 Angular Framework provides its own HTTP client library which can be used to perform these types of operations. In the Javascript environment, the HTTP request is always an asynchronous operation which means it just sends the HTTP request to the API and it executes the next line of code without waiting for the response of the call. And when the API returns the response after some milliseconds or seconds or minutes it notified us and we can start the processing on the based of the response. So, in Angular there are two different ways to handle these types of operations – one is Observables and another is Promises. We will briefly discuss these two later in this article.

Basic Idea of Ajax Call

Ajax stands for Asynchronous JavaScript and XML. Actually, Ajax is not any programming language or framework or tool. Ajax is basically is a concept to client-side script that communicates between the server and the client machine to perform any type of operations. The best definition as per Ajax in the overview of SPA based application will be “Ajax is a method to exchange or pull data from a server ”. In a basic concept, Ajax always refers to the use of XmlHttpRequest objects to interact with a web server dynamically via JavaScript library. There are several benefits we can achieve by using Ajax in any web-based application like –

1. Ajax is used to perform callback operations, which make a perfect quick round trip to and from the server to send or retrieve data without post the entire page to the server. In this way, we can minimize the network utilization and application performance can be a boost.
2. Ajax always allows us to make asynchronous calls to a web server. In this way, the client browser avoids waiting for all data to arrive before allowing the user to act once more.
3. Since Ajax call, complete page postback is not required, so any Ajax-enabled web application will always be more responsive and user-friendly.
4. With the help of Ajax, we can improve the speed, performance, and usability of a web application.

About HttpClient

In Angular 6, Google first introduced a new easier way to operate with http requests with the new library called HttpClient. Angular provides the new library name so that it can avoid the new breaking changes made with the existing Http library. With the help of this new Http library, we can take advantage of the new functionality like the ability to listen for progress events, interceptors monitor to modify requests or response. Actually, the HTTP module always uses RxJS Observable-based API internally. For this reason, when we pass multiple ajax calls to the server using the HTTP module, it will return all responses as an observable object which we need to subscribe to one way or another. When we working using observable objects we need to remember some key points like,

* To receive the value as a response against any HTTP calls we need to subscribe to the observables. If we forget to subscribe, then nothing will happen.
* If we subscribe multiples times to the observables, then multiple HTTP requests will be triggered.
* By nature or type, every observable are single-value streams. So, if the HTTP request is successful then these observables will emit only one value and then complete.
* If the HTTP request fails, then these Observables emit an error.

To use the HttpClient in an Angular application, we need to import the HttpClientModule module into the root module of the application to use the Http Service.

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

**import** { NgModule, NO\_ERRORS\_SCHEMA } **from** '@angular/core';

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

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,HttpClientModule

],

bootstrap: [AppComponent],

schemas: [NO\_ERRORS\_SCHEMA]

})

**export** **class** AppModule { }

JavaScript

Copy

What are Observables?

Observable is one of the existing features used in the Angular Framework. But basically, it is not the Angular specific features, rather it is a proposed standard for managing any type of async data or operation which will be considered in the ES7 version. With the help of Observables, we can perform the continuous operations for multiple communication through which multiple values of data can be passed. So, by using Observables we can deal with array-like data operations to parse, modify and maintain data. That’s why the Angular framework use observables extensively.

Observables are actually a new primitive type that basically acts as a blueprint for how we want to create streams, subscribe to them, react to new values and combine streams together to build new streams.

What are Promises?

In our application, when we execute any task synchronously, we wait for it to finish before moves to the next line of code. But if we execute the same task asynchronously, the program moves to the next line of code before the task finished its job, maybe finishing off that task may require a few seconds or more. So in a simple word, we can assume that synchronous programming like waiting in line or queue and asynchronously programming like taking a ticket and don’t wait in a queue. Since we have a ticket we can do some other tasks and when that particular task completed it will notify us against that ticket. One of the easiest ways to program asynchronously is to use callbacks functions. We can pass to an asynchronous function a function as a parameter which will be invoked when the task is completed.

**function** doAsyncTask(cb) {

setTimeout(() => {

console.log("Async Task Calling By Callback Function");

cb();

}, 1000);

}

doAsyncTask(() => console.log("Callback Function Called"));

JavaScript

Copy

In the above example, doAsyncTask function will be fired an asynchronous task and returns immediately. When the async task completes, a function will be called. It is a callback function, as initialize by short name cb, because of it calls-you-back.

