ASP.NET Web API Notes

## 1. Overview of ASP.NET Web API

ASP.NET Web API is used to create HTTP services for various clients, including browsers and mobile devices. It is part of the ASP.NET framework and is ideal for building RESTful applications that return data in formats like JSON or XML.

## 2. Core Components

- \*\*Controllers\*\*: Classes derived from `ApiController` handle incoming HTTP requests.  
- \*\*Routing\*\*: ASP.NET Web API uses routing to map requests to the appropriate controller action. Routes can be configured in the `WebApiConfig` class or by using attribute routing (e.g., `[Route("api/values")]`).  
- \*\*Models\*\*: Data structures that represent the application’s data. Models can be simple or complex objects that controllers use to pass data back and forth.  
- \*\*Dependency Injection\*\*: ASP.NET Web API supports DI, making it easy to inject services or repositories into controllers.

## 3. HTTP Verbs and CRUD Operations

- \*\*GET\*\*: Retrieve resources (e.g., `Get` method in the controller).  
- \*\*POST\*\*: Create a new resource (e.g., `Post` method).  
- \*\*PUT\*\*: Update an existing resource (e.g., `Put` method).  
- \*\*DELETE\*\*: Remove a resource (e.g., `Delete` method).

## 4. Building a Basic Web API

- \*\*Creating a Controller\*\*: Add a new controller by inheriting from `ApiController` or `ControllerBase` in ASP.NET Core. Define methods corresponding to HTTP verbs.  
- \*\*Defining Routes\*\*: Use attribute routing or configure routes in `Startup.cs` for ASP.NET Core projects.  
- \*\*Action Results\*\*: Return appropriate HTTP status codes (e.g., `Ok`, `CreatedAtAction`, `NoContent`, `BadRequest`).

## 5. Serialization and Content Negotiation

ASP.NET Web API supports content negotiation, which automatically selects the best response format (e.g., JSON, XML) based on the client’s request. JSON is the default format. XML support can be added by configuring the `Startup.cs` or by using media type formatters.

## 6. Middleware and Pipeline

- \*\*Middleware\*\*: Configure middleware in `Startup.cs` for tasks like authentication, error handling, and logging.  
- \*\*Exception Handling\*\*: Implement global exception handling using middleware or filters.

## 7. Authorization and Authentication

ASP.NET Web API supports various authentication methods, including JWT (JSON Web Token), OAuth, and Identity. Configure authorization policies and apply them to specific controllers or actions using `[Authorize]`.

## 8. Versioning

Web APIs often require versioning as they evolve. ASP.NET Web API supports URL versioning, query string parameters, and headers for version management. Tools like \*\*Microsoft.AspNetCore.Mvc.Versioning\*\* package can help implement versioning.

## 9. Testing

- Use tools like \*\*Postman\*\* or \*\*Swagger\*\* (via Swashbuckle) to test your Web API.  
- Unit tests can also be written for controller methods using a testing framework like \*\*XUnit\*\* or \*\*MSTest\*\*.

## 10. Best Practices

- \*\*Use DTOs\*\*: Use Data Transfer Objects (DTOs) to encapsulate the data structure.  
- \*\*Validation\*\*: Apply model validation using data annotations or custom validation.  
- \*\*Asynchronous Programming\*\*: Make use of `async` and `await` to improve performance for I/O-bound operations.  
- \*\*Caching\*\*: Use caching for frequently accessed data to reduce load on the server.  
- \*\*Error Handling\*\*: Centralize error handling and return meaningful error messages to clients.