Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM **(INDEPENDENT SKILL DEVELOPMENT MISSION)**

INTRODUCTION TO ASP.NET

WHAT IS ASP.NET AND ITS HISTORY

ASP.Net is a popular open-source web framework developed by Microsoft. It allows developers to build dynamic websites, web applications, and web services. ASP.Net is part of the .NET ecosystem, which is a robust set of technologies designed for creating web applications, services, and more. The framework was first introduced in 2002 as part of the .NET Framework with the goal of improving the ease of developing web applications by offering a more powerful and flexible platform than previous technologies.

Over the years, ASP.Net has evolved significantly. Initially, ASP.Net Web Forms dominated the scene, allowing developers to build web applications using a drag-and-drop interface. However, with the growth of modern web development trends, ASP.Net transitioned to embrace the Model-View-Controller (MVC) pattern and introduced Web API for RESTful web services, offering developers more control, flexibility, and scalability. The framework has undergone continuous improvement, becoming more streamlined with ASP.Net Core, a cross-platform version that runs on Windows, macOS, and Linux.

ASP.Net has gained widespread adoption due to its strong integration with the .NET framework, support for various programming languages like C# and VB.NET, and a rich set of libraries and tools. It offers features such as security, caching, data access, and session management, which are essential for building enterprise-level applications. Today, ASP.Net Core is the future of the framework, allowing developers to build high-performance applications that are scalable, maintainable, and capable of running on multiple platforms.

DIFFERENCE BETWEEN ASP.NET WEB FORMS, MVC, AND WEB API

ASP.Net provides three different approaches to building web applications: Web Forms, MVC (Model-View-Controller), and Web API. Understanding these approaches and their use cases is crucial for any developer working with ASP.Net.

ASP.Net Web Forms: Web Forms is the older of the three approaches. It is based on an event-driven programming model, where developers can design the UI using a drag-and-drop interface. The code is separated into two files: one for the UI (the .aspx file) and another for the backend logic (the .aspx.cs file). Web Forms abstracts much of the underlying HTML, allowing developers to focus more on logic than on markup. However, this can lead to less control over the rendered HTML and make the application difficult to scale and maintain. Web Forms is still used in legacy applications but is being gradually replaced by more modern approaches.

ASP.Net MVC: MVC is a design pattern that separates an application into three components: the Model (which represents data and business logic), the View (which is responsible for rendering UI), and the Controller (which handles user input and updates the Model). This approach gives developers full control over the HTML output, making it easier to maintain and scale applications. ASP.Net MVC follows the principles of separation of concerns, which results in cleaner, more modular code. It is widely used for building dynamic, data-driven web applications and APIs.

ASP.Net Web API: Web API is designed to build HTTP-based services that can be consumed by a variety of clients, including web browsers, mobile devices, and other web services. Unlike Web Forms and MVC, which are used for building web applications, Web API is specifically intended for building RESTful APIs. It is lightweight, flexible, and designed to handle HTTP requests and responses in a stateless manner. Web API supports a wide range of data formats, including JSON and XML, making it ideal for modern web development, where APIs are an essential part of the application architecture.

Each of these approaches has its strengths and is suited for different use cases. Web Forms is suitable for simple, data-driven web applications, while MVC is better for building scalable, maintainable, and testable web applications. Web API is ideal for creating services that need to be consumed by multiple clients over HTTP.

SETTING UP YOUR DEVELOPMENT ENVIRONMENT (VISUAL STUDIO)

Setting up your development environment is a critical step when working with ASP.Net. Visual Studio is the most widely used Integrated Development Environment (IDE) for building ASP.Net applications, offering robust features like code completion, debugging, and project templates that make development more efficient.

Step 1: Installing Visual Studio
To begin working with ASP.Net, download and install Visual Studio from the official Microsoft website. During installation, make sure to select the **ASP.Net and web development** workload. This will install all the necessary tools and libraries required

to build ASP.Net applications, including the .NET framework and the ASP.Net runtime.