ES6 provides us an alternative mechanism built into the language called a promise. So, the promise is a placeholder for future value. It basically serves the same functionality as callbacks but it provides a nice and organized syntax which provides us an easier way to handle errors. We can create an instance of a promise by called new on the Promise class like below,

**var** promise = **new** Promise((resolve, reject) => {

});

JavaScript

Copy

We always pass an inner function with two arguments (resolve, reject). As we define the function, we can call or named this argument whenever we want, but the standard convention is to call them as resolve and reject. Actually, resolve and reject are in fact functions themselves. Within this inner function, we actually perform our asynchronous processing and then when we are ready we can call the resolve() as below,

**var** promise = **new** Promise((resolve, reject) => {

setTimeout(() => {

console.log("Async Work Complete");

resolve();

}, 1000);

});

JavaScript

Copy

So, if we want to use the promise in our callback function example, it will be like this,

**let** error = true;

**function** doAsyncTask() {

**var** promise = **new** Promise((resolve, reject) => {

setTimeout(() => {

console.log("Async Work Complete");

**if** (error) {

reject();

}

**else**

{

resolve();

}

}, 1000);

});

**return** promise;

}

JavaScript

Copy

Observables vs Promises

When we use HttpClient for the Ajax calls in Angular Framework, we can use both implementations (i.e. observables or promises) which provides an easy way of API operation for handling requests or receiving a response. But still, there are some key differences that make Observables a superior alternative compared to the promises,

* Promises can only accept one value at a time unless we compose multiple promises execution plan (Eg: $q.all).
* Promises can’t be canceled if required.

In Angular 8, it mainly introduces reactive programming concepts totally based on the observables for dealing with the asynchronous processing of data. In earlier versions of Angular (i.e. Angular 1.x), we normally use promises to handle the asynchronous processing related to the ajax call. But still, in Angular 8, we can still use the promises for the same purpose in place of observables. The main objective of reactive programming is the observable element which mainly used to the model or variable which can be observed. Basically, at the normal concept, promises and observables both seem to very similar to each other. Both observables and promises are used to execute asynchronous ajax call along with callback functions for both successful and error responses and also notify or inform us when the result is there.

How to Define Observables

So before we start discussing HTTP verbs, first we need to understand how we can define observables objects. So, to create an observable, we need to use the create method of the Observable object. We need to provide a function object as a parameter in which we need to initialize the observable processing. This function can return a function to cancel the observable.

**var** observable = Observable.create((observer) => {

setTimeout(() => {

observer.next('some event');

}, 500);

});

JavaScript

Copy

Just like promises, observables can also provide multiple notifications with the help of different methods as below,

* next – It emits an event and it can be executed several times.
* error – It mainly used to throw an error. Once it is executed the stream will be a break. It means error callback will be called immediately and no more events or steps can be executed.
* complete – It is used to indicate that the observable as completed. After these events, no events will be executed.

With the help of observables, we can register the callback for previously described notifications. The subscribe method tackles this issue. It can accept three callback function as parameters,

* onNext – This callback will be called when an event is triggered.
* onError – This callback will be called when an error is thrown.
* onCompleted – This callback will be called when an observable complete.

Http Verbs

In the HTTP protocol, a set of verbs has been described for each type of operation or URL. For any URL, we can perform a GET, PUT, POST, DELETE, HEAD, and OPTIONS request. In this section, we will discuss how to execute the API URL for the mentioned verbs. For that purpose, we assume that we perform the dependency injection for HttpClient in the constructor level of the component as below,

constructor(**private** http: HttpClient) {

}

JavaScript

Copy

**GET**

To perform a GET request, we need to simply call the get function on our http client. This will returns an observable which needs to subscribe to receive the data or response from the server-side.

performGET() {

console.log("GET");

**let** url = `<http://localhost/get>`;

**this**.http.get(url).subscribe(res => console.log(res.text()));

}

JavaScript

Copy

**DELETE**

To Perform a DELETE request we just need to call the delete function of the http client. The format or the function is quite similar to the get function as above. But we can pass query params within this function.

performDELETE() {

console.log("DELETE");

