1. **RESTful API Service**

This project is a set of APIs for the library management system. It has endpoints for different entities – books, author, borrowers. It also has support for data validation using middleware. I included the complete database logic also, and it is compatible by both relational and non-relational databases (I left the code in place for SSMS and Amazon Dynamo, it should also work for Cassandra). I also included a comprehensive error handling logic using the middleware. I also included unit tests and functional tests. In addition to logic and code management, I also included files related to hosting and documentation using Swagger.

**Code**

It is a ASP.NET service-based code repository. I also included Docker and other hosting related files (replace the required keys).

1. **Voting Management System**

**Summary**

Voting management application is an API based service. There are endpoints exposed for both Polls and Votes. Security with JWT authentication is also implemented. I used middleware for error handling and also prevention of unauthorized manipulation including uniqueness of votes. One additional thing that I worked on for this is the real-time updates for voting counts using Socket.IO/SignalR software for .NET.

**Code**

I used dotnet core software to build the APIs, middleware and services.

**How to use**

1. Create a new Angular project and a new. NETCore Web API project.
2. In the. NETCore Web API project, install the IdentityCore NuGet package.
3. Create a new controller called UsersController and add the following routes:

Polls, Users, Votes

In the Startup file (I did not include it. This gets automatically created once we create a new project) register middleware’s as follows,

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// ...

app.UseMiddleware<VotingMiddleware>();

// ...

}

Register SignalR Hub

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

public void ConfigureServices(IServiceCollection services)

{

// ...

services.AddSignalR();

// ...

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// ...

app.UseEndpoints(endpoints =>

{

// ...

endpoints.MapHub<PollHub>("/pollHub");

});

// ...

}

For Authentication:

// Startup.cs

// Configure JWT authentication

services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = Configuration["Jwt:Issuer"],

ValidAudience = Configuration["Jwt:Issuer"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))

};

});

// Sample code for rate limiting middleware in Startup.cs

app.UseMiddleware<RateLimitMiddleware>();

1. **Weather App**

**Summary**

It takes location as the input and displays information about its weather. It also has CSS to support styling. I used OpenWeatherMapp API to get more accurate and granular weather information. It also has weather forecast option using the same API.

**Code**

The code has three files – index.html, styles.css and scripts.js.

The HTML has all the pieces together, CSS has the styling, and the JS file has the API call to the Weather-related APIs and random generation/weather forecast logic.

1. **Quiz App**

**Summary**

Javascript based questionnaire paired with HTML. It also has CSS and styling components, which are easily configurable. There is also a component for random question generation. The score is displayed at the end

**About the code**

There is only one file, the questions can be altered in the future and are easily configurable.