Angular 7 Project With ASP.NET CORE APIs (Gym Project)

Saineshwar Bageri Updated date Feb 06, 2019

169.5k 121 61

DatabaseScript.rar

Download Free .NET & JAVA Files API

Link to download the project source code here.

Let's see what this project is all about. This project is a basic gym project which has 2 modules in it.

1. Admin end
2. User end

Admin end

Let's start with the Admin end first. In this part, the admin has all the rights of the application's master, such as adding Users, Role, Scheme, Plan, and various reports, such as month-wise and year-wise income reports, all member reports, and also, it has a renewal report which shows how many renewals are there for the period the admin chooses.

User end

If you look at the User end, a user is a person who does the work of registering new members and collecting payments. The user has limited access, such as a user can register a new member and renew membership and see payment details of the member along with renewal date.

The project has 3 parts.

1. Angular CLI which is on top of node.js
2. NET Core for APIs
3. SQL Server for database parts

Platform Used

Angular Version 7 is used.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Microsoft Visual Studio Community 2017

The link to download Microsoft Visual Studio Community 2017 is here.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Microsoft SQL Server 2012

Link to download SQL Server Express is here.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Visual Studio Code

Link to download Visual Studio code is here.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

JWT Token for Authentication of APIs

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Image is referenced from here.

External packages which are used in .NET Core Project

1. JWT Token for Authentication of APIS
2. Dapper ORM
3. AutoMapper
4. Linq.Dynamic.Core

External packages which are used in Angular Project

1. @angular/material
2. @ngx-bootstrap/datepicker

Let’s start with database tables first.

Database

For this application, I have used SQL Server database. You can download the entire script of this database and use it.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Let us explore the tables of this database one by one so that you will have a clear idea of which table is used for which purpose.

Fiscal year

This table contains the financial year details.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Member Details

Whenever a new member takes a subscription, all member details are stored in this database.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Payment Details

Whenever a new member takes a subscription, all payment details are stored in this database.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

PeriodTB

This table contains periods, such as 3 months, 6 months, and 1 year.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

SchemeMaster

This table contains Schemes.

Example: - GYM+CARDIO1, GYM

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Plan Master

This table contains plans along with the plan amount and Service Tax. For one scheme, there can be multiple plans.

Example: - plan names are Quarterly, Half Yearly, Yearly, Men Special Plan, Women Special Plan.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Role

This table contains Roles.

Example: - Admin, User.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

UsersInRoles

This table contains UserID and roleID means a role assigned to the user is stored in this table.

Example: - Admin, User.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Users

This table contains the user login credentials.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

After understanding the table structure, let’s move forward to understand the API application structure.

API Project Structure

In this part, we are going to have a look at the API project in detail. API Project Name is WebGYM which is developed in ASP.NET CORE Framework version 2.1.

The project structure is a simple and clean repository pattern with ASP.NET Core built-in dependency injection framework.

Let’s start from Model class library. This class library contains all Models which are used in the application and which are similar to database entities. In the same way, you can see ViewModels; these models are used for taking requests and displaying the response of APIs. Further, we are having a view into Interface Class library which contains all the interfaces of the application used for loose coupling of applications. These interfaces are implemented in the concrete class library. This layer directly interacts with your database. In this project, I have used Entity Framework Core and Dapper ORM to access the database.

Added ASP.NET MVC Core Web Project.

WebGYM

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Next, we are going to have a look at WebGYM.Models Class Library.

WebGYM.Models

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Next, we are going to have a look at WebGYM.ViewModels Class Library.

WebGYM.ViewModels

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

After having a look at View Models Class library, next, let's have a look into Interface Class library.

This Class Library contains all Interfaces in which we have declared methods. This interface is going to implement a concrete class (WebGYM.Concrete).

WebGYM.Interface

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

WebGYM.Concrete

The WebGYM.Concrete Class Library contains all the classes which are using Entity Framework Core ORM for accessing the database including the DatabaseContext class, the main class for interacting with the database. I have also used Dapper ORMrm for some database operations.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

After viewing the class library, next, we are going to view the API Controller Structure.

