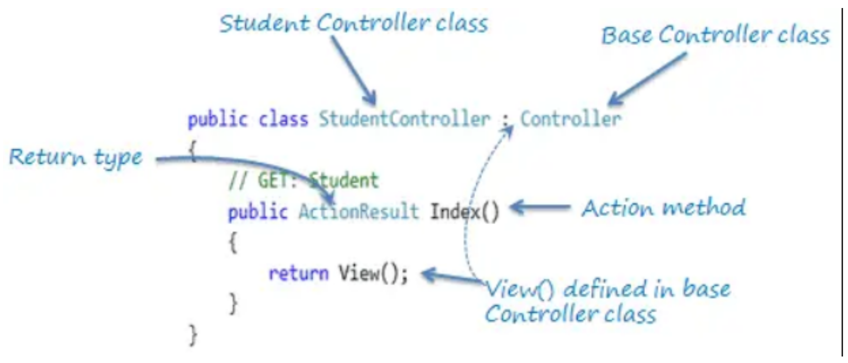
Web Development

Introduction to Web Development
Web API project
Building Web API
Action Method Response (HTTP status code etc.)
Security (CORS, Authentication, Authorization,Exception, JWT token etc.)
HTTP caching
Versioning
Use of Swagger
Use of POSTMAN
Deployment

Action Method Response

All the public methods of the `Controller` class are called `Action` methods. They are like any other normal methods with the following restrictions:

1. Action method must be public. It cannot be private or protected
2. Action method cannot be overloaded
3. Action method cannot be a static method.

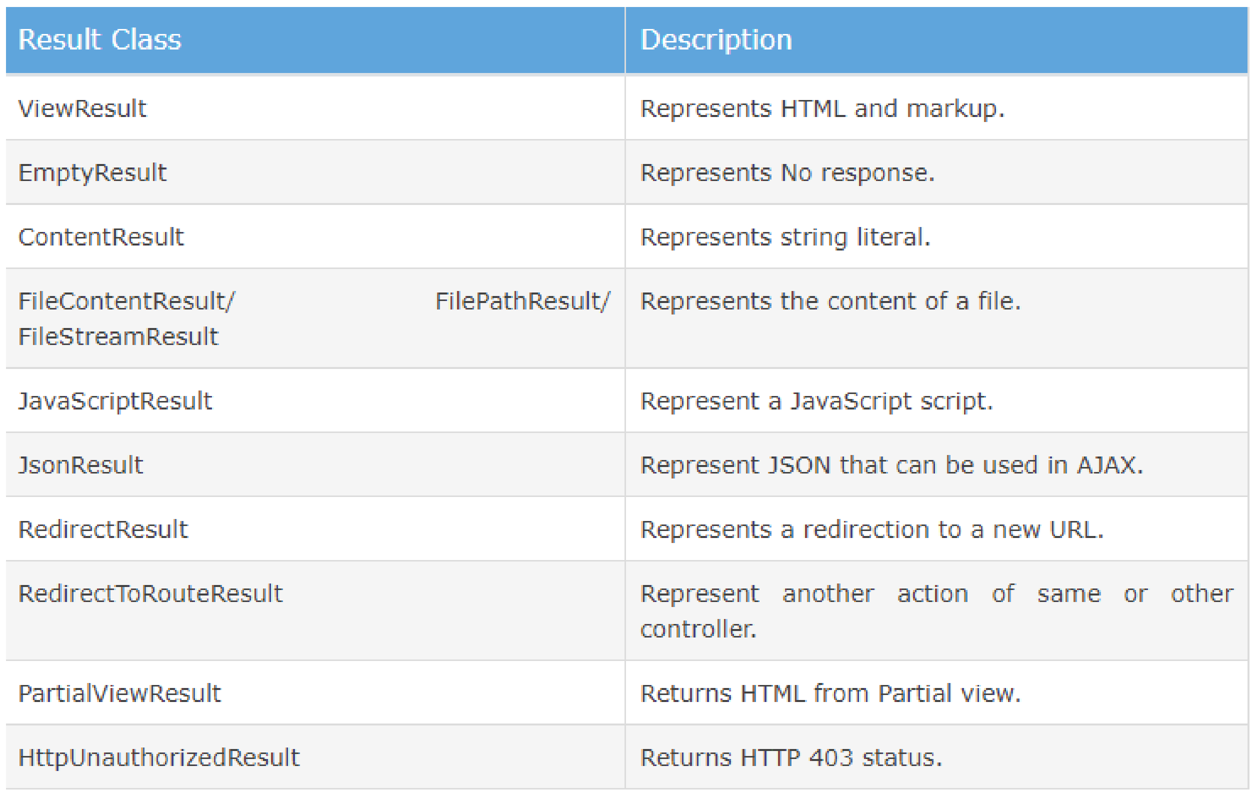


```
public class StudentController : Controller
{
    // GET: Student
    public ActionResult Index()
    {
        return View();
    }
}
```

