|  |
| --- |
| [Type the company name] |
| Angular 4 CRUD |
| *Crud Operation Using http class and json server* |
|  |
| **Sweta Gupta** |
| ***01 JAN 2018*** |

|  |
| --- |
| *“This Document will walk through Angular 4 CRUD example using Http class. We will perform create, read, update and delete operation here using Http class. Http performs HTTP request using XMLHttpRequest as default backend. Http class has methods such as post, get, put, delete etc. These methods return instance of Observable.”* |

**Angular 4 CRUD Operation using http and json server**

**“**This document will walk through Angular 4 CRUD example using Http class. We will perform create, read, update and delete operation here using Http class. Http performs HTTP request using XMLHttpRequest as default backend. Httpclass has methods such as post, get, put, delete etc. These methods return instance of Observable. To get data from the instance of Observable, we need to subscribe it. Http is an injectable class and can be instantiated using dependency injection with constructor. In our CRUD example, Http.post will perform create operation, Http.getwill perform read operation, Http.put will perform update operation and Http.delete will perform delete operation. Interaction with HTTP should be done within a service class to separate this code from component and hence that will make code easy to understand and help in bug fixing. To run the example we need a server that will respond our request URL. **“**

**Contents**

* [Technologies Used](https://www.concretepage.com/angular-2/angular-4-crud-example#technologies)
* [Project Structure](https://www.concretepage.com/angular-2/angular-4-crud-example#project)
* [Angular Http Class](https://www.concretepage.com/angular-2/angular-4-crud-example#Http)
* [CRUD Operation using Http](https://www.concretepage.com/angular-2/angular-4-crud-example#CRUD)
* [1. Import HttpModule](https://www.concretepage.com/angular-2/angular-4-crud-example#HttpModule)
* [2. Instantiate Http Claas](https://www.concretepage.com/angular-2/angular-4-crud-example#Instantiate)
* [3. Http.post](https://www.concretepage.com/angular-2/angular-4-crud-example#post)
* [4. Http.get](https://www.concretepage.com/angular-2/angular-4-crud-example#get)
* [5. Http.put](https://www.concretepage.com/angular-2/angular-4-crud-example#put)
* [6. Http.delete](https://www.concretepage.com/angular-2/angular-4-crud-example#delete)
* [7. RxJS Observable](https://www.concretepage.com/angular-2/angular-4-crud-example#Observable)
* [Using JSON-SERVER](https://www.concretepage.com/angular-2/angular-4-crud-example#JSON-SERVER)
* [Complete Example](https://www.concretepage.com/angular-2/angular-4-crud-example#Example)
* [Run Application](https://www.concretepage.com/angular-2/angular-4-crud-example#run)

### Technologies Used

Find the technologies being used in our example.   
1. Angular 4.2.4   
2. TypeScript 2.3.3   
3. Node.js 6.10.1   
4. Angular CLI 1.3.1   
5. Angular Compiler CLI 4.2.4

### Project Structure

Find the project structure of our demo application.