Controllers

The Controllers folder contains all API controllers created in this application.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

DbContext (we are using Entity Framework core in this project)

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Appsettings.json file

In the appsettings.json file, we store all application settings in a key-value pair. Here, we have stored the connection string of database.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Setting Dependence injection in Startup.cs class

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Filters and Encryption library

For the authentication of APIs, I have Used JSON Web Tokens and for error logging, I have created "CustomExceptionFilterAttribute" which is registered as a filter in ConfigureServices Method in start-up Class.

You can see the error log folder which contains text files where generated errors in the application are stored. For the encryption of password, I have used Advanced Encryption

Standard (AES) Algorithm.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

The above shows how the directory and folder structures look. Now, let's check out the application screens.

After having a view of completed API application screens, next, we are going to view Angular Application Project Structure.

Angular Application Project Structure

Below is a complete view of Angular Application project structure. If you see the below snapshot you will see all folders with proper naming which tells which folder contains which Component in it.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Next let’s view some folders to know how we are maintaining Services, Models, Component and Html Views.

Let’s Expand RoleMaster Folder. If you see in that folder we have 2 folders; one is Models folder and another is Service Folder.

Model Folder contains Class which is used with Views and API to Post, Get Request and Response.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Let’s have look at the completed flow of role module and you will get a good idea of how the application is designed.

app.RoleModel.ts

This is a model which is used on role HTML template for model binding.

```
export class RoleModel
{
    public RoleName: string;
    public Status: boolean;
    public RoleId : number;
}
```

app.Role.component.ts

Role Component, which is going to render role template.

```
import { Component } from '@angular/core';
import { RoleModel } from './Models/app.RoleModel';
import { RoleService } from './Services/app.role.Service';
import { Router } from '@angular/router';

@Component({
    templateUrl : './app.Role.html',
    styleUrls: ['../Content/vendor/bootstrap/css/bootstrap.min.css',
    '../Content/vendor/metisMenu/metisMenu.min.css',
    '../Content/dist/css/sb-admin-2.css',
    '../Content/vendor/font-awesome/css/font-awesome.min.css'
]
})

export class RoleComponent
{
    private _roleService;
    RoleModel : RoleModel = new RoleModel();
    output: any;

    constructor(private _Route: Router, roleService :RoleService ){
        this._roleService = roleService;
    }

    onSubmit()
    {
        this._roleService.AddRole(this.RoleModel).subscribe(
            response => {
            this.output = response
            if (this.output.StatusCode == "409") {
                alert('Role Already Exists');
            }
            else if (this.output.StatusCode == "200") {
                alert('Role Saved Successfully');
                this._Route.navigate(['/Role/All']);
            }
            else {
                alert('Something Went Wrong');
            }
        });
    }

}
```

Moving forward you can see the Services folder; in this folder we are consuming Web API Services.

