**Demo Web API .NET CORE 6**

**1. Basic Terms**

**1.1.** Web API

A Web API comprises a collection of publicly accessible endpoints that interact with the HTTP protocol. In the context of this project, the API facilitates operations such as adding, editing, retrieving, and deleting elements.

**1.2.** HTTP Protocol

The HTTP protocol governs the fundamental principles of data exchange over the Internet, dictating the structure of requests and responses. The project utilizes various HTTP verbs:

- GET: Retrieve resources from the server

- POST: Add new resources

- PUT: Update resources

- DELETE: Delete resources

**1.3.** REST

Representational State Transfer (REST) is a design style and standard for building web applications, emphasizing simplicity and readability. It ensures that addresses associated with specific methods/resources are self-explanatory through their path and corresponding HTTP verb. Recommended practices can be found here: <https://www.partech.nl/nl/publicaties/2020/07/9-trending-best-practices-for-rest-api-development#>

**1.4.** CORS

Cross-Origin Resource Sharing (CORS) is a security mechanism enforced by web browsers. It governs how web pages from one domain can request and interact with resources hosted on another domain. CORS establishes rules that determine whether a web application can access resources (such as APIs, fonts, images) from a different domain.

**2.** **Web API Project Files for .NET Core 5 and .NET Core 6**

**2.1.** .NET Core 5:

- Properties: The `LaunchSettings.json` file configures project startup settings, categorized into profiles for IIS or localhost. For instance, it controls whether the browser opens on startup.

- appsettings.json: This file centralizes configuration settings, eliminating scattered configurations throughout the project. Additional appsettings files like `appsettings.Development.json` store version-specific settings, determined by profiles in `LaunchSettings.json`.

- Program.cs: The `Main` method initializes and runs the web host, which is based on the `Startup.cs` file.

- Startup.cs: This file hosts two methods: `ConfigureServices` configures services and injects dependencies, while `Configure` arranges the request flow before the API responds. It constructs the HTTP request pipeline, with each middleware invoked on the app builder (IApplicationBuilder).

**2.2.** .NET Core 6:

- Program.cs: Similar to the previous version, it handles methods assigned to both `Startup.cs` and `Program.cs`. This includes creating the builder, registering dependencies, and housing middleware.

- appsettings.json: Continues to serve its previous role.

- Project file: Introduces the new feature `<implicitusings>enable<implicitusings>`, permitting global usings to declutter files.

- Program.cs: Includes steps such as creating the builder and adding builder options like NLog.

- Migration from .NET Core 5 to .NET Core 6: Copy code from the old `Startup.cs` to `Program.cs` (`Configure` and `ConfigureServices`), and adjust by placing `builder.` before existing configuration and service additions.

- Remove 'this' references.

**3.** **Middleware**

Middleware represents a layer of code that interfaces with the HTTP request context, including headers and HTTP verbs. Organized optimally, it influences the sequence of execution (e.g., authentication > mapping controllers). Middleware can also ascertain the active environment.

**4.** **Dependency Injection**

Dependency Injection is a software design pattern facilitating Inversion of Control. By registering dependencies in the service container, the framework assumes responsibility for creating and disposing of service instances. To refactor controller actions into services:

- Create a class with the methods to migrate from the controller.

- Develop a service based on this class.

- Register the service in `Startup`.

**5.** **Attributes**

The `[ApiController]` attribute enforces model validation on each request, rendering manual `ModelState` checks obsolete.

**6. Entity Framework**

The project leverages Entity Framework (EF) for database creation, seeding, and data operations. To use SQL Server, the package `Microsoft.EntityFrameworkCore.SqlServer` is required, along with `Microsoft.EntityFrameworkCore.Tools` for migrations. Before creating new migrations, add new entity sets to the DbContext class.

**7.** **Bogus**

To initially populate the database, a seeder class is employed, utilizing the Bogus package for rapid and easy data generation.

**8.** **AutoMapper**

The AutoMapper package facilitates the automatic mapping of entities to data transfer objects.

**9.** **NLOG**

NLOG is a logging library that tracks changes and events. The configuration is loaded from the `nlog.config` file, and logging occurs in `log.txt` within the build directory.

**10.** **Error Handling**

Error handling is integrated into the middleware to handle each request's processing.

**11.** **Swagger**

Using Swashbuckle.AspNetCore middleware, Swagger generates documentation. Adding Swagger UI enables access to endpoints, such as "swagger/index.html," where the documentation is presented.

**12. Microsoft.AspNetCore.Identity**

The `IPasswordHasher` from Microsoft.AspNetCore.Identity is used for hashing user passwords.

**13.** **FluentValidation**

FluentValidation validates models, enhancing data integrity.

**14. Microsoft.AspNetCore.Authentication.JwtBearer**

JwtBearer is necessary for employing JSON Web Tokens (JWT) for secure claims representation between parties. Configuration settings are added to `appsettings.json`, and the service is registered in `Program.cs` (or `Startup.cs` in .NET Core 5).

**15.** **LINQPad**

LINQPad serves as a scratchpad for LINQ queries.

**16.** **Microsoft.AspNetCore.StaticFiles**

The Microsoft.AspNetCore.StaticFiles middleware is included in Microsoft.AspNetCore.All, eliminating the need for separate installation in ASP.NET Core 2.x applications.

**17.** **Sources**

This demo draws inspiration from a Udemy course, recommended for those venturing into web API development: <https://www.udemy.com/course/praktyczny-kurs-aspnet-core-rest-web-api-od-podstaw/>

The generic project controller WeatherForecast, automatically generated, serves as a reference within the project.

For information on JWT in .NET Core 6, refer to this article: <https://www.infoworld.com/article/3669188/how-to-implement-jwt-authentication-in-aspnet-core-6.html>