angular-demo

|

|--src

| |

| |--app

| | |

| | |--article.component.ts

| | |--article.service.ts

| | |--article.ts

| | |--article.component.html

| | |--article.component.css

| | |

| | |--app.component.ts

| | |--app.module.ts

| |

| |--assets

| | |

| | |--images

| | | |

| | | |--loading.gif

| | |

| |

| |--main.ts

| |--index.html

| |--styles.css

|

|--node\_modules

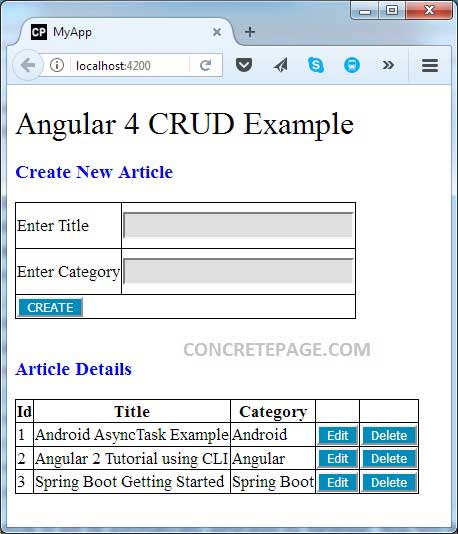
|--package.json

### Angular Http Class

Http performs HTTP request using XMLHttpRequest as default backend. Http is injectable. For any request it returns the instance of Observable. Http has following methods.   
**post**: Performs HTTP POST request.   
**get**: Performs HTTP GET request.   
**put**: Performs HTTP PUT request.   
**delete**: Performs HTTP DELETE request.   
**patch**: Performs HTTP PATCH request.   
**head**: Performs HTTP HEAD request.   
**options**: Performs HTTP OPTIONS request.   
**request**: Performs any type of HTTP request.

### CRUD Operation using Http

Now we will perform CREATE, READ, UPDATE and DELETE (CRUD) operation using Http. We will perform   
**1.** CREATE operation using Http.post method.   
**2.** READ operation using Http.get method.   
**3.** UPDATE operation using Http.put method.   
**4.** DELETE operation using Http.delete method.   
  
The output of our Angular 4 application for CRUD operation will be as follows.



Now we will discuss CRUD using Http step by step.

### 1. Import HttpModule

To use Http class, we need to import HttpModule in application module. HttpModule is from @angular/http API.

import { HttpModule } from '@angular/http';

@NgModule({

imports: [

---

HttpModule

],

------

})

export class AppModule { }

### 2. Instantiate Http Claas

Http is injectable and can be instantiated using constructor. We will perform dependency injection of Http using constructor. Http is the class of @angular/http API. Interaction with HTTP should be done within a service class to separate this code from component and hence that will make code easy to understand and help in bug fixing. Find a sample code snippet to instantiate Http using constructor within a service.

import { Http } from '@angular/http';

@Injectable()

export class ArticleService {

constructor(private http:Http) {

}

------

}

### 3. Http.post

We will perform create operation using Angular Http.post() method. It hits the request URL using HTTP POST method. Http.post() method syntax is as follows.

post(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response>

The description of parameters is given as below.   
**url**: This is the REST web service URL to create article.   
**body**: This is of any type object that will be passed to REST web service server. In our example we will create an Angular class as Article and pass its instance to body parameter.   
**options**: This is optional. This accepts the instance of Angular RequestOptions that is instantiated using Angular RequestOptionsArgs. Using RequestOptions we pass request parameter, request headers etc.   
  
Http.post() returns instance of Observable. Observable is a representation of any set of values over any amount of time.   
**Code**  
Find the code to create the article. Here we will use Angular Http.post() method.

createArticle(article: Article):Observable<number> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

return this.http.post(this.articleUrl, article, options)

.map(success => success.status)

.catch(this.handleError);

}

We are passing header Content-Type as application/json. After successful operation we are returning status code as an instance of Observable. Headers is the angular class that is used to configure request headers.

### 4. Http.get

We will perform read operation using Angular Http.get() method. It hits the URL using HTTP GET method. Find its syntax.

get(url: string, options?: RequestOptionsArgs) : Observable<Response>

Find the description of the parameters.   
**url**: Web service URL to read article.   
**options**: This is optional. It is used to pass request parameter, headers etc.   
  
Http.get() returns the instance of Observable.   
  
**Code**  
Find the Angular code using Http.get() that will pass request parameter to filter the result.

getArticleById(articleId: string): Observable<Article> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let cpParams = new URLSearchParams();

cpParams.set('id', articleId);

let options = new RequestOptions({ headers: cpHeaders, params: cpParams });

return this.http.get(this.articleUrl, options)

.map(this.extractData)

.catch(this.handleError);

}

URLSearchParams creates the query string in the URL.

### 5. Http.put

We will perform update operation using Angular 2 Http.put() method. It hits the URL using HTTP PUT method. Find its syntax.

put(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response>

Find the description of parameters.   
**url**: This is the REST web service URL to update article.   
**body**: This is of any type object that will be passed to REST web service server. In our example we will create an Angular class as Article and pass its instance to body parameter.   
**options**: This is optional. This is used to pass request parameter, request headers etc.   
  