Role.Service

In this class, we use HttpClient to call API services and along with that we also send JWT token which is used for authentication. If you see Role.Service class you will find all kind of Http Method get, post, put, delete.

```
import { Injectable } from '@angular/core';
import { Observable, throwError } from 'rxjs'
import { catchError, tap } from 'rxjs/operators'
import { HttpClient, HttpErrorResponse, HttpHeaders, HttpResponse } from '@a
import { RoleModel } from '../Models/app.RoleModel';
import { environment } from 'src/app/Shared/environment';
@Injectable({
    providedIn: 'root'
})

export class RoleService {

    private data: any;
    private apiUrl = environment.apiEndpoint + "/api/CreateRole/";
    token: any;
    username: any;

    constructor(private http: HttpClient) {
        this.data = JSON.parse(localStorage.getItem('AdminUser'));
        this.token = this.data.token;
    }

    public AddRole(rolemodel: RoleModel) {
        let headers = new HttpHeaders({ 'Content-
Type': 'application/json' });
        headers = headers.append('Authorization', 'Bearer ' + `${this.token
        return this.http.post<any>
(this.apiUrl, rolemodel, { headers: headers })
            .pipe(
                catchError(this.handleError)
            );
    }

    // Get All Role
    public GetAllRole() {
        let headers = new HttpHeaders({ 'Content-
Type': 'application/json' });
        headers = headers.append('Authorization', 'Bearer ' + `${this.token
        return this.http.get<RoleModel[]>
(this.apiUrl, { headers: headers }).pipe(tap(data => data),
            catchError(this.handleError)
        );
    }

    // Get All Role By ID
    public GetRoleById(RoleId) {
        var editUrl = this.apiUrl + '/' + RoleId;
        let headers = new HttpHeaders({ 'Content-
Type': 'application/json' });
        headers = headers.append('Authorization', 'Bearer ' + `${this.token
        return this.http.get<RoleModel>
(editUrl, { headers: headers }).pipe(tap(data => data),
            catchError(this.handleError)
        );
    }

    // Update Role
    public UpdateRole(rolemodel: RoleModel) {
        var putUrl = this.apiUrl + '/' + rolemodel.RoleId;
        let headers = new HttpHeaders({ 'Content-
Type': 'application/json' });
        headers = headers.append('Authorization', 'Bearer ' + `${this.token
        return this.http.put<any>
(putUrl, rolemodel, { headers: headers })
            .pipe(
                catchError(this.handleError)
            );
    }

    public DeleteRole(RoleId) {
        var deleteUrl = this.apiUrl + '/' + RoleId;
        let headers = new HttpHeaders({ 'Content-
Type': 'application/json' });
        headers = headers.append('Authorization', 'Bearer ' + `${this.token
        return this.http.delete<any>(deleteUrl, { headers: headers })
            .pipe(
                catchError(this.handleError)
            );
    }

    private handleError(error: HttpErrorResponse) {
        if (error.error instanceof ErrorEvent) {
            // A client-side or network error occurred. Handle it accordingly.
            console.error('An error occurred:', error.error.message);
        } else {
            // The backend returned an unsuccessful response code.
            // The response body may contain clues as to what went wrong,
            console.error(`Backend returned code ${error.status}, ` + `body
        }
        // return an observable with a user-facing error message
        return throwError('Something bad happened; please try again later.'
    };
}
```

After we have viewed Services folder, next let's view Html Templates of Role Module.

app.Role.html template

app.Role.html Template which is been used as templateUrl in app.Role.component.ts class.

```
<h4>Add Role</h4>
<hr>
<div class="panel panel-default">
    <div class="panel-heading">Add Role</div>
    <div class="panel-body">
        <form #f="ngForm" novalidate (ngSubmit)="onSubmit()">
            <div class="row">
                <div class="col-md-4">
                    <label for="name">RoleName</label>
                    <input type="text" class="form-control" name="RoleName" [(ngModel)]="RoleModel.RoleName" maxlength="50"
                        #refRoleName="ngModel" id="RoleName" required>
                    <div *ngIf="!refRoleName.valid  && (refRoleName.dirty |
danger">
                        <div [hidden]="!refRoleName.errors.required">
                            RoleName is required
                        </div>
                    </div>
                </div>
                <div class="col-md-4">
                    <label for="name">Status</label>
                    <input type="checkbox" name="Status" [(ngModel)]="RoleM
                        id="Status" required>
                    <div *ngIf="!refStatus.valid  && (refStatus.dirty || re
danger">
                        <div [hidden]="!refStatus.errors.required">
                            Status is required
                        </div>
                    </div>
                </div>
                <div class="col-md-4">
                    <button type="submit" style="margin-top: 10px" [disabled]="!f.form.valid" class="btn btn-success">Submit</button>
```

```
                        <a style="margin-left: 10px; margin-top:7px;" class="btn btn-info" [routerLink]="['/Role/All']">
                            All Roles </a>
                    </div>

                </div>
            </form>

        </div>
</div>
```

