**RESTFUL API WITH EXPRESS**

* Express is a middle-ware framework used in network-based applications across multi-tiers.
* Express handles functionalities like

1. Body-parser
2. CORS(Cross Origin Recourse Sharing)
3. Accessing static files
4. Handling API requests

Body-parser:-

* Body-parser is a library used by middle-ware to transmit the data to JSON form.
* Body-parser enables URL encoding and converts the data to JSON form.
* Syntax:

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

app.use(bodyParser.urlencoded({

    extended:true

}));

app.use(bodyParser.json());

CORS:-

* Distributed applications have to share data and resources across the origin.
* The server will block the data sharing if it is from a remote resource, it requires configuration of CORS in the middle-ware.
* A response object allows different content types and methods , it includes the following:

1. Access-Control-Allow-Origin.
2. Access-Control-Allow-Headers.
3. Access-Control-Allow-Methods.

* Headers define content type.
* Methods define request type [GET, POST].
* Origin define the remote location.

**Creating a RESTful API with NodeJS**

**Api.js**

var mongoClient= require('mongodb').MongoClient;

var express = require('express');

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

var url = 'mongodb://127.0.0.1:27017';

//Configuring API

var app= express();

//Configuring Bodyparser

app.use(bodyParser.urlencoded({

    extended:true

}));

app.use(bodyParser.json());

//configuring CORS

app.use(function(req,res,next){

    res.header('Access-Control-Allow-Origin','\*');

    res.header('Allow-Control-Allow-Header,Origin,X-Requested-With,Content-Type,Accept');

    res.header('Allow-Control-Allow-Methods','GET','POST','PUT','DELETE');

    next();

});

//Create API Request

app.get("/api/getProducts", function(req,res){

    mongoClient.connect(url,function(err, client){

        if(!err){

            var database= client.db('test');

            database.collection("products").find({}).toArray(function(err, documents){

                if(!err){

                    res.send(documents);

                }

            })

        }

    })

})

app.post("/api/addProducts", function(req,res){

    mongoClient.connect(url, function(err, client){

        if(!err){

            var database= client.db('test');

            var data={

                product\_id: req.body.product\_id,

                product\_name: req.body.product\_name,

                price: req.body.price,

                mfd: req.body.mfd,

                photo: req.body.photo,

                category\_id: req.body.category\_id,

                category: req.body.category,

            };

            database.collection('products').insert(data,function(err, result){

                if(!err){

                    console.log('Record Inserted');

                }

            })

        }

    })

})

app.listen(8080);

console.log('server started at port 8080');

**RXJS**

**[Reactive Extensible JavaScript ]**

* It is a reactive extensible javascript library.
* It is a library that provides asynchronous techniques to interact with any server site technologies.
* It provides various functions that are responsible for handling various asynchronous request and response.
* It is reactive extension for javascript which handles mapping, filtering, iterating like operations over the data.
* The commonly used components of rxjs are:

1. Observable:

It represents a type of an invokable collection for identifying the type of data returned on request.

1. Observer:

It is a collection of call-backs that knows how to listen to the values delivered by observable.

1. Subscription:

It represents the execution of an observable and also responsible for cancel the execution.

1. Operators:

It provides a set of functions that enable the operation like map, filter, concat etc.

1. Subject:

It is similar to an event emitter that emits values based on specific state of application.

1. Schedulers:

They are used to perform scheduled events on regular time intervals.

**Angular HTTP Client**

* Angular provides a httpClient Module that internally uses XmlHttpRequest object to handle the http request from client.
* It is a service provided by angular and used to manage various request from client , it includes GET, POST, PUT, DELETE etc..
* The httpClient is defined in @angular/common/http library and is controlled by httpClient Module.
* Example:

1. Go to server side port and start the API

>node api.js

2. Go to Angular project and add a new file into ‘app’ folder by name

‘**Iproduct.ts’**

interface IProduct {

    product\_id;

    product\_name;

    price;

    mfd;

    photo;

    categoty\_id;

    category;

}

3. **Add a new service to the project**

>ng g s apidata –spec=false

4. **apidata.service.ts**

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

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

import { Observable } from 'rxjs';

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

@Injectable()

export class ApidataService {

  public products = [];

  public apiurl = 'http://127.0.0.1:8080/api/getProducts';

  public posturl = 'http://127.0.0.1:8080/api/addProducts';

  constructor(private http: HttpClient) { }

  public GetProducts(): Observable<IProduct[]> {

    return this.http.get<IProduct[]>(this.apiurl);

  }

}

5. **Add a component**

**>**ng g c demo –spec=false

6. Go to ‘demo.component.ts’

import { ApidataService } from './../apidata.service';

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

@Component({

  selector: 'app-demo',

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

  styleUrls: ['./demo.component.css']

})

export class DemoComponent implements OnInit {

  public products = [];

  constructor(private data: ApidataService) { }

  ngOnInit() {

    this.data.GetProducts().subscribe(data => this.products = data);