**let** url = `<http://localhost/delete>`;

**let** search = **new** URLSearchParams();

search.set('foo', 'moo');

search.set('limit', 25);

**this**.http.delete(url, {search}).subscribe(res => console.log(res.json()));

}

JavaScript

Copy

**POST**

To perform a POST request, we call the post function of the http client. The format of the post function is the same as getting a function. But we normally perform the POST request to pass the data as well as to the server. So the in the second parameter in case of post, need to pass an object in place of query parameters which will be passed as the payload for the request.

performPOST() {

console.log("POST");

**let** url = `<http://localhost/post>`;

**this**.http.post(url, {moo:"foo",goo:"loo"}).subscribe(res =>

console.log(res.json()));

}

JavaScript

Copy

**PUT**

To perform a PUT request we just need to call the put function. It will work exactly the same as the post function.

performPUT() {

console.log("PUT");

**let** url = `<http://localhost/put>`;

**let** search = **new** URLSearchParams();

search.set('foo', 'moo');

search.set('limit', 25);

**this**.http.put(url, {moo:"foo",goo:"loo"}, {search}).subscribe(res =>

console.log(res.json()));

}

JavaScript

Copy

Handling Errors

When we perform an ajax call from our Angular application,an error or exceptions on the server-side may occur. In that case, it will return that error or exception message to the angular application. So we need to know how we can receive that exception message so that we can display those messages in the front of the user or store in our error logs. So, whether we are handling the response as an Observable or as a Promise, we can always implement error handling function as the second param of the http method. In case of a promise, it looks like as below,

performGETAsPromiseError() {

console.log("GET AS PROMISE ERROR");

**let** url = `<http://localhost/post>`;

**this**.http.get(url)

.toPromise()

.then(

res => console.log(res.json()),

msg => console.error(`Error: ${msg.status} ${msg.statusText}`) ①

);

}

JavaScript

Copy

In the case of observables, the same will look the below,

performGETAsError() {

console.log("GET AS OBSERVABLES ERROR");

**let** url = `<http://localhost/post>`;

**this**.http.get(url).subscribe(

res => console.log(res.json()),

msg => console.error(`Error: ${msg.status} ${msg.statusText}`)

);

}

JavaScript

Copy

Headers

In Ajax call, another important part is HTTP headers through which we can pass several configuration-related information to the server-side. By nature, HTTP headers are types of meta-data which normally attached by the browser when we call any HTTP request. Sometimes we need to pass some extra customer headers with our ajax request and we can do that very easily with the help of Angular http client. For this purpose, we first need to import two helper classes from the http module in our component or service.

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

JavaScript

Copy

We first need to create an instance of the HttpHeaders and then we can define our mentioned headers in that instance as below –

**const** httpOptions = {

headers: **new** HttpHeaders({

'Content-Type': 'application/json; charset=utf-8'

})

};

JavaScript

Copy

After it, we just need to pass the httpHeader to the proper HTTP verb function.

performPOST() {

console.log("POST");

**let** url = `<http://localhost/post>`;

**this**.http.post(url, {moo:"foo",goo:"loo"},httpOptions).subscribe(res =>

console.log(res.json()));

}

JavaScript

Copy

Demo 1 - Http Core API Samples

Now in this demo, we will demonstrate how to use ajax call in Angular. For this purpose, we will develop a complete flow related to the employee where we can perform employee add, display existing records, edit data or delete employee data. In this sample demo, we create three different components as below –

* EmployeeListComponent – For display list employee of information
* AddEmployeeComponent – For Add New Employee Details
* UpdateEmployeeComponent – For update existing Employee Details

**app.component.employeelist.ts**

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

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

@Component({

selector: 'employee-list',

templateUrl: 'app.component.employeelist.html'

})

**export** **class** EmployeeListComponent **implements** OnInit {

**public** data: any = [];

**public** showDetails: boolean = true;

**public** showEmployee: boolean = false;

**public** editEmployee: boolean = false;

**private** \_selectedData: any;

**private** \_deletedData: any;

constructor(**private** http: HttpClient) {

}

ngOnInit(): **void** {

}

ngAfterViewInit(): **void** {

**this**.loadData();

}

**private** loadData(): **void** {

**let** self = **this**;

**this**.http.get("<http://localhost:81/SampleAPI/employee/getemployee>")

.subscribe((res: Response) => {

self.data = res;