Http.put() returns the instance of Observable.   
**Code**  
Find the angular code that is using Http.put() method to update the article.

updateArticle(article: Article):Observable<number> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

return this.http.put(this.articleUrl +"/"+ article.id, article, options)

.map(success => success.status)

.catch(this.handleError);

}

In our Angular application we have created a class as Article and we are passing its instance to Http.put()method. The article will be updated for the given article id.

### 6. Http.delete

We will perform delete operation using Angular Http.delete() method. Http.delete() hits the URL using HTTP DELETE method. Find its syntax.

delete(url: string, options?: RequestOptionsArgs) : Observable<Response>

Find the description of the parameters.   
**url**: Web service URL to delete article.   
**options**: This is optional. It is used to pass request parameter, headers etc.   
  
Http.get() returns the instance of Observable.   
**Code**  
Find the code using Http.delete() method to delete article by id.

deleteArticleById(articleId: string): Observable<number> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

return this.http.delete(this.articleUrl +"/"+ articleId)

.map(success => success.status)

.catch(this.handleError);

}

In the path parameter we are passing article id to delete the article.

### 7. RxJS Observable

Observable is a RxJS API. Observable is a representation of any set of values over any amount of time. All angular Http methods return instance of Observable. Find some of its operators.   
**map**: It applies a function to each value emitted by source Observable and returns finally an instance of Observable.   
**catch**: It is called when an error is occurred. catch also returns Observable.   
  
To fetch data from instance of Observable we need to subscribe it using RxJS subscribe operator. The actual hit to server goes only when we call subscribe or use async pipe on instance of Observable. In our example we will use subscribe operator to fetch data from Observable. Find the sample code.

this.articleService.getAllArticles()

.subscribe(articles => {

//Data from server has been received.

//perform operation on articles

}

);

### Using JSON-SERVER

json-server is fake REST API to test applications. It is very easy to install and use it with few steps.   
**Step-1**: Open the command prompt and run the following command.

npm install -g json-server

Now we are ready with json-server .   
**Step-2**: Create a db.json file in a directory. In our demo we will perform operations on articles with three fields that are id, title and category.   
**db.json**

{

"articles": [

{

"id": 1,

"title": "Android AsyncTask Example",

"category": "Android"

}

]

}

**Step-3**: Now we need to run the json-server. Go to the directory where db.json file is lying and then run following command.

json-server --watch db.json

**Step-4**: In this way, we are ready to use json-server with following URLs.

http://localhost:3000/articles

The above URL will fetch all articles.   
**Step-5**: For CRUD operation we need to use following URLs.   
**1. GET /articles** : Fetches all articles.   
**2. GET /articles/1** : Fetches article by id.   
**3. POST /articles** : Create article.   
**4. PUT /articles/1** : Update article by id.   
**5. DELETE /articles/1** : Delete article by id.   
  
Find the [link](https://www.npmjs.com/package/json-server" \t "_blank) for json-server reference.

### Complete Example

**article.service.ts**

import { Injectable } from '@angular/core';

import { Http, Response, Headers, URLSearchParams, RequestOptions } from '@angular/http';

import { Observable } from 'rxjs';

import { Article } from './article';

@Injectable()

export class ArticleService {

//URL for CRUD operations

articleUrl = "http://localhost:3000/articles";

//Create constructor to get Http instance

constructor(private http:Http) {

}

//Fetch all articles

getAllArticles(): Observable<Article[]> {

return this.http.get(this.articleUrl)

.map(this.extractData)

.catch(this.handleError);

}

//Create article

createArticle(article: Article):Observable<number> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

return this.http.post(this.articleUrl, article, options)

.map(success => success.status)

.catch(this.handleError);

}

//Fetch article by id

getArticleById(articleId: string): Observable<Article> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

console.log(this.articleUrl +"/"+ articleId);

return this.http.get(this.articleUrl +"/"+ articleId)