Step 2: Creating a New ASP.Net Project
Once Visual Studio is installed, open it and select **Create a New Project**. From the project template selection screen, choose an ASP.Net template. You can select from options like **ASP.Net Core Web Application** or **ASP.Net Web Application**, depending on your preferred version of the framework. For beginners, starting with the Web Application template is recommended, as it includes a simple web form and is easy to configure.

Step 3: Exploring Visual Studio Interface
After creating a project, Visual Studio's interface will present multiple panels, including the Solution Explorer, where you can view the structure of your project, and the Code Editor, where you will write your application code. The **Toolbar** at the top allows you to run, debug, and build your application. The **Properties Window** helps you configure settings like the database connection or UI elements. Explore these features to get familiar with the IDE.

Step 4: Running Your First ASP.Net Application
Once your project is set up, you can test it by running the application. Visual Studio's built-in web server, IIS Express, will launch the app in a browser. You can see the default page and test it by making changes to the code. This gives you a hands-on experience with the environment and helps you understand how ASP.Net applications are executed.

EXERCISE

1. **Create a New ASP.Net Project**: Open Visual Studio, create a new ASP.Net Core or Web Application project, and run it. Modify the default page to display a personalized message and test the changes in the browser.

2. **Compare Web Forms with MVC**: Create two separate projects—one using Web Forms and the other using MVC. Try adding a simple page in both approaches and compare the code structure, the way data is managed, and how the UI is rendered.

CASE STUDY: DEVELOPING A SIMPLE BLOG APPLICATION USING ASP.NET MVC

In this case study, we will explore how to develop a basic blog application using ASP.Net MVC. The blog will allow users to post articles, view them, and delete their posts. By creating this application, you'll gain a deeper understanding of how the MVC pattern works in ASP.Net.

- **Model**: Create a Post model that contains properties like Title, Content, and DatePosted.
- **View**: Design a simple view to display a list of posts. Use Razor syntax to dynamically generate HTML based on the model.
- **Controller**: Implement a controller to handle user actions like posting an article, viewing posts, and deleting posts.

BASICS OF WEB DEVELOPMENT

UNDERSTANDING HTTP, HTML, CSS, AND JAVASCRIPT

Web development is a broad field that involves creating websites and web applications using a combination of technologies. Four fundamental technologies form the foundation of web development: HTTP, HTML, CSS, and JavaScript. Each of these plays a critical role in how websites function and interact with users.

HTTP (Hypertext Transfer Protocol) is the protocol used to transfer data over the web. It defines how messages are formatted and transmitted between clients (usually web browsers) and servers. When you type a URL into your browser, it sends an HTTP request to a server, which then responds with the requested content. The request-response cycle is essential for the functioning of the web. The most common HTTP methods include **GET** (to retrieve data), **POST** (to send data), **PUT** (to update data), and **DELETE** (to remove data). Understanding HTTP is crucial for managing how data flows between clients and servers.

HTML (Hypertext Markup Language) is the standard markup language used to create the structure of web pages. HTML uses tags to define elements like headings, paragraphs, links, images, forms, and more. It provides the skeleton of a webpage, allowing developers to organize content and create interactive elements. For example, an HTML file might look like this:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>My First Web Page</title>

</head>

<body>

  <h1>Welcome to My Website</h1>

  <p>This is a simple webpage created with HTML.</p>

</body>
```

```
</html>
```

In this example, the page contains a heading and a paragraph, defining the content structure.

CSS (Cascading Style Sheets) is used to style and layout HTML elements. While HTML provides the structure, CSS controls the appearance. CSS allows developers to control colors, fonts, spacing, positioning, and other visual aspects of a web page. For example, if you want to change the color of the heading from the previous HTML code, you would use CSS:

```
h1 {

  color: blue;