});

}

**private** addEmployee(): **void** {

**this**.showDetails = false;

**this**.showEmployee = true;

}

**private** onHide(args: boolean): **void** {

**this**.showDetails = !args;

**this**.showEmployee = args;

**this**.editEmployee = args;

**this**.loadData();

}

**private** onUpdateData(item: any): **void** {

**this**.\_selectedData = item;

**this**.\_selectedData.DOB = **new** Date(**this**.\_selectedData.DOB);

**this**.\_selectedData.DOJ = **new** Date(**this**.\_selectedData.DOJ);

**this**.showDetails = false;

**this**.editEmployee = true;

}

**private** onDeleteData(item: any): **void** {

**this**.\_deletedData = item;

**if** (confirm("Do you want to Delete Record Permanently?")) {

**let** self = **this**;

**const** httpOptions = {

headers: **new** HttpHeaders({

'Content-Type': 'application/json; charset=utf-8'

})

};

**this**.http.post("<http://localhost:81/SampleAPI/employee/DeleteEmployee>", **this**.\_deletedData, httpOptions)

.subscribe((res: Response) => {

self.loadData();

});

}

}

}

JavaScript

Copy

**app.component.employeelist.html**

<div **class**="panel">

<h3>HTTP Module Sample - Add and Fetch Data</h3>

<div **class**="panel-body container-fluid" \*ngIf="showDetails">

<div **class**="row row-lg">

<table **class**="table" style="width:100%;">

<thead>

<tr>

<th **class**="cell-100">Srl No</th>

<th **class**="cell-300">Alias</th>

<th **class**="cell-300">Employee Name</th>

<th **class**="cell-300">Date **of** Birth</th>

<th **class**="cell-300">Join Date</th>

<th **class**="cell-300">Department</th>

<th **class**="cell-300">Designation</th>

<th **class**="cell-300">Salary</th>

<th **class**="cell-180"></th>

</tr>

</thead>

<tbody>

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

<td **class**="cell-100">{{item.Id}}</td>

<td **class**="cell-300">{{item.Code}}</td>

<td **class**="cell-300">{{item.Name}}</td>

<td **class**="cell-300">{{item.DOB | date :'shortDate'}}</td>

<td **class**="cell-300">{{item.DOJ | date :'mediumDate'}}</td>

<td **class**="cell-300">{{item.Department}}</td>

<td **class**="cell-300">{{item.Designation}}</td>

<td **class**="cell-300">{{item.Salary |currency:'INR':true}}</td>

<td **class**="cell-180">

<a (click)="onUpdateData(item);" style="cursor:pointer;">

<span **class**="badge badge-primary">Edit</span>

</a>

<a (click)="onDeleteData(item);" style="cursor:pointer;">

<span **class**="badge badge-danger">Delete</span>

</a>

</td>

</tr>

</tbody>

</table>

<p>

<button **class**="btn btn-primary" (click)="addEmployee()">

Add Employee

</button>

</p>

</div>

</div>

<div **class**="panel-body container-fluid" \*ngIf="showEmployee">

<employee-add (onHide)="onHide($event);"></employee-add>

</div>

<div **class**="panel-body container-fluid" \*ngIf="editEmployee">

<employee-update [source]="\_selectedData" (onHide)="onHide($event);"></employee-update>

</div>

</div>

JavaScript

Copy

**app.component.employeeadd.ts**

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

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

@Component({

selector: 'employee-add',

templateUrl: 'app.component.employeeadd.html'

})

**export** **class** AddEmployeeComponent **implements** OnInit {

**public** \_model: any = {};

@Output() **private** onHide: EventEmitter<boolean> = **new** EventEmitter<boolean>();

constructor(**private** http: HttpClient) {

}

ngOnInit(): **void** {

}

**public** onCancel(): **void** {

**this**.\_model = {};

**this**.onHide.emit(false);

}

**public** submit(): **void** {

**if** (**this**.validate()) {

**let** self = **this**;

**const** httpOptions = {

headers: **new** HttpHeaders({

'Content-Type': 'application/json; charset=utf-8'

})

};

**this**.http.post("<http://localhost:81/SampleAPI/employee/AddEmployee>", **this**.\_model, httpOptions)

.subscribe((res: Response) => {

self.onCancel();

