**Web Development**

**Introduction to Web Development**

Web development involves building applications that run on web browsers, interacting with users through the internet. It consists of both front-end (UI/UX) and back-end (server-side) development. In the case of Web APIs, it's specifically about building backend services that provide endpoints for clients to communicate with (such as a mobile app, web browser, or another API).

In .NET Framework 4.8, Web API is used to build RESTful services. These services expose endpoints over HTTP, allowing clients to interact with resources (such as data or functionalities) over the web.

**Web API project**

A Web API project in .NET Framework 4.8 is typically created using the ASP.NET Web API template. This template provides the necessary setup to build a RESTful API with standard HTTP methods like GET, POST, PUT, DELETE, etc.

* Web API is used to expose your server-side logic to clients (like browsers or mobile apps) via HTTP requests.
* You define controllers and actions within those controllers to handle the requests.

The key structure typically includes:

* Controllers: Where the logic to handle requests resides.
* Models: Represent the data structure.
* Routing: Defines how HTTP requests map to controller actions.

ASP.NET Web API is a framework for building HTTP-based services that can be consumed by a wide range of clients, including browsers, mobile devices, and desktop applications. It is designed for creating RESTful APIs in a flexible and scalable manner.

**Key Features of ASP.NET Web API**

1. **Lightweight Framework**: It is a lightweight architecture suitable for small and large-scale applications.
2. **RESTful Principles**: Designed to work seamlessly with HTTP verbs like GET, POST, PUT, DELETE, etc.
3. **Content Negotiation**: Automatically returns data in formats like JSON, XML, etc., based on client preference.
4. **Dependency Injection**: Supports IoC (Inversion of Control) for better testability and flexibility.
5. **Self-Hosting**: Can run independently without IIS, useful for services running in isolated environments.

**Building Web API**

1. **Start a New Project**:
   * Open Visual Studio.
   * Select **File > New > Project**.
   * Choose **ASP.NET Web Application (.NET Framework)**.
   * Name the project and click **OK**.
2. **Select Project Template**:
   * In the **New ASP.NET Project** dialog, choose **Empty**.
   * Enable **Web API** to include required libraries and configurations.
3. **Structure of Web API Project**:
   * **Controllers Folder**: Contains Web API controllers. Each controller handles HTTP requests.
   * **Models Folder**: Contains model classes representing the data structure.
   * **App\_Start Folder**: Contains configuration files like WebApiConfig.cs for routing.
   * **Web.config**: Configuration file for the application.
4. **Configuring Routes**:
   * Open WebApiConfig.cs (in the App\_Start folder).
   * Define the API routing using the MapHttpRoute method
5. **Creating a Controller:**
   * Add a new controller by right-clicking the Controllers folder and selecting Add > Controller.
   * Choose Web API Controller - Empty and name it (e.g., ProductsController).
6. **Adding Actions:**
   * Implement methods in the controller to handle HTTP requests
7. **Testing the API:**
   * Run the application and use a tool like Postman or Swagger to test your API endpoints.

**Action Method Response (HTTP status code etc.)**

Action methods in Web API return HTTP responses. The status code in the HTTP response indicates the outcome of the request.

**Common HTTP Status Codes:**

* **200 Ok()** : The request was successful (used with GET, PUT, POST, etc.)
* **201 Created()** : A resource was successfully created (used with POST)
* **302 Redirect()** : The resource has temporarily moved to a different URI (used with GET)
* **400 BadRequest()** : The request is invalid (e.g., missing parameters, malformed syntax)
* **401 Unauthorized()** : Authentication is required and has failed or not been provided
* **404 NotFound()** : The requested resource could not be found on the server
* **409 Conflict()** : There is a conflict with the current state of the resource (e.g., trying to create a resource that already exists)
* **500 InternalServerError()** : Something went wrong on the server, causing it to fail processing the request

**Security (CORS, Authentication, Authorization,Exception, JWT token etc.)**

Security is crucial when developing Web APIs. In .NET Framework, you can secure your API in several ways:

* CORS (Cross-Origin Resource Sharing): Enables or restricts resources to be requested from another domain. You would configure this in your Web API settings or use attributes like EnableCorsAttribute to allow cross-origin requests.

Package : Microsoft.AspNet.Cors

* Authentication: Verifies the identity of the user making the request. For example, you can use Basic Authentication, OAuth, or JWT (JSON Web Tokens) for token-based authentication.
* Authorization: After authentication, authorization determines whether the authenticated user has permission to access the requested resource. You can control this with roles or claims.
* JWT Token: A commonly used method for stateless authentication. It includes user data and is passed in headers for authorization.
* Exception Handling: Catch and handle exceptions globally with middleware or custom exception filters to provide meaningful error responses.

**HTTP caching**

Caching improves performance by storing copies of responses and serving them without contacting the server repeatedly. ASP.NET Web API supports output caching and client-side caching.

Types of Caching

1. Server-Side Caching: Store responses on the server.
2. Client-Side Caching: Use cache-related headers like Cache-Control.

Ex:

HttpContext.Current.Cache.Insert(key, value, null, DateTime.MaxValue, duration);

HttpContext.Current.Cache.Remove(key);

HttpContext.Current.Cache[key];

**Versioning**

Versioning your API is important to maintain backward compatibility when new features or changes are introduced. Common strategies for API versioning are:

* URL Versioning: e.g., /api/v1/products
* Header Versioning: Include a version header like X-API-Version: 1.

In .NET, you can version an API by using routing conventions like api/v{version}/[controller].

**Use of Swagger**

Swagger (now known as OpenAPI) is a framework for documenting and testing RESTful APIs. In .NET, you can use Swashbuckle to automatically generate Swagger documentation for your API.

* It creates an interactive UI where developers can explore your API's endpoints and make test calls.

Package : Swashbuckle

**Use of POSTMAN**

Postman is a tool used for testing Web APIs. With Postman, you can:

* Send HTTP requests to your API (GET, POST, PUT, DELETE).
* View the response status, headers, and body.
* Automate tests for APIs.

It's a great way to ensure that your API works as expected during development and debugging.

**Deployment**

Deployment involves taking your Web API and making it accessible to the public (or a private network). In the case of .NET Framework 4.8, typical deployment strategies include:

* IIS (Internet Information Services): Host your API on a Windows server with IIS.

Steps in Deployment:

1. Publish your Web API project from Visual Studio.
2. Configure your hosting environment (e.g., IIS, cloud service).
3. Ensure all dependencies (like databases) are correctly configured.