**Angular 8 - Http Client Programming**

[Next](https://www.tutorialspoint.com/angular8/angular8_angular_material.htm)

Http client programming is a must needed feature in every modern web application. Nowadays, lot of application exposes their functionality through REST API (functionality over HTTP protocol). With this in mind, Angular Team provides extensive support to access HTTP server. Angular provides a separate module, **HttpClientModule** and a service, **HttpClient** to do HTTP programming.

Let us learn how to how to use **HttpClient** service in this chapter. Developer should have a basic knowledge in Http programming to understand this chapter.

Expense REST API

The prerequisite to do Http programming is the basic knowledge of Http protocol and REST API technique. Http programming involves two part, server and client. Angular provides support to create client side application. **Express** a popular web framework provides support to create server side application.

Let us create an **Expense Rest API** using express framework and then access it from our **ExpenseManager** application using Angular HttpClient service.

Open a command prompt and create a new folder, **express-rest-api**.

cd /go/to/workspace

mkdir express-rest-api

cd expense-rest-api

Initialise a new node application using below command −

npm init

**npm init** will ask some basic questions like project name (express-rest-api), entry point (server.js), etc., as mentioned below −

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (expense-rest-api)

version: (1.0.0)

description: Rest api for Expense Application

entry point: (index.js) server.js

test command:

git repository:

keywords:

author:

license: (ISC)

About to write to \path\to\workspace\expense-rest-api\package.json: {

"name": "expense-rest-api",

"version": "1.0.0",

"description": "Rest api for Expense Application",

"main": "server.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

Is this OK? (yes) yes

Install **express, sqlite** and **cors** modules using below command −

npm install express sqlite3 cors

Create a new file **sqlitedb.js** and place below code −

var sqlite3 = require('sqlite3').verbose()

const DBSOURCE = "expensedb.sqlite"

let db = new sqlite3.Database(DBSOURCE, (err) => {

if (err) {

console.error(err.message)

throw err

}else{

console.log('Connected to the SQLite database.')

db.run(`CREATE TABLE expense (

id INTEGER PRIMARY KEY AUTOINCREMENT,

item text,

amount real,

category text,

location text,

spendOn text,

createdOn text

)`,

(err) => {

if (err) {

console.log(err);

}else{

var insert = 'INSERT INTO expense (item, amount, category, location, spendOn, createdOn) VALUES (?,?,?,?,?,?)'

db.run(insert, ['Pizza', 10, 'Food', 'KFC', '2020-05-26 10:10', '2020-05-26 10:10'])

db.run(insert, ['Pizza', 9, 'Food', 'Mcdonald', '2020-05-28 11:10', '2020-05-28 11:10'])

db.run(insert, ['Pizza', 12, 'Food', 'Mcdonald', '2020-05-29 09:22', '2020-05-29 09:22'])

db.run(insert, ['Pizza', 15, 'Food', 'KFC', '2020-06-06 16:18', '2020-06-06 16:18'])

db.run(insert, ['Pizza', 14, 'Food', 'Mcdonald', '2020-06-01 18:14', '2020-05-01 18:14'])

}

}

);

}

});

module.exports = db

Here, we are creating a new sqlite database and load some sample data.

Open server.js and place below code −

var express = require("express")

var cors = require('cors')

var db = require("./sqlitedb.js")

var app = express()

app.use(cors());

var bodyParser = require("body-parser");

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

var HTTP\_PORT = 8000

app.listen(HTTP\_PORT, () => {

console.log("Server running on port %PORT%".replace("%PORT%",HTTP\_PORT))

});

app.get("/", (req, res, next) => {

res.json({"message":"Ok"})

});

app.get("/api/expense", (req, res, next) => {

var sql = "select \* from expense"

var params = []

db.all(sql, params, (err, rows) => {

if (err) {

res.status(400).json({"error":err.message});

return;

}

res.json(rows)

});

});

app.get("/api/expense/:id", (req, res, next) => {

var sql = "select \* from expense where id = ?"

