Phase-3 web api

Introduction To Web Development

Web development refers to the process of creating websites and web applications that we use on the internet. ASP.NET Web Application is a framework developed by Microsoft. This framework is used to build web applications and services. Using it, we can create dynamic and interactive websites with the help of C# or VB.NET.

ASP.NET is a framework used for backend development. Some of its key features include:

1. **Dynamic Web Applications**: It allows us to create dynamic web pages that handle real-time data.
2. **Scalability**: This framework is suitable for building scalable applications.
3. **Integration with .NET Framework**: ASP.NET works with the .NET Framework, which provides multiple libraries and tools.

Workflow:

Client Request: The user sends a request through their browser.
Server Processing: The ASP.NET server processes the request and fetches the required data.
Response: The server sends the processed data back to the client in HTML format.

Web api project:
ASP.NET Web API is a framework used for creating RESTful web services. It primarily works on the backend and delivers data in JSON or XML format to client applications, such as browsers or mobile apps.

What is a Web API?

An API (Application Programming Interface) is a medium that allows two applications to interact with each other.
A Web API is an API that works through the HTTP protocol. It is used to create web services that:

- Provide data.
- Enable communication between the client and the server.

Workflow of a Web API

1. **Client Request**: The client (browser, mobile app) sends an HTTP request (GET, POST, PUT, DELETE).
2. **Routing**: The request is routed to the Web API controller.
3. **Controller Action**: The action methods in the Web API controller process the request and return data.
4. **Response**: The data is sent back to the client in JSON or XML format.

Action method:

In a Web API, the Action Method Response with Status Code is very important because it tells the client what the result of the request was. Web API action methods not only return data but also provide an HTTP status code, which helps the client understand the status of the response.

Options for Action Method Response

Web API provides different ways for action methods to return responses:

1. **Direct Data Return**: Only data is returned (e.g., in JSON or XML format).
2. **HTTP Response with Status Code**: The response is returned along with an HTTP status code.

Best Practices for Status Code Handling

1. **Consistent Responses**: Always return the proper status code in each action method.
2. **Error Messages**: Send meaningful error messages to the client (e.g., `BadRequest("Invalid input data")`).
3. **Exception Handling**: Properly handle exceptions and send a meaningful message along with a 500 Internal Server Error status.

Different status code:

200 - OK - successful request

201 - created - create new data

400 - Badrequest - invalid request

401 - unauthorized - required authorization

404 - not found - data not found

500 - internal server error - server issue

Etc..

CORS (Cross-Origin Resource Sharing)

CORS is a mechanism that allows web browsers to request resources from a different origin (domain). When a client (browser) calls a Web API hosted on another domain, CORS is used.

Why CORS is Important:

- If the client and server domains are different (e.g., frontend at `http://localhost:3000` and backend at `http://localhost:5000`), requests may be blocked without CORS policies.
- CORS ensures that only trusted domains can access resources

Authentication

Authentication is the process in which the server verifies the client's identity. In Web API, authentication ensures that the requesting user is authorized.

Types of Authentication:

- **Basic Authentication**: Via username and password.
- **Token-based Authentication**: The server generates a token (e.g., JWT), which is sent in the client’s requests. JWT Authentication is a commonly used method.

Authorization

Authorization is the process in which the server verifies what resources or actions an authenticated user is allowed to access. Once the user is authenticated, authorization determines what actions they can perform.

```
[Authorize(Roles = "Admin")]
public IHttpActionResult GetAdminData() {
    return Ok("This is Admin data.");
}
```

Exception Handling

Exception handling is used to manage errors that might occur on the server side. In Web API, exception handling ensures that if an error occurs, it is sent to the client with the appropriate status code.

JWT Token (JSON Web Token)

JWT is a token-based authentication mechanism used to verify the user's identity. When a user logs in successfully, the server generates a JWT token that is shared with the client. This token is sent with every request in the client’s headers for server authentication.

JWT Token Structure: JWT is a string divided into three parts:

- **Header**: Specifies the type of token (JWT) and the signing algorithm.
- **Payload**: Stores claims, such as user identity and privileges.
- **Signature**: Signs the token to prevent tampering.

HTTP caching:

HTTP Caching is a technique used to make web applications faster by temporarily storing frequently accessed data (in the cache) to reduce network traffic and improve response times.

When a client (browser) or server requests a resource (e.g., an image, HTML page, JSON data), the resource is sent from the server. In HTTP caching, the server or the client (browser) stores that resource in the cache. In subsequent requests, if the resource hasn't changed, the cached version is returned instead of fetching it from the server again.

Importance of HTTP Caching

- **Performance Improvement**: Using the cache, web pages load faster.
- **Reduced Server Load**: Unnecessary load on the server is minimized.

Use Cases of HTTP Caching

- **Static Assets (Images, JS, CSS)**: Static files are cached for long durations to prevent unnecessary requests to the server.
- **API Responses**: If data rarely changes, API responses can be cached to reduce server load.

Private Caching: Used for resources that are relevant only to a specific user. (`Cache-Control: private`)
Public Caching: Used for resources that are the same for all users. (`Cache-Control: public`)
No Cache: If you don't want a resource to be cached, you can use `Cache-Control: no-cache`.
Max-Age: Specifies the duration for which a resource should be cached. Example: `Cache-Control: max-age=86400` (24 hours).

HTTP Caching Workflow:

- **First Request**:
    1. The client requests a resource from the server.
    2. The server responds with the resource along with caching headers (`Cache-Control`, `Expires`, `ETag`, `Last-Modified`).

- **Subsequent Request**:
    1. The client requests the resource again.
    2. If the resource is cached and not expired, the client uses the cached version without contacting the server.
    3. If the cache has expired or the resource has changed, the client fetches the updated version from the server.

API Versioning Concept:

The main purpose of API versioning is to handle multiple versions of an API so that clients can receive the appropriate response for the correct version.
When breaking changes occur in an API, versioning ensures that older clients receive the compatible version while newer clients can access the latest features.

Versioning Methods in Web API

There are several common methods to implement API versioning in ASP.NET Web API:

a) URI Path Versioning (URL Path Versioning)

In this method, the version is specified in the URL path. This is the simplest and most common method.

Example:

- Version 1: `https://api.example.com/v1/products`
- Version 2: `https://api.example.com/v2/products`

Query String Versioning

In this method, the version is specified through the query string

Example:

- Version 1: `https://api.example.com/products?version=1`
- Version 2: `https://api.example.com/products?version=2`

Benefits of API Versioning:

- **Backward Compatibility**: Existing clients continue to use the old API version, while new clients get the latest features.
- **Controlled Updates**: You can update your API without affecting older users.
- **Flexibility**: Clients can choose the version of the API that fits their requirements.

Swagger:

Swagger is an API documentation tool used to define, document, and test web APIs. The main focus of Swagger is to describe the API's endpoints, define their parameters, and provide a testing interface. It allows developers to easily document their APIs and enables users to interact with and test the API through a user-friendly interface.

API Testing: Through Swagger UI, you can test the API, but its main focus is on documentation.

API Design: Swagger provides the convenience of defining the API specification while designing the API.

Postman:

Postman is an API testing tool used to send API calls, view their responses, and validate the behavior of APIs. While Postman is primarily used for API testing, it also includes features for API documentation and automation.

API Testing: The main use case of Postman is API testing, where you make API calls and validate the responses.

Collaboration: Postman offers an option to collaborate with teams, allowing you to share collections.