package.json

Package json contains all packages which are used in this project.

```
{
  "name": "gym-project",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^7.0.4",
    "@angular/cdk": "^7.1.1",
    "@angular/common": "~7.0.0",
    "@angular/compiler": "~7.0.0",
    "@angular/core": "~7.0.0",
    "@angular/forms": "~7.0.0",
    "@angular/http": "~7.0.0",
    "@angular/material": "^7.1.1",
    "@angular/platform-browser": "~7.0.0",
    "@angular/platform-browser-dynamic": "~7.0.0",
    "@angular/router": "~7.0.0",
    "core-js": "^2.5.4",
    "hammerjs": "^2.0.8",
    "html2canvas": "^1.0.0-alpha.12",
    "jspdf": "^1.5.2",
    "ngx-bootstrap": "^3.1.2",
    "rxjs": "~6.3.3",
    "xlsx": "^0.14.1",
    "zone.js": "~0.8.26"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.10.0",
    "@angular/cli": "~7.0.6",
    "@angular/compiler-cli": "~7.0.0",
    "@angular/language-service": "~7.0.0",
    "@types/node": "~8.9.4",
    "@types/jasmine": "~2.8.8",
    "@types/jasminewd2": "~2.0.3",
    "codelyzer": "~4.5.0",
    "jasmine-core": "~2.99.1",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~3.0.0",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~1.1.2",
    "karma-jasmine-html-reporter": "^0.2.2",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "~5.11.0",
    "typescript": "~3.1.6"
  }
}
```

After we have completed single module next, we are going to have a look on all admin screens of applications.

Admin Application Screens

Login is common for Admin and User.

We are going to log into the application with admin credentials.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

After logging into the application the first screen which appears is your dashboard.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

After logging into the application, we are first going to view Add Scheme.

Add Scheme

In this part, we are going to Add a New Scheme.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Scheme

Displays all schemes which are added.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Edit Scheme

In this part, we can just edit the status of the scheme. If you want to change the name then you can delete and add a new scheme.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Add Plan

In this part, we are going to add the plan and while adding we are going to select Period and Scheme. Also enter Plan amount and required service tax for it.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Plan

Displays all Plans which are added.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Edit Plan

In this part, we can Edit Plan which we have added.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Add Role

In this part, we are going to add new roles

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Role

Displays all roles which are added.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Edit Role

In this part, we are going to edit and update role status.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Create New User

In this part, we are going to add new user/admin who is going to use this application.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Users

Displays all Users which are added.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Edit Users

In this part, we can Edit User Details which we have added such as User Email or Phone and Password.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Assign / Remove Role

In this part, we are going to assign and remove roles to the user which we have created.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Assigned Roles

In this part, we are going to assign and remove roles to the user which we have created.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Member Report

In this part, admin can export all member data for reporting.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Exported all Member Report in Excel

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Year wise Collection Report

In this part, the admin will select the year and click on the submit button to generate a report and also can export it.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Exported Year Wise Collection Report in Excel

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Month-wise Collection Report

In this part, the admin will select Month and click on the submit button to generate and export report.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Exported Month Wise Collection Report in Excel

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Renewal Report

In this part the admin just needs to choose dates and submit to get all renewal Member details.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Exported Renewal Report in Excel

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

User Application Screens

We are going to log in to the application with User credentials.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

User Dashboard

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Add Member

From this view, we are going to register new members and also, we will select a scheme and plan here. According to the plan we choose it will show the amount the member needs to pay.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Member

Display all Members who got a membership.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Edit Member

In the edit part, we can edit some Member details but we cannot change the plan of members. If something is wrong then we need to delete the member again and create a new one.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

All Payment Details

All the payment details are displayed in this View.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Receipt

In this part, we need to provide a receipt to a member who registered for the membership. On-grid, we have Receipt button. Just clicking on it will show you a receipt of the particular member. On the receipt page, we have a Print button to generate the PDF.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

PDF format of the receipt

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Renewal of Membership

In this part, we are going to renew the membership of members. The page is simple; we just need to search the member name which is an autocomplete feature and after choosing a name, it will get all his/her details. Here, we can change the Member Scheme and Plan if he wants and we can choose New Membership Date for renewal.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

After searching the Member, click on Member name and all the details are displayed.