var params = [req.params.id]

db.get(sql, params, (err, row) => {

if (err) {

res.status(400).json({"error":err.message});

return;

}

res.json(row)

});

});

app.post("/api/expense/", (req, res, next) => {

var errors=[]

if (!req.body.item){

errors.push("No item specified");

}

var data = {

item : req.body.item,

amount: req.body.amount,

category: req.body.category,

location : req.body.location,

spendOn: req.body.spendOn,

createdOn: req.body.createdOn,

}

var sql = 'INSERT INTO expense (item, amount, category, location, spendOn, createdOn) VALUES (?,?,?,?,?,?)'

var params =[data.item, data.amount, data.category, data.location, data.spendOn, data.createdOn]

db.run(sql, params, function (err, result) {

if (err){

res.status(400).json({"error": err.message})

return;

}

data.id = this.lastID;

res.json(data);

});

})

app.put("/api/expense/:id", (req, res, next) => {

var data = {

item : req.body.item,

amount: req.body.amount,

category: req.body.category,

location : req.body.location,

spendOn: req.body.spendOn

}

db.run(

`UPDATE expense SET

item = ?,

amount = ?,

category = ?,

location = ?,

spendOn = ?

WHERE id = ?`,

[data.item, data.amount, data.category, data.location,data.spendOn, req.params.id],

function (err, result) {

if (err){

console.log(err);

res.status(400).json({"error": res.message})

return;

}

res.json(data)

});

})

app.delete("/api/expense/:id", (req, res, next) => {

db.run(

'DELETE FROM expense WHERE id = ?',

req.params.id,

function (err, result) {

if (err){

res.status(400).json({"error": res.message})

return;

}

res.json({"message":"deleted", changes: this.changes})

});

})

app.use(function(req, res){

res.status(404);

});

Here, we create a basic CURD rest api to select, insert, update and delete expense entry.

Run the application using below command −

npm run start

Open a browser, enter **http://localhost:8000/**and press enter. You will see below response −

{

"message": "Ok"

}

It confirms our application is working fine.

Change the url to **http://localhost:8000/api/expense** and you will see all the expense entries in JSON format.

[

{

"id": 1,

"item": "Pizza",

"amount": 10,

"category": "Food",

"location": "KFC",

"spendOn": "2020-05-26 10:10",

"createdOn": "2020-05-26 10:10"

},

{

"id": 2,

"item": "Pizza",

"amount": 14,

"category": "Food",

"location": "Mcdonald",

"spendOn": "2020-06-01 18:14",

"createdOn": "2020-05-01 18:14"

},

{

"id": 3,

"item": "Pizza",

"amount": 15,

"category": "Food",

"location": "KFC",

"spendOn": "2020-06-06 16:18",

"createdOn": "2020-06-06 16:18"

},

{

"id": 4,

"item": "Pizza",

"amount": 9,

"category": "Food",

"location": "Mcdonald",

"spendOn": "2020-05-28 11:10",

"createdOn": "2020-05-28 11:10"

},

{

"id": 5,

"item": "Pizza",

"amount": 12,

"category": "Food",

"location": "Mcdonald",

"spendOn": "2020-05-29 09:22",

"createdOn": "2020-05-29 09:22"

}

]

Finally, we created a simple CURD REST API for expense entry and we can access the REST API from our Angular application to learn HttpClient module.

Configure Http client

Let us learn how to configure **HttpClient** service in this chapter.

**HttpClient** service is available inside the **HttpClientModule** module, which is available inside the @angular/common/http package.

To register **HttpClientModule** module −

Import the HttpClientModule in **AppComponent**

import { HttpClientModule } from '@angular/common/http';

Include HttpClientModule in imports meta data of AppComponent.

@NgModule({

imports: [

BrowserModule,

// import HttpClientModule after BrowserModule.