As you can see in the above figure, the `Index()` method is public, and it returns the `ActionResult` using the `View()` method. The `View()` method is defined in the `Controller` base class, which returns the appropriate `ActionResult`.

ActionResult

MVC framework includes various `Result` classes, which can be returned from an action method. The result classes represent different types of responses, such as HTML, file, string, JSON, javascript, etc. The following table lists all the result classes available in ASP.NET MVC.



Result Class	Description
ViewResult	Represents HTML and markup.
EmptyResult	Represents No response.
ContentResult	Represents string literal.
FileContentResult/ FilePathResult/ FileStreamResult	Represents the content of a file.
JavaScriptResult	Represent a JavaScript script.
JsonResult	Represent JSON that can be used in AJAX.
RedirectResult	Represents a redirection to a new URL.
RedirectToRouteResult	Represent another action of same or other controller.
PartialViewResult	Returns HTML from Partial view.
HttpUnauthorizedResult	Returns HTTP 403 status.

HTTP caching

Overview

The HTTP cache stores a response associated with a request and reuses the stored response for subsequent requests.

There are several advantages to reusability. First, since there is no need to deliver the request to the origin server, then the closer the client and cache are, the faster the response will be. The most typical example is when the browser itself stores a cache for browser requests.

Also, when a response is reusable, the origin server does not need to process the request — so it does not need to parse and route the request, restore the session based on the cookie, query the DB for results, or render the template engine. That reduces the load on the server.

Types of caches

In the HTTP Caching spec, there are two main types of caches: private caches and shared caches.

Private caches

A private cache is a cache tied to a specific client — typically a browser cache. Since the stored response is not shared with other clients, a private cache can store a personalized response for that user.

On the other hand, if personalized contents are stored in a cache other than a private cache, then other users may be able to retrieve those contents — which may cause unintentional information leakage.

Shared cache

The shared cache is located between the client and the server and can store responses that can be shared among users. And shared caches can be further sub-classified into proxy caches and managed caches.

Types of Caching in ASP.NET ➖

Output Caching:

- Description: Output caching stores the HTML output generated by a web page or user control so that it can be reused for subsequent requests.
- Implementation: You can enable output caching by using the `OutputCache` directive in the ASP.NET page or by setting caching options programmatically in the code.

Fragment Caching:

- Description: Fragment caching allows you to cache specific parts (fragments) of a page rather than the entire page. This is useful when only certain portions of a page are expensive to generate.
- Implementation: Similar to output caching, you can use the `OutputCache` directive or programmatically set fragment caching options.

Data Caching:

- Description: Data caching involves storing application data, such as datasets or objects, in memory to reduce the need to fetch the data from the data source repeatedly.
- Implementation: You can use the `Cache` object in ASP.NET to store and retrieve data from the cache.

SQL Cache Dependency:

- Description: SQL cache dependency allows you to cache data based on the changes in a SQL Server database. When the underlying data changes, the cached data is automatically invalidated.
- Implementation: You can set up SQL cache dependency by specifying the SQL commands and connection string, and associating the cached item with a database table.

Post-cache Substitution:

- Description: Post-cache substitution allows you to replace specific portions of a cached page with dynamic content when the page is served.
- Implementation: You can use the `Substitution` control in ASP.NET to specify dynamic content that will replace specific placeholders in the cached output.

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

1. Cross-Origin Resource Sharing (CORS)

1.1 Overview

- Definition: Explain what CORS is and its significance in web security.

1.2 Implementation

- Server Configuration:
    - Detail how to configure the server to handle CORS.
- Client Configuration:
    - Guide developers on configuring the client-side to work with CORS.

1.3 Best Practices

- Provide best practices for securing CORS, including whitelisting origins and handling preflight requests.