}
```

This CSS rule targets all <h1> tags and changes their text color to blue.

JavaScript is the programming language that enables web pages to be interactive. JavaScript allows developers to implement dynamic content, such as form validation, event handling, animations, and more. It runs on the client-side (in the browser), enabling real-time user interaction without the need to refresh the page. For example, a simple JavaScript function to display a message when a user clicks a button might look like this:

```
<button onclick="showMessage()">Click Me</button>



<script>

  function showMessage() {

    alert('Hello, welcome to my website!');

  }

</script>
```

In this example, when the button is clicked, the JavaScript function showMessage() is triggered, displaying an alert message.

Together, these technologies form the backbone of modern web development. HTTP facilitates the transfer of data, HTML structures the content, CSS styles it, and

JavaScript makes it interactive. Understanding these technologies is essential for any web developer.

Client-side vs Server-side Scripting

Web applications can be divided into two main types of scripting: **client-side** and **server-side**. These terms refer to where the code is executed—either on the user's device (client-side) or on the server (server-side). Both types of scripting are essential for building functional and dynamic websites, and each has its own role in web development.

Client-side scripting involves code that is executed directly in the user's browser. This code is typically written in languages like JavaScript, HTML, and CSS. Client-side scripts are responsible for creating an interactive user experience by manipulating the DOM (Document Object Model), responding to user events like clicks, form submissions, or mouse movements, and updating the content dynamically without reloading the entire page. For example, when a user fills out a form, client-side JavaScript can validate the input before the form is submitted to the server.

Client-side scripting has the advantage of reducing the load on the server, as much of the processing is handled on the user's device. It also provides a faster response time, as there is no need to send a request to the server for every user interaction. However, because client-side code runs on the user's browser, it can be more vulnerable to security risks like cross-site scripting (XSS) attacks. Therefore, sensitive operations should not be performed purely on the client side.

An example of client-side scripting is a simple form validation implemented in JavaScript:

```
function validateForm() {

  var name = document.forms["myForm"]["name"].value;

  if (name == "") {

    alert("Name must be filled out");

    return false;

  }

}
```

In this case, when the form is submitted, the JavaScript code checks whether the "name" field is empty. If it is, an alert is shown, and the form submission is prevented.

Server-side scripting, on the other hand, involves code that runs on the web server. This code is typically written in languages like PHP, Python, Ruby, or ASP.Net. Server-side scripts are responsible for handling complex tasks such as accessing databases, performing authentication, generating dynamic content, and processing form submissions. The server processes the code and sends the resulting output (usually HTML) to the client's browser.

An example of server-side scripting is a PHP script that retrieves data from a database and displays it on a webpage:

```
<?php

$servername = "localhost";

$username = "username";

$password = "password";

$dbname = "myDB";


// Create connection

$conn = new mysqli($servername, $username, $password, $dbname);


// Check connection

if ($conn->connect_error) {

  die("Connection failed: " . $conn->connect_error);

}


$sql = "SELECT id, firstname, lastname FROM Users";