HttpClientModule,

]

})

export class AppModule {}

Create expense service

Let us create a new service **ExpenseEntryService** in our **ExpenseManager** application to interact with **Expense REST API**. ExpenseEntryService will get the latest expense entries, insert new expense entries, modify existing expense entries and delete the unwanted expense entries.

Open command prompt and go to project root folder.

cd /go/to/expense-manager

Start the application.

ng serve

Run the below command to generate an Angular service, **ExpenseService**.

ng generate service ExpenseEntry

This will create two Typescript files (expense entry service & its test) as specified below −

CREATE src/app/expense-entry.service.spec.ts (364 bytes)

CREATE src/app/expense-entry.service.ts (141 bytes)

Open **ExpenseEntryService** (src/app/expense-entry.service.ts) and import **ExpenseEntry, throwError** and **catchError** from rxjs library and import **HttpClient, HttpHeaders** and **HttpErrorResponse** from @angular/common/http package.

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

import { ExpenseEntry } from './expense-entry'; import { throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

import { HttpClient, HttpHeaders, HttpErrorResponse } from

'@angular/common/http';

Inject the HttpClient service into our service.

constructor(private httpClient : HttpClient) { }

Create a variable, **expenseRestUrl** to specify the **Expense Rest API** endpoints.

private expenseRestUrl = 'http://localhost:8000/api/expense';

Create a variable, **httpOptions** to set the Http Header option. This will be used during the Http Rest API call by Angular **HttpClient** service.

private httpOptions = {

headers: new HttpHeaders( { 'Content-Type': 'application/json' })

};

The complete code is as follows −

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

import { ExpenseEntry } from './expense-entry';

import { Observable, throwError } from 'rxjs';

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

import { HttpClient, HttpHeaders, HttpErrorResponse } from '@angular/common/http';

@Injectable({

providedIn: 'root'

})

export class ExpenseEntryService {

private expenseRestUrl = 'api/expense';

private httpOptions = {

headers: new HttpHeaders( { 'Content-Type': 'application/json' })

};

constructor(

private httpClient : HttpClient) { }

}

HTTP GET

HttpClient provides get() method to fetch data from a web page. The main argument is the target web url. Another optional argument is the option object with below format −

{

headers?: HttpHeaders | {[header: string]: string | string[]},

observe?: 'body' | 'events' | 'response',

params?: HttpParams|{[param: string]: string | string[]},

reportProgress?: boolean,

responseType?: 'arraybuffer'|'blob'|'json'|'text',

withCredentials?: boolean,

}

Here,

* **headers** − HTTP Headers of the request, either as string, array of string or array of HttpHeaders.
* **observe** − Process the response and return the specific content of the response. Possible values are body, response and events. The default option of observer is body.
* **params** − HTTP parameters of the request, either as string, array of string or array of **HttpParams**.
* **reportProgress** − Whether to report the progress of the process or not (true or false).
* **responseType** − Refers the format of the response. Possible values are **arraybuffer, blob, json** and **text**.
* **withCredentials** − Whether the request has credentials or not (true or false).

All options are optional.

**get()** method returns the response of the request as **Observable**. The returned Observable emit the data when the response is received from the server.

The sample code to use **get()** method is as follows −

httpClient.get(url, options)

.subscribe( (data) => console.log(data) );

Typed Response

**get()** method has an option to return observables, which emits typed response as well. The sample code to get typed response (ExpenseEntry) is as follows:

httpClient.get<T>(url, options) .subscribe( (data: T) => console.log(data) );

Handling errors

Error handling is one of the important aspect in the HTTP programming. Encountering error is one of the common scenario in HTTP programming.