.map(this.extractData)

.catch(this.handleError);

}

//Update article

updateArticle(article: Article):Observable<number> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

return this.http.put(this.articleUrl +"/"+ article.id, article, options)

.map(success => success.status)

.catch(this.handleError);

}

//Delete article

deleteArticleById(articleId: string): Observable<number> {

let cpHeaders = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: cpHeaders });

return this.http.delete(this.articleUrl +"/"+ articleId)

.map(success => success.status)

.catch(this.handleError);

}

private extractData(res: Response) {

let body = res.json();

return body;

}

private handleError (error: Response | any) {

console.error(error.message || error);

return Observable.throw(error.status);

}

}

**article.ts**

export class Article {

constructor(public id: number, public title: string, public category: string) {

}

}

**article.component.ts**

import { Component, OnInit } from '@angular/core';

import { FormControl, FormGroup, Validators } from '@angular/forms';

import { ArticleService } from './article.service';

import { Article } from './article';

@Component({

selector: 'app-article',

templateUrl: './article.component.html',

styleUrls: ['./article.component.css']

})

export class ArticleComponent implements OnInit {

//Component properties

allArticles: Article[];

statusCode: number;

requestProcessing = false;

articleIdToUpdate = null;

processValidation = false;

//Create form

articleForm = new FormGroup({

title: new FormControl('', Validators.required),

category: new FormControl('', Validators.required)

});

//Create constructor to get service instance

constructor(private articleService: ArticleService) {

}

//Create ngOnInit() and and load articles

ngOnInit(): void {

this.getAllArticles();

}

//Fetch all articles

getAllArticles() {

this.articleService.getAllArticles()

.subscribe(

data => this.allArticles = data,

errorCode => this.statusCode = errorCode);

}

//Handle create and update article

onArticleFormSubmit() {

this.processValidation = true;

if (this.articleForm.invalid) {

return; //Validation failed, exit from method.

}

//Form is valid, now perform create or update

this.preProcessConfigurations();

let article = this.articleForm.value;

if (this.articleIdToUpdate === null) {

//Generate article id then create article

this.articleService.getAllArticles()

.subscribe(articles => {

//Generate article id

let maxIndex = articles.length - 1;

let articleWithMaxIndex = articles[maxIndex];

let articleId = articleWithMaxIndex.id + 1;

article.id = articleId;

//Create article

this.articleService.createArticle(article)

.subscribe(successCode => {

this.statusCode = successCode;

this.getAllArticles();

this.backToCreateArticle();

},

errorCode => this.statusCode = errorCode

);

});

} else {

//Handle update article

article.id = this.articleIdToUpdate;

this.articleService.updateArticle(article)

.subscribe(successCode => {

this.statusCode = successCode;

this.getAllArticles();

this.backToCreateArticle();

},

errorCode => this.statusCode = errorCode);

}

}

//Load article by id to edit

loadArticleToEdit(articleId: string) {

this.preProcessConfigurations();

this.articleService.getArticleById(articleId)

.subscribe(article => {

this.articleIdToUpdate = article.id;

this.articleForm.setValue({ title: article.title, category: article.category });

this.processValidation = true;

this.requestProcessing = false;

},

errorCode => this.statusCode = errorCode);

}

//Delete article

deleteArticle(articleId: string) {

this.preProcessConfigurations();

this.articleService.deleteArticleById(articleId)

.subscribe(successCode => {

//this.statusCode = successCode;

//Expecting success code 204 from server

this.statusCode = 204;

this.getAllArticles();

this.backToCreateArticle();

},

errorCode => this.statusCode = errorCode);

}

//Perform preliminary processing configurations

preProcessConfigurations() {

this.statusCode = null;

this.requestProcessing = true;

}

//Go back from update to create

backToCreateArticle() {

this.articleIdToUpdate = null;

this.articleForm.reset();

this.processValidation = false;