$result = $conn->query($sql);
```

```
if ($result->num_rows > 0) {
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
?>
```

In this example, the PHP script connects to a MySQL database, retrieves data from a table, and displays it on the webpage. This is all done server-side, and the user only sees the resulting HTML.

The main advantage of server-side scripting is that it provides greater security and control over the application, as sensitive data and business logic are kept hidden from the user. However, server-side scripts typically require more server resources, and the user has to wait for a response from the server, which can result in slower load times.

EXERCISE

1. **Create a Simple Web Page**: Create a webpage that uses HTML, CSS, and JavaScript. Add a heading, a paragraph, and a button. When the button is clicked, display an alert message using JavaScript.
2. **Client-side Form Validation**: Build a form that includes fields like name, email, and phone number. Use JavaScript to validate that the fields are not empty before submitting the form.

CASE STUDY: BUILDING A SIMPLE TO-DO LIST APPLICATION

In this case study, we will build a simple to-do list application using client-side scripting with JavaScript and server-side scripting with PHP.

- **Client-side (JavaScript)**: The user will add new tasks to the list, remove completed tasks, and view the list dynamically, without needing to refresh the page.
- **Server-side (PHP)**: The tasks will be stored in a MySQL database. The server will handle adding new tasks, removing tasks, and fetching tasks from the database.

ASP.NET CORE FRAMEWORK

OVERVIEW OF ASP.NET CORE

ASP.Net Core is an open-source, cross-platform web framework developed by Microsoft. It enables the creation of modern, high-performance, cloud-based, and web applications. ASP.Net Core is designed to be modular, scalable, and lightweight, offering improved performance and flexibility compared to its predecessor, ASP.Net. It provides developers with the tools they need to create web applications and APIs that can run on Windows, Linux, and macOS.

ASP.Net Core is built to address the challenges posed by modern application development, such as cloud computing, microservices architecture, and cross-platform support. The framework allows developers to build responsive applications that can easily scale, while also providing full control over the application's pipeline. One of the key advantages of ASP.Net Core is its ability to run on multiple operating systems, making it a good choice for projects that require portability or need to be deployed in a cloud environment. It is also designed to be efficient, minimizing overhead and offering high throughput.

Some of the essential features of ASP.Net Core include:

- **Cross-platform compatibility**: ASP.Net Core can run on Windows, macOS, and Linux, providing a broad range of deployment options.
- **Modular structure**: The framework is built in a modular way, allowing developers to include only the necessary components for their applications, reducing the overall footprint.
- **High performance**: ASP.Net Core offers high throughput, especially with the Kestrel web server, which is designed for maximum performance.
- **Built-in dependency injection**: ASP.Net Core integrates dependency injection (DI) into the framework, promoting a more maintainable and testable codebase.
- **Unified web API and MVC**: Developers can create both web applications and APIs using the same framework, enabling flexibility and ease of use.

ASP.Net Core's design principles make it well-suited for modern cloud applications, real-time applications, microservices, and scalable web solutions. It is a powerful framework that has evolved to meet the demands of today's software development landscape.

Creating a Simple ASP.Net Core Application

Creating a simple ASP.Net Core application involves several key steps, from setting up the development environment to running the application on a web server. Let's go through the process of building a basic "Hello World" application using ASP.Net Core.

Setting Up the Development Environment

Before you can begin developing an ASP.Net Core application, you must set up your development environment. The essential components include:

- **.NET SDK**: You need to download and install the .NET SDK, which includes the runtime and tools required to develop ASP.Net Core applications.
- **Visual Studio or Visual Studio Code**: Visual Studio is a powerful IDE (Integrated Development Environment) for Windows, while Visual Studio Code is a lightweight, cross-platform editor that works on macOS, Linux, and Windows.

Once the development environment is set up, you can create a new ASP.Net Core project by using the command line or Visual Studio.

Steps to Create a Basic Web Application

1. **Create the project**: Open a terminal or Visual Studio and run the following command to create a new web application:
2. dotnet new webApp -n MyFirstAspNetCoreApp

This creates a simple web application template.

3. **Run the application**: To run the application, navigate to the project directory and use the following command:
4. cd MyFirstAspNetCoreApp
5. dotnet run

The application will start, and you can visit http://localhost:5000 in your browser to see the result.

6. **Modify the default page**: Open the Views/Home/Index.cshtml file and modify it to display a custom message, such as "Hello, ASP.Net Core!"

7. <h1>Hello, ASP.Net Core!</h1>

Save the file and refresh the browser. You should now see the updated message.

Key Concepts and Customization

In ASP.Net Core, everything is configured using a combination of C# code and configuration files, such as Startup.cs and appsettings.json. The Startup.cs file is where you configure services, middleware, and the request pipeline.

This simple application introduces the foundational concepts of ASP.Net Core, such as MVC architecture, routing, and views, which you will build on as you learn more advanced features.

MVC Architecture in ASP.Net Core

The Model-View-Controller (MVC) architecture is a software design pattern that divides an application into three interconnected components: the model, the view, and the controller. ASP.Net Core utilizes MVC architecture to create scalable, maintainable, and testable web applications.

Model

The **Model** represents the application's data and business logic. It is responsible for retrieving, storing, and manipulating data. Models can interact with a database, process user input, and ensure that the data is in the correct format.

In an ASP.Net Core MVC application, the model is typically represented by classes that correspond to database tables or business objects. The model can include validation logic and business rules to ensure that data remains consistent and accurate.

View

The **View** is the user interface (UI) component that displays data to the user. In ASP.Net Core, views are typically created using Razor, a markup syntax that combines HTML with C# code. Views receive data from the controller and display it in a format that is readable and user-friendly.

For example, a view might display a list of products from a database. The data is passed from the controller to the view, which then renders it in an HTML format.

CONTROLLER

The **Controller** acts as the intermediary between the Model and the View. It handles user requests, retrieves data from the Model, and passes it to the View for display. Controllers are responsible for processing user input, such as form submissions or button clicks, and triggering appropriate actions.

In an ASP.Net Core MVC application, controllers are defined as C# classes that inherit from Controller. The controller receives requests via HTTP actions and typically returns a view or a redirect to another action.

Example of MVC in Action

Consider a simple product catalog application that displays a list of products to the user. The following example shows how the MVC architecture works:

- **Model**: A Product class with properties like Id, Name, and Price.
- **Controller**: A ProductController class with an action method Index() that fetches the list of products from a database or in-memory collection.
- **View**: A Razor view that renders the list of products as an HTML table.

Controller:

```
public class ProductController : Controller

{

  public IActionResult Index()

  {

    var products = new List<Product>

    {

      new Product { Id = 1, Name = "Laptop", Price = 1000 },

      new Product { Id = 2, Name = "Smartphone", Price = 500 }

    };
    return View(products);
  }
}
```

View (Index.cshtml):

```
@model IEnumerable<Product>


<h1>Product List</h1>
<table>
  <tr>
    <th>Name</th>
    <th>Price</th>
  </tr>
  @foreach (var product in Model)
  {
    <tr>
      <td>@product.Name</td>
      <td>@product.Price</td>
    </tr>
  }
</table>
```

Exercises and Case Study

Exercise:

1. Create a simple ASP.Net Core application that displays a list of books, including their title, author, and price. Use MVC architecture to implement the Model, View, and Controller. The data can be hardcoded for simplicity.

2. Modify the application to allow users to add new books. Create a form in the View, and implement the logic in the Controller to handle POST requests.

Case Study:

Company XYZ wants to build a simple e-commerce web application to sell products online. The company decides to use ASP.Net Core MVC architecture for the application. The application will display product categories, allow users to add items to the cart, and process payments.

Your task is to design the application using MVC principles. Define the models, views, and controllers that would be required to implement the product catalog, shopping cart, and checkout process.

Assignment Solution: Create a Basic ASP.Net Core Application and Deploy It to a Local Server

In this assignment, you will create a basic ASP.Net Core application and deploy it to a local server. Follow the steps below for a detailed, step-by-step guide on how to complete this assignment.

Prerequisites

Before starting, ensure you have the following installed on your machine:

- **Visual Studio** (latest version recommended) or **Visual Studio Code** with the .NET SDK.
- **.NET SDK** (ASP.Net Core requires the .NET SDK, which includes the necessary tools for building applications).
- **IIS Express** or a similar local server (IIS Express is installed with Visual Studio).

Step-by-Step Guide

Step 1: Install .NET SDK

1. Go to the official Microsoft .NET download page and download the latest .NET SDK.
2. Run the installer and follow the prompts to install the .NET SDK on your machine.

Step 2: Create a New ASP.Net Core Application

1. **Open Visual Studio**:
    - If you don't have Visual Studio installed, you can download it from here. Make sure to install the **ASP.Net and Web Development** workload.

2. **Create a New Project**:