Errors in HTTP Programming can be categories into two groups −

* Client side issues can occur due to network failure, misconfiguration, etc., If client side error happens, then the **get()** method throws **ErrorEvent** object.
* Server side issues can occur due to wrong url, server unavailability, server programming errors, etc.,

Let us write a simple error handling for our **ExpenseEntryService** service.

private httpErrorHandler (error: HttpErrorResponse) {

if (error.error instanceof ErrorEvent) {

console.error("A client side error occurs. The error message is " + error.message);

} else {

console.error(

"An error happened in server. The HTTP status code is " + error.status + " and the error returned is " + error.message);

}

return throwError("Error occurred. Pleas try again");

}

The error function can be called in **get()** as specified below −

httpClient.get(url, options)

.pipe(catchError(this.httpErrorHandler)

.subscribe( (data) => console.log(data) )

Handle failed request

As we mentioned earlier, errors can happen and one way is to handle it. Another option is to try for certain number of times. If the request failed due to network issue or the HTTP server is temporarily offline, the next request may succeed.

We can use **rxjs** library’s **retry** operator in this scenario as specified below

httpClient.get(url, options)

.pipe(

retry(5),

catchError(this.httpErrorHandler))

.subscribe( (data) => console.log(data) )

Fetch expense entries

Let us do the actual coding to fetch the expenses from **Expense Rest API** in our ExpenseManager application.

Open command prompt and go to project root folder.

cd /go/to/expense-manager

Start the application.

ng serve

Add **getExpenseEntries()** and **httpErrorHandler()** method in **ExpenseEntryService** (src/app/expense-entry.service.ts) service.

getExpenseEntries() : Observable<ExpenseEntry[]> {

return this.httpClient.get<ExpenseEntry[]>(this.expenseRestUrl, this.httpOptions)

.pipe(retry(3),catchError(this.httpErrorHandler));

}

getExpenseEntry(id: number) : Observable<ExpenseEntry> {

return this.httpClient.get<ExpenseEntry>(this.expenseRestUrl + "/" + id, this.httpOptions)

.pipe(

retry(3),

catchError(this.httpErrorHandler)

);

}

private httpErrorHandler (error: HttpErrorResponse) {

if (error.error instanceof ErrorEvent) {

console.error("A client side error occurs. The error message is " + error.message);

} else {

console.error(

"An error happened in server. The HTTP status code is " + error.status + " and the error returned is " + error.message);

}

return throwError("Error occurred. Pleas try again");

}

Here,

* **getExpenseEntries()** calls the **get()** method using expense end point and also configures the error handler. Also, it configures **httpClient** to try for maximum of 3 times in case of failure. Finally, it returns the response from server as typed **(ExpenseEntry[])** Observable object.
* **getExpenseEntry** is similar to getExpenseEntries() except it passes the id of the ExpenseEntry object and gets ExpenseEntry Observable object.

The complete coding of **ExpenseEntryService** is as follows −

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

import { ExpenseEntry } from './expense-entry';

import { Observable, throwError } from 'rxjs';

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

import { HttpClient, HttpHeaders, HttpErrorResponse } from '@angular/common/http';

@Injectable({

providedIn: 'root'

})

export class ExpenseEntryService {

private expenseRestUrl = 'http://localhost:8000/api/expense';

private httpOptions = {

headers: new HttpHeaders( { 'Content-Type': 'application/json' })

};

constructor(private httpClient : HttpClient) { }

getExpenseEntries() : Observable {

return this.httpClient.get(this.expenseRestUrl, this.httpOptions)

.pipe(

retry(3),

catchError(this.httpErrorHandler)

);

}

getExpenseEntry(id: number) : Observable {

return this.httpClient.get(this.expenseRestUrl + "/" + id, this.httpOptions)

.pipe(

retry(3),

catchError(this.httpErrorHandler)

);

}

private httpErrorHandler (error: HttpErrorResponse) {

if (error.error instanceof ErrorEvent) {

console.error("A client side error occurs. The error message is " + error.message);

} else {

console.error(

"An error happened in server. The HTTP status code is " + error.status + " and the error returned is " + error.message);

}

return throwError("Error occurred. Pleas try again");

}

}

Open **ExpenseEntryListComponent** (src-entry-list-entry-list.component.ts) and inject **ExpenseEntryService** through constructor as specified below:

constructor(private debugService: DebugService, private restService :