});

}

}

**private** reset(): **void** {

**this**.\_model = {};

}

**private** validate(): boolean {

**let** status: boolean = true;

**if** (**typeof** (**this**.\_model.code) === "undefined") {

alert('Alias is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.name) === "undefined") {

alert('Name is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.dob) === "undefined") {

alert('dob is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.doj) === "undefined") {

alert('DOJ is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.department) === "undefined") {

alert('Department is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.designation) === "undefined") {

alert('Designation is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.salary) === "undefined") {

alert('Salary is Blank');

status = false;

**return**;

}

**return** status;

}

}

JavaScript

Copy

**app.component.employeeadd.html**

<div class="row row-lg">

<h4>Provide Employee Details</h4>

<table class="table">

<tr>

<td>Employee Code</td>

<td><input type="text" [(ngModel)]="\_model.code" /></td>

</tr>

<tr>

<td>Employee Name</td>

<td><input type="text" [(ngModel)]="\_model.name" /></td>

</tr>

<tr>

<td>Date of Birth</td>

<td><input type="date" [(ngModel)]="\_model.dob" /></td>

</tr>

<tr>

<td>Date of Join</td>

<td><input type="date" [(ngModel)]="\_model.doj" /></td>

</tr>

<tr>

<td>Department</td>

<td><input type="text" [(ngModel)]="\_model.department" /></td>

</tr>

<tr>

<td>Designation</td>

<td><input type="text" [(ngModel)]="\_model.designation" /></td>

</tr>

<tr>

<td>Salary</td>

<td><input type="number" [(ngModel)]="\_model.salary" /></td>

</tr>

<tr>

<td></td>

<td>

<input type="button" value="Submit" (click)="submit()" />

<input type="button" value="Cancel" (click)="onCancel()" />

</td>

</tr>

</table>

</div>

Markup

Copy

**app.component.employeeupdate.ts**

**import** { Component, OnInit, EventEmitter, Output, Input } **from** '@angular/core';

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

@Component({

selector: 'employee-update',

templateUrl: 'app.component.employeeupdate.html'

})

**export** **class** UpdateEmployeeComponent **implements** OnInit {

**public** \_model: any = {};

@Input()

**set** source(data: any) {

**this**.\_model = data;

}

**get** source() {

**return** **this**.\_model;

}

@Output() **private** onHide: EventEmitter<boolean> = **new** EventEmitter<boolean>();

constructor(**private** http: HttpClient) {

}

ngOnInit(): **void** {

**if** (**this**.\_model == **undefined**) {

**this**.\_model = **this**.source;

}

}

**public** onCancel(): **void** {

**this**.\_model = {};

**this**.onHide.emit(false);

}

**public** onUpdate(): **void** {

**if** (**this**.validate()) {

**let** self = **this**;

**const** httpOptions = {

headers: **new** HttpHeaders({

'Content-Type': 'application/json; charset=utf-8'

})

};

**this**.http.put("<http://localhost:81/SampleAPI/employee/UpdateEmployee>", **this**.\_model, httpOptions)

.subscribe((res: Response) => {

self.onCancel();

});

}

}

**private** reset(): **void** {

**this**.\_model = {};

}

**private** validate(): boolean {

**let** status: boolean = true;

**if** (**typeof** (**this**.\_model.Code) === "undefined") {

alert('Alias is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.Name) === "undefined") {

alert('Name is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.Department) === "undefined") {

alert('Department is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.Designation) === "undefined") {

alert('Designation is Blank');

status = false;

**return**;

}

**else** **if** (**typeof** (**this**.\_model.Salary) === "undefined") {

alert('Salary is Blank');

status = false;