Angular 7 Project With ASP.NET CORE APIS (Gym Project)

Choose a date to renew and click on the Renew button to renew membership.

Wow, finally, we have completely gone through the application. You can download the entire application and database and play with it. Also, try to make this application better by contributing to it on GitHub.

Angular | Angular 7 | Angular 7 Project | ASP.NET CORE | ASP.NET CORE APIS | Gym Project

Next Recommended Reading

Angular 5 App With ASP.NET Core 2.0 Web API

Saineshwar Bageri TOP 100

Senior Technical lead and Microsoft MVP from C# Corner MVP working on .Net Web Technology (ASP.NET, ASP.NET Core,.Net Core, C#, Sqlserver, MVC, Windows, Console Application, javascript, jquery, json, ORM Dap... Read more

https://tutexchange.com/

96 22.2m 5 3

View Previous Comments

61 121

Type your comment here and press Enter Key (Minimum 10 characters)

Follow Comments

And Sir, what are the processes of publish these types of projects ASP.NET CORE with SQL, and Angular- am new and am really swimming into this. like am trying to publish similar project but am keep on getting error: uncaught typeerror: cannot read property 'length' of null at object.acquirecontext (chart.min.js:7) at ni.construct (chart.min.js:7) at new ni (chart.min.js:7) maybe someone can assist me thanks in advance.

Francis Banda Jul 09, 2021

1904 229 0 0 0 Reply

You are still a big boss!!!! thank you

Francis Banda Jul 09, 2021

1904 229 0 0 0 Reply

I need the SQL DB file at ashiqur08@gmail.com. I have made a payment from SBI account today at 14052021 at 3.20 am and the amount is Rs 200.

ashiqur rahaman May 13, 2021

2116 17 0 0 0 Reply

Excellent article. But the Database script is not open, it showing an error to open

Janardhana Rao Kotagiri Feb 23, 2021

2028 105 0 0 0 Reply

You are superb. Excellent article. API is awesome.

jubula samal Feb 19, 2021

1559 575 0 0 1 Reply

Find me on https://tutexchange.com/

Saineshwar Bageri Feb 20, 2021

96 23.4k 22.2m 0

One of the best article i have been seen. Could you please share the DB script to my email slyess85@gmail.com

lyes yacine Dec 19, 2020

1905 228 0 1 0 Reply

Could you please share the DB script to my email: tranvandang007@gmail.com. Thanks you so much!

Dang tv Aug 29, 2020

2077 56 0 0 0 Reply

It's so useful article. Thank you so much for your sharing..............

Hamid Khan Aug 25, 2020

324 7.4k 1.5m 0 0 Reply

Hi Sir how can i get all codes which will help me .Desidesign86@gmail.com plse it will help

zohra khan Jul 12, 2020

1986 147 0 0 1 Reply

Everything is available in this article all code you need just read the article properly.

Saineshwar Bageri Jul 13, 2020

96 23.4k 22.2m 0

How can we create that AngularCoreGym layer as a class library to that project. Did you created that Angular project separately and then from your local drive you have used that project as a class library OR you have created that project starting from an empty application in visual studio ?

Sagar Chowdhury Jun 24, 2020

1885 248 0 0 0 Reply

TRENDING UP

01 Typescript Express Node Server

02 ASP.NET CORE - CRUD Using Dependency Injection

03 How to Migrate (P2V) Physical to a Virtual Data Center - Convergence VMware Virtualization Concepts

04 JWT Token Authentication In Angular 14 And .NET Core 6 Web API

05 Easy To Learn .NET 6.0 And Angular - Getting Started Admin LTE Design

06 Caching In Entity Framework Core Using NCache

07 Implement In-Memory Cache In The .NET Core API

08 Types Of Cloud Computing In VMware Virtualization Concepts

09 Getting Started With Angular Electron Application Development

10 Rockin' The Code World with dotNetDave ft. Jirí Cincura Ep. 57

Learn Angular 8 In 25 Days

CHALLENGE YOURSELF

Angular 13