ExpenseEntryService ) { }

Change the **getExpenseEntries()** function. Call getExpenseEntries() method from **ExpenseEntryService** instead of returning the mock items.

getExpenseItems() {

this.restService.getExpenseEntries()

.subscribe( data =− this.expenseEntries = data );

}

The complete **ExpenseEntryListComponent** coding is as follows −

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

import { ExpenseEntry } from '../expense-entry';

import { DebugService } from '../debug.service';

import { ExpenseEntryService } from '../expense-entry.service';

@Component({

selector: 'app-expense-entry-list',

templateUrl: './expense-entry-list.component.html',

styleUrls: ['./expense-entry-list.component.css'],

providers: [DebugService]

})

export class ExpenseEntryListComponent implements OnInit {

title: string;

expenseEntries: ExpenseEntry[];

constructor(private debugService: DebugService, private restService : ExpenseEntryService ) { }

ngOnInit() {

this.debugService.info("Expense Entry List component initialized");

this.title = "Expense Entry List";

this.getExpenseItems();

}

getExpenseItems() {

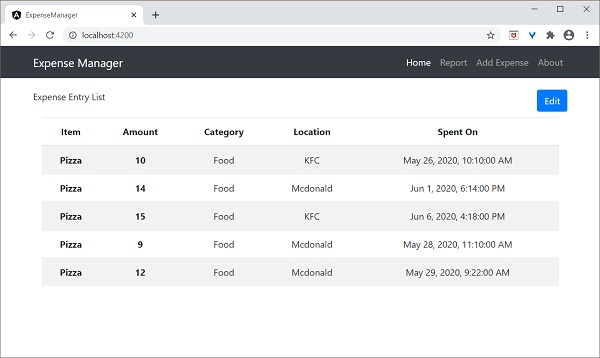
this.restService.getExpenseEntries()

.subscribe( data => this.expenseEntries = data );

}

}

Finally, check the application and you will see the below response.



HTTP POST

HTTP POST is similar to HTTP GET except that the post request will send the necessary data as posted content along with the request. HTTP POST is used to insert new record into the system.

**HttpClient** provides **post()** method, which is similar to **get()** except it support extra argument to send the data to the server.

Let us add a new method, **addExpenseEntry()** in our **ExpenseEntryService** to add new expense entry as mentioned below −

addExpenseEntry(expenseEntry: ExpenseEntry): Observable<ExpenseEntry> {

return this.httpClient.post<ExpenseEntry>(this.expenseRestUrl, expenseEntry, this.httpOptions)

.pipe(

retry(3),

catchError(this.httpErrorHandler)

);

}

HTTP PUT

HTTP PUT is similar to HTTP POST request. HTTP PUT is used to update existing record in the system.

**httpClient** provides **put()** method, which is similar to **post()**.

Update expense entry

Let us add a new method, **updateExpenseEntry()** in our **ExpenseEntryService** to update existing expense entry as mentioned below:

updateExpenseEntry(expenseEntry: ExpenseEntry): Observable<ExpenseEntry> {

return this.httpClient.put<ExpenseEntry>(this.expenseRestUrl + "/" + expenseEntry.id, expenseEntry, this.httpOptions)

.pipe(

retry(3),

catchError(this.httpErrorHandler)

);

}

HTTP DELETE

HTTP DELETE is similar to http GET request. HTTP DELETE is used to delete entries in the system.

**httpclient** provides **delete()** method, which is similar to **get()**.

Delete expense entry

Let us add a new method, **deleteExpenseEntry()** in our **ExpenseEntryService** to delete existing expense entry as mentioned below −

deleteExpenseEntry(expenseEntry: ExpenseEntry | number) : Observable<ExpenseEntry> {

const id = typeof expenseEntry == 'number' ? expenseEntry : expenseEntry.id

const url = `${this.expenseRestUrl}/${id}`;

return this.httpClient.delete<ExpenseEntry>(url, this.httpOptions)

.pipe(

retry(3),

catchError(this.httpErrorHandler)

);

}

Print Pag