**return**;

}

**return** status;

}

**private** parseDate(dateString: string): Date {

**if** (dateString) {

**return** **new** Date(dateString);

} **else** {

**return** **null**;

}

}

}

JavaScript

Copy

**app.component.employeeupdate.html**

<div **class**="row row-lg">

<h4>Provide Employee Details</h4>

<table **class**="table">

<tr>

<td>Employee Code</td>

<td><input type="text" [(ngModel)]="\_model.Code" /></td>

</tr>

<tr>

<td>Employee Name</td>

<td><input type="text" [(ngModel)]="\_model.Name" /></td>

</tr>

<tr>

<td>Date **of** Birth</td>

<td><input type="text" [(ngModel)]="\_model.DOB" readonly /></td>

</tr>

<tr>

<td>Date **of** Join</td>

<td><input type="text" [(ngModel)]="\_model.DOJ" readonly /></td>

</tr>

<tr>

<td>Department</td>

<td><input type="text" [(ngModel)]="\_model.Department" /></td>

</tr>

<tr>

<td>Designation</td>

<td><input type="text" [(ngModel)]="\_model.Designation" /></td>

</tr>

<tr>

<td>Salary</td>

<td><input type="number" [(ngModel)]="\_model.Salary" /></td>

</tr>

<tr>

<td></td>

<td>

<input type="button" value="Update" (click)="onUpdate()" />

<input type="button" value="Cancel" (click)="onCancel()" />

</td>

</tr>

</table>

</div>

JavaScript

Copy

**app.component.ts**

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

@Component({

selector: 'app-root',

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

styleUrls : ['./custom.css']

})

**export** **class** AppComponent **implements** OnInit {

ngOnInit(){

}

}

JavaScript

Copy

**app.component.html**

<div style="padding-left: 20px;padding-top: 20px; width: 1000px;">

<div class="row">

<div class="col-xs-12">

<employee-list></employee-list>

</div>

</div>

</div>

Markup

Copy

**app.module.ts**

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

import { NgModule, NO\_ERRORS\_SCHEMA } from '@angular/core';

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

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

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

import { EmployeeListComponent } from './app.component.employeelist';

import { AddEmployeeComponent } from './app.component.employeeadd';

import { UpdateEmployeeComponent } from './app.component.employeeupdate';

@NgModule({

declarations: [

AppComponent,EmployeeListComponent,AddEmployeeComponent, UpdateEmployeeComponent

],

imports: [

BrowserModule,HttpClientModule, FormsModule, ReactiveFormsModule

],

bootstrap: [AppComponent],

schemas: [NO\_ERRORS\_SCHEMA]

})

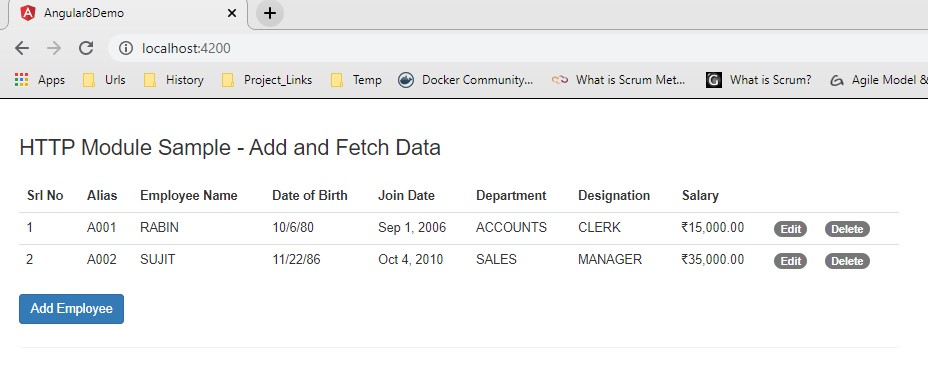
export class AppModule { }

JavaScript

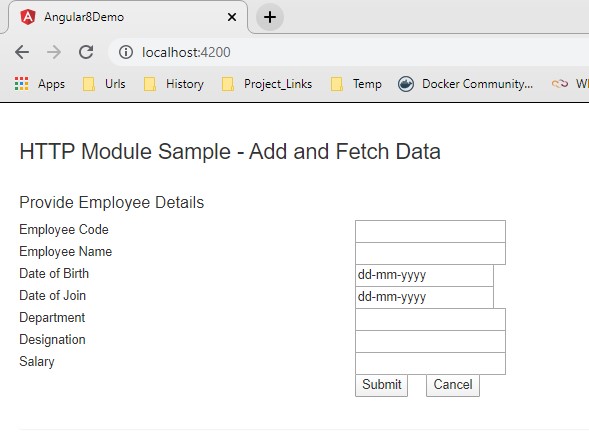
Copy

Now run the demo in the browser,

**Employee List**



**New Employee Add**



**Update Employee Info**