}

}

**article.component.html**

<h1>Angular 4 CRUD Example</h1>

<h3 \*ngIf="articleIdToUpdate; else create">

Update Article for Id: {{articleIdToUpdate}}

</h3>

<ng-template #create>

<h3> Create New Article </h3>

</ng-template>

<div>

<form [formGroup]="articleForm" (ngSubmit)="onArticleFormSubmit()">

<table>

<tr><td>Enter Title</td><td><input formControlName="title">

<label \*ngIf="articleForm.get('title').invalid && processValidation" [ngClass] = "'error'"> Title is required. </label>

</td></tr>

<tr><td>Enter Category</td><td><input formControlName="category">

<label \*ngIf="articleForm.get('category').invalid && processValidation" [ngClass] = "'error'"> Category is required. </label>

</td></tr>

<tr><td colspan="2">

<button \*ngIf="!articleIdToUpdate">CREATE</button>

<button \*ngIf="articleIdToUpdate">UPDATE</button>

<button (click)="backToCreateArticle()" \*ngIf="articleIdToUpdate">Go Back</button>

</td></tr>

</table>

</form>

<br/>

<div \*ngIf="statusCode; else processing">

<div \*ngIf="statusCode === 201" [ngClass] = "'success'">

Article added successfully.

</div>

<div \*ngIf="statusCode === 409" [ngClass] = "'success'">

Article already exists.

</div>

<div \*ngIf="statusCode === 200" [ngClass] = "'success'">

Article updated successfully.

</div>

<div \*ngIf="statusCode === 204" [ngClass] = "'success'">

Article deleted successfully.

</div>

<div \*ngIf="statusCode === 500" [ngClass] = "'error'">

Internal Server Error.

</div>

</div>

<ng-template #processing>

<img \*ngIf="requestProcessing" src="assets/images/loading.gif">

</ng-template>

</div>

<h3>Article Details</h3>

<table>

<tr><th> Id</th> <th>Title</th><th>Category</th><th></th><th></th></tr>

<tr \*ngFor="let article of allArticles" >

<td>{{article.id}}</td> <td>{{article.title}}</td> <td>{{article.category}}</td>

<td><button type="button" (click)="loadArticleToEdit(article.id)">Edit</button> </td>

<td><button type="button" (click)="deleteArticle(article.id)">Delete</button></td>

</tr>

</table>

**article.component.css**

h1 {

font-size: 2.0em;

margin: 20px 0 0 0;

font-weight: 400;

}

h3 {

color: blue;

}

table {

border-collapse: collapse;

}

table, th, td {

border: 1px solid black;

font-size:17px;

}

input {

width: 225px;

margin: 8px 0;

background-color: #dfdfdf;

font-size:17px;

}

button {

background-color: #008CBA;

color: white;

}

.error{

color: red;

font-size: 20px;

}

.success{

color: green;

font-size: 20px;

}

**app.component.ts**

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: `

<app-article></app-article>

`

})

export class AppComponent {

}

**app.module.ts**

import { NgModule } from '@angular/core';

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

import { ReactiveFormsModule } from '@angular/forms';

import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

import { ArticleComponent } from './article.component';

import { ArticleService } from './article.service';

@NgModule({

imports: [

BrowserModule,

HttpModule,

ReactiveFormsModule

],

declarations: [

AppComponent,

ArticleComponent

],

providers: [

ArticleService

],

bootstrap: [

AppComponent

]

})

export class AppModule { }

### Run Application

To run the application, find following steps.   
**1.** Start JSON-SERVER. To install and start JSON-SERVER, follow the steps given above in the article.   
**2.** Download source code using download link given below on this page.   
**3.** Use downloaded **src** in your angular CLI application. To install angular CLI, find the [link](https://angular.io/guide/quickstart" \t "_blank).   
**4.** Run **ng serve** using command prompt.   
**5.** Now access the URL **http://localhost:4200**   
  
**Note**: Angular version 4.3 has introduced HttpClient to perform HTTP requests. If we are using Angular version 4.3 or higher, we should use HttpClient. Find a sample example to handle GET request using HttpClient.