**Web Development**

**Introduction to Web Development**

Web development involves creating and managing websites and web applications. In the context of ASP.NET Web API, it's a framework designed for building HTTP-based services. These services can communicate with clients through various HTTP methods like GET, POST, PUT, and DELETE.

ASP.NET Web API (non-Core, non-MVC) is built on the .NET Framework and provides a lightweight, easy-to-use platform to build RESTful services. It allows for the creation of web services that can be consumed by various clients, including web browsers, mobile applications, and other systems.

Key Benefits:

* Lightweight and efficient.
* Easy integration with different types of clients.
* Supports various protocols such as HTTP, HTTPS, WebSockets, etc.

**Attribute and Conventional Routing**

**Conventional Routing**

In ASP.NET Web API, routing refers to mapping HTTP requests to specific controller actions. With conventional routing, the route URL follows a specific pattern that the framework uses to map the incoming requests.

**Attribute Routing**

Attribute routing is more flexible, allowing you to specify routes directly in the controller actions using attributes.

**Action Method Response**

A WebAPI returns an Http Response which can be of different types (eg. IHttpActionResult). They returns Http Responses such as Ok, NotFound, InternalServerError, etc.

**Security in Web API**

**CORS (Cross-Origin Resource Sharing)**

CORS is a mechanism that allows web applications from one domain to access resources from another domain. In ASP.NET Web API, you can configure CORS to allow or restrict specific domains.

To enable CORS:

1. Install the Microsoft.AspNet.WebApi.Cors package via NuGet.
2. Modify the WebApiConfig.cs to enable CORS:

config.EnableCors();

CORS can also be applied at the attribute level of an action method and will be having higher priority than the global CORS configuration.

**JWT (JSON Web Token)**

JWT is a compact, URL-safe token used to represent claims between two parties. In Web API, JWT is commonly used for authentication and authorization.

To implement JWT authentication:

1. Add NuGet package System.IdentityModel.Tokens.Jwt.
2. Implement a JWT generation mechanism and validate the token in the Web API.

JWT validation typically occurs in the Authorization header. Add the [Authorize] attribute at top of the action method expecting a jwt token in the request header.

**Exception Handling**

To handle exceptions globally in an ASP.NET Web API, you can use a custom exception filter or handle it inside action methods using try-catch blocks and return the response accordingly.

**HTTP Caching**

HTTP Caching is a mechanism for improving the performance of your Web API by reducing redundant requests. You can implement caching using HTTP headers like Cache-Control, ETag, etc.

**API Versioning**

API versioning ensures that older versions of the API remain functional while new features are introduced. Eg:

config.Routes.MapHttpRoute(

name: "ApiVersioning",

routeTemplate: "api/v{version}/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

**Swagger for API Documentation**

Swagger is a tool for documenting and testing Web APIs. To use Swagger in Web API, you need to install the Swashbuckle NuGet package and then enable it in the WebApiConfig.cs file as:  
config.EnableSwagger(c => c.SingleApiVersion("v1", "My API")) .EnableSwaggerUi();

**Deployment of Web API**

To deploy an ASP.NET Web API, you can use IIS (Internet Information Services) or cloud services such as Azure.

**Deploying to IIS:**

1. Publish the Web API using Visual Studio or via the command line.
2. Create a website in IIS and point the site to the folder where the application is published.
3. Ensure the Web.config file is configured correctly for production.