to be fetched. On the template, user can enter ID and fetched result can be displayed as shown in *code listing 12.13.*

Code Listing 12.13

```
<input type="number" [(ngModel)]='idtofetch'
placeholder="Enter Id" />
<button (click)='getCar()'>Fetch Car </button>
<div>
     <h3>Fetched Car with Id : {{idtofetch}}</h3>
     {{car.id}}
     {{car.model}}
     {{car.price}}
     {{car.color}}
</div>
```

Create Data

To create a record in Angular in-memory web api, you need to perform `http post` operation and pass object to be inserted in the database. To do that add a method `addCar` in `AppService` as shown in *code listing 12.14.*

Code Listing 12.14

```
addCar  (car: Car): Observable<Car> {
            return   this.http.post<Car>(this.apiurl,   car,
this.httpOptions).pipe(
         tap(data => console.log(data)),
         catchError(this.handleError)
      );
   }
```

We are performing HTTP POST operation, passing car object to be inserted and setting HTTP options such as HTTP headers to Application/ JSON. Angular in-memory web api will return newly created object ON successful completion of operation.

Next, you can use `addCar` method from `AppService` in component as shown in *code listing 12.15.*

Code Listing 12.15

```
addCar() {
      this.appservice.addCar(this.carFormGroup.value).
subscribe(data => {
        this.car = data;
        console.log(this.car);
     });
    this.getCars();
   }
```

We are calling **addCar** method of service and passing car object to be inserted. We have created car object using Reactive Form as shown in *code listing 12.16.*

Code Listing 12.16

```
this.carFormGroup = new FormGroup(
      {
         model : new FormControl(this.carToAdd.model),
         price : new FormControl(this.carToAdd.price),
         color: new FormControl(this.carToAdd.color)
        }
     );
```

On the template, form is created as shown in *code listing 12.17.*

Code Listing 12.17

```
<form [formGroup]='carFormGroup' (ngSubmit)='addCar()'
novalidate >
<input type="text" formControlName='model'
placeholder="Enter Model"   />
<input type="text" formControlName='color'
placeholder="Enter Color"   />
<input type="number" formControlName='price'
placeholder="Enter Price"   />
<button>Add Car</button>
```

Update Data

To update a record in Angular in-memory web api, you need to perform **`http put`** operation and pass object to be updated in the database. To do that, add a method **`updateCar`** in **`AppService`** as shown in *code listing 12.18.*

Code Listing 12.18

```
updateCar  (car: Car): Observable<null   |  Car> {
      return this.http.put<Car>(this.apiurl,   car,   this.
httpOptions).pipe(
      tap(data => console.log(data)),
      catchError(this.handleError)
   );
}
```

We are performing HTTP PUT operation, passing car object to be updated and setting HTTP options such as HTTP headers to Application/JSON. Next, you can use **`updateCar`** method from **`AppService`** in component as shown in *code listing 12.19.*

Code Listing 12.19

```
updateCar()   {
         this.appservice.getCar(this.idtoupdate).
subscribe(data => {
      this.carToUpdate = data;
      this.carToUpdate.model =  'updated model';
         this.appservice.updateCar(this.carToUpdate).
subscribe(data1 => {
         this.getCars();
      });
   });
```

On template, you can have a form like create data example to accept user input for update.

Delete Data

To delete a record in Angular in-memory web api, you need to perform **HTTP DELETE** operation and pass id o to be deleted in the database. To do that add a method **`deleteCar`** in **`AppService`** as shown in *code listing 12.20.*

Code Listing 12.20

```
deleteCar (id: number): Observable<Car> {
        const url = `${this.apiurl}/${id}`;
     return this.http.delete<Car>(url, this.httpOptions).
pipe(
          catchError(this.handleError)
      );
     }
```

We are performing HTTP DELETE operation, passing car id to be deleted and setting HTTP options such as HTTP headers to Application/JSON. Next, you can use **`deleteCar`** method from **`AppService`** in component as shown in *code listing 12.21.*

Code Listing 12.21

```
deleteCar()   {
            this.appservice.deleteCar(this.idtodelete).
subscribe(data => {
         this.getCars();
      });
   }
```

On template, you can have a form like create data example to accept user input for delete.

Summary

In Angular, you use $http service to perform HTTP requests to API. You can have local fake API to perform CRUD operations using Angular in-memory web api. In this chapter, you learnt the following topics:

- Angular in-memory web api
- Setting up Angular in-memory web api
- Create service to perform HTTP operations
- Read Data
- Create Data
- Update Data
- Delete Data