  }

6. Go to ‘app.module.ts’

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

imports: [

    HttpClientModule

  ],

  providers: [ApidataService]

**NOTE:**

The subscribe function of RXJS is going to return various error status codes and status text which includes the following.

|  |  |
| --- | --- |
| status code | reason |
| 100  200  201  300  301  302  400  401  403  404  407  408  413  500  501 | Continue  Ok  Created  Multiple choices  Moved permanently  Found  Bad request  Unauthorised  Forbidden  Not found  Proxy authentication required  Request time out  Request entity too-large  Internal server error  Not implemented |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**RXJS error HANDLING**

* RxJs provides a library for handling the exceptions thrown by the asynchronous methods.
* The ‘rxjs/operators’ will provide the following error methods

1. throwError
2. catchError

* throwError uses HttpErrorResponce to identify the error type and catchError will catch the exceptions thrown by error response.
* You can concartinate the exceptions to service methods by using ‘pipe()’ function.

Example:

1. **Go to ‘apidata.service.ts’**

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

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

import { Observable } from 'rxjs';

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

@Injectable()

export class ApidataService {

  public products = [];

  public apiurl = 'http://127.0.0.1:8080/api/getProducts';

  public posturl = 'http://127.0.0.1:8080/api/addProducts';

  constructor(private http: HttpClient) { }

  public GetProducts(): Observable<IProduct[]> {

    return this.http.get<IProduct[]>(this.apiurl);

  }

  /\*\*

   \* Observable: A sequence of items that arrive asynchronously over time.

   \* HTTP Call - single item.

   \* single item- HTTP Responce.

   \*/

  public AddProducts(data) {

 return this.http.post<any>(this.posturl, data).pipe(catchError(this.catchError));

  }

  public catchError(error: HttpErrorResponse) {

    return throwError(error.statusText);

  }

}

**2.Import ‘FormsModule’ in app.module.ts**

 imports: [

    FormsModule

  ],

**3. Go to ‘demo.component.ts’**

import { ApidataService } from './../apidata.service';

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

@Component({

  selector: 'app-demo',

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

  styleUrls: ['./demo.component.css']

})

export class DemoComponent implements OnInit {

  public products = [];

  constructor(private data: ApidataService) { }

  ngOnInit() {

    this.data.GetProducts().subscribe(data => this.products = data);

  }

  /\*\*

   \* Obsrvable will not provide data to any one until someone subscribe to it.

   \* once we subscribe the products data arrived asynchronously.

   \*/

  public RegisterProduct(data) {

    this.data.AddProducts(data).subscribe(error => console.log('Something went wrong'));

    alert('Record Inserted');

  }

}

**4.Go to ‘demo.component.html’**

<!DOCTYPE html>

<html>

    <body>

        <div class="container">

            <div class="form-group">

                <h2>Add Products</h2>

                <form name="frmRegister" #frmRegister="ngForm" method="POST" (submit)="RegisterProduct(frmRegister.value)">

                    <dl>

                        <dt>Product ID</dt>

                        <dd>

                            <input type="number" name="product\_id" ngModel #product\_id="ngModel" class="form-control">

                        </dd>

                        <dt>Product Name</dt>

                        <dd>

                            <input type="text" name="product\_name" ngModel #product\_name="ngModel" class="form-control">

                        </dd>

                        <dt>Product Price</dt>

                        <dd>

                            <input type="number" name="price" ngModel #price="ngModel" class="form-control">

                        </dd>

                        <dt>Photo</dt>

                        <dd>

                            <input type="text" name="photo" ngModel #photo="ngModel" class="form-control">

                        </dd>

                        <dt>Manufactured</dt>

                        <dd>

                            <input type="date" name="mfd" ngModel #mfd="ngModel" class="form-control">

                        </dd>

                        <dt>Category ID</dt>

                        <dd>

                            <input type="number" name="category\_id" ngModel  #category\_id="ngModel" class="form-control">

                        </dd>

                        <dt>Category Name</dt>

                        <dd>

                            <input type="text" name="category" ngModel  #category="ngModel" class="form-control">

                        </dd>

                    </dl>

                    <button class="btn btn-primary">Submit</button>

                </form>

            </div>

            <div class="form-group">

                <h2>Products Data</h2>

                <table class="table table-hover" border="1px">

                    <thead class="thead-dark">

                        <tr>

                            <th>Name</th>

                            <th>Product ID</th>

                            <th>Price</th>

                            <th>Category Name</th>

                            <th>CategoryId</th>

                            <th>Manufactured</th>

                        </tr>

                    </thead>

                    <tbody>

                        <tr \*ngFor="let item of products">

                            <td>{{item.product\_name | uppercase}}</td>

                            <td>{{item.product\_id}}</td>

                            <td>{{item.price | currency:'&#8377;'}}</td>

                            <td>{{item.category| uppercase}}</td>

                            <td>{{item.category\_id}}</td>

                            <td>{{item.mfd | date}}</td>

                        </tr>

                    </tbody>

                </table>

            </div>

        </div>

    </body>

</html>