2. Authentication

2.1 Overview

- Definition: Explain what authentication is and why it is crucial for security.

2.2 Implementation

- Token-based Authentication:
    - Describe the process of implementing token-based authentication.
- Session-based Authentication:
    - Explain the use of session-based authentication and its implementation.

2.3 Best Practices

- Provide guidelines for secure password storage, use of multi-factor authentication, and regular token expiration.

3. Authorization

3.1 Overview

- Definition: Clarify the concept of authorization and its role in controlling access.

3.2 Implementation

- Role-Based Access Control (RBAC):
    - Detail the implementation of RBAC.
- Attribute-Based Access Control (ABAC):
    - Explain how to implement ABAC for more fine-grained control.

3.3 Best Practices

- Emphasize the principle of least privilege, regular access reviews, and proper error handling for unauthorized access attempts.

4. Exception Handling

4.1 Overview

- Definition: Explain the importance of proper exception handling for security.

4.2 Implementation

- Server-side Exception Handling:
    - Describe how to handle exceptions on the server side.
- Client-side Exception Handling:
    - Explain best practices for handling errors on the client side.

4.3 Best Practices

- Provide guidelines for avoiding exposing sensitive information in error messages and logging errors securely.

5. JSON Web Token (JWT)

5.1 Overview

- Definition: Explain what JWT is and its use in authentication and information exchange.

5.2 Implementation

- Token Generation:
    - Detail how to generate JWT tokens.
- Token Verification:
    - Explain the process of verifying JWT tokens.

5.3 Best Practices

- Highlight the importance of using strong algorithms, securing token storage, and regularly rotating keys.

Use of Swagger

Swagger is an Open Source set of rules, specifications and tools for developing and describing RESTful APIs. The Swagger framework allows developers to create interactive, machine and human-readable API documentation.

API specifications typically include information such as supported operations, parameters and outputs, authorization requirements, available endpoints and licenses needed. Swagger can generate this information automatically from the source code by asking the API to return a documentation file from its annotations.

Swagger helps users build, document, test and consume RESTful web services. It can be used with both a top-down and bottom-up API development approach. In the top-down, or design-first, method, Swagger can be used to design an API before any code is written. In the bottom-up, or code-first method, Swagger takes the code written for an API and generates the documentation.

Benefits of Swagger

In addition to its goal of standardizing and simplifying API practices, a few additional benefits of Swagger are:

- It has a friendly user interface that maps out the blueprint for APIs.
- Documentation is comprehensible for both developers and non-developers like clients or project managers.
- Specifications are human and machine readable.
- Generates interactive, easily testable documentation.
- Supports the creation of API libraries in over 40 languages.
- Helps automate API-related processes.

Use of POSTMAN

Introduction to Postman

- Postman is a standalone software testing API (Application Programming Interface) platform to build, test, design, modify, and document APIs. It is a simple Graphic User Interface for sending and viewing HTTP requests and responses.
- While using Postman, for testing purposes, one doesn't need to write any HTTP client network code. Instead, we build test suites called collections and let Postman interact with the API.
- In this tool, nearly any functionality that any developer may need is embedded. This tool has the ability to make various types of HTTP requests like GET, POST,

PUT, PATCH, and convert the API to code for languages like JavaScript and Python.

Use of Postman

API Testing:

- Manual Testing: Postman allows you to send various types of HTTP requests (GET, POST, PUT, DELETE, etc.) to API endpoints and inspect the responses.
- Automated Testing: You can create and run automated test scripts using the Postman collection runner, allowing you to test multiple API endpoints and scenarios.

Request Building:

- Postman provides a visual interface for constructing and formatting HTTP requests. You can easily add parameters, headers, and authentication details to your requests.

Environment and Variables:

- Postman allows you to define environments and use variables, making it easy to switch between different configurations (e.g., development, staging, production) without changing your requests manually.

Mock Servers:

- Postman can generate mock servers for your APIs. This is useful for testing and development when the actual API is not available or is still in progress.

Documentation:

- You can create and share API documentation using Postman. This includes details such as request and response examples, authentication methods, and endpoint descriptions.

Collaboration:

- Postman enables collaboration among team members by allowing the sharing of collections, environments, and documentation. Teams can work on API development and testing collaboratively.

Monitoring:

- Postman provides monitoring capabilities to track the performance of your APIs. You can set up monitors to run collections at scheduled intervals and receive notifications for any issues.

Automated Workflows:

- Postman supports the creation of automated workflows through the use of pre-request scripts, tests, and other automation features, allowing you to streamline complex API testing scenarios.

Integration with CI/CD:

- Postman can be integrated into continuous integration and continuous deployment (CI/CD) pipelines, ensuring that API tests are automatically executed as part of the development and deployment process.

Security Testing:

- Postman can be used for security testing of APIs by incorporating security-related tests into the automated testing process.

Deployment ➖

Deployment, in the context of software development, refers to the process of making a software application or system available for use. It involves moving the developed application from the development environment to a production environment where end-users can access and utilize it. Deployment encompasses various tasks, including installation, configuration, testing, and validation to ensure that the software operates correctly in the production environment.

Key aspects of deployment include:

- **Installation:**
    - The software is installed on the target servers or devices. This involves copying executable files, libraries, configuration files, and other necessary components to the designated locations.
- **Configuration**:
    - Setting up and configuring the software for the specific production environment. This may involve specifying database connections, server addresses, and other environment-specific settings.
- **Testing:**
    - Conducting thorough testing to ensure that the deployed software functions as expected in the production environment. This includes functional testing, performance testing, and other relevant tests.
- **Validation:**
    - Verifying that the deployed application meets the requirements and specifications outlined during the development phase. Validation ensures that the software aligns with user expectations and business needs.
- **Rollback Plan:**
    - Having a contingency plan in place to revert to a previous state in case the deployment encounters issues or if there is a need to roll back to a previous version.

Deployment can be done manually, where each step is executed by individuals, or it can be automated using deployment tools and scripts. Automated deployment is often preferred for its efficiency, consistency, and ability to reduce the risk of human error.

Versioning

Once a web API Service is made public, different client applications start using and relying on it.

As business grows and requirements change the services need to change without breaking existing clients and at the same time satisfying new client requirements.

Wats to Implement:

· URI’s

· QueryString

· versionHeader

· AcceptHeader

· MediaType

Controller Selector in web api select controller() method in defaultHttpController Selector class selects the controller based on the information in the URI.

Base library features

The Base Class Library (BCL) in C# refers to a set of classes, interfaces, and value types that are part of the .NET Framework. These classes provide fundamental functionality for developing applications in C#. The BCL is an integral part of the .NET framework and includes a wide range of features, such as:
Data Types: Basic data types like integers, floats, strings, and more.
Collections: Classes for working with lists, arrays, dictionaries, queues, etc.
File I/O: Classes for reading and writing files and working with directories.

Networking: Classes for handling network communication.
Threading: Classes for managing multithreading and asynchronous programming.
Reflection: Classes for examining and interacting with the metadata of types.
Security: Classes for handling encryption, cryptography, and security.
XML and Data Serialization: Classes for working with XML and serializing/deserializing data.
Exception Handling: Classes for handling exceptions and errors.
DateTime and Time: Classes for working with dates and times.
LINQ (Language-Integrated Query): A set of features for querying collections in a uniform way.

The BCL provides a foundation for building C# applications and is an essential part of the .NET ecosystem. When you write C# code, you often use classes and functionalities from the Base Class Library to perform common tasks and interact with the underlying system.

In C#, the Base Class Library (BCL) provides a wide range of classes and functionality that developers use regularly for various programming tasks. Here are some commonly used namespaces and classes from the BCL:

System:

- Object: The root of the type hierarchy.
- String: Provides methods for working with strings.
- Console: Handles console input and output.
- Math: Contains mathematical functions.
- DateTime: Represents dates and times.

System.Collections:

- List<T>, Dictionary<K, V>, Queue<T>, Stack<T>: Collection classes for managing data.

System.IO:

- File, Directory: Provides methods for file and directory manipulation.
- StreamReader, StreamWriter: Read from and write to streams.

System.Net:

- WebClient, HttpWebRequest, HttpWebResponse: Classes for making HTTP requests.

System.Threading:

- Thread, ThreadPool, Task: Classes for multithreading and asynchronous programming.

System.Xml:

- XmlDocument, XmlElement, XmlReader, XmlWriter: Classes for working with XML.

System.Linq:

- LINQ (Language Integrated Query) provides a set of standard query operators that operate on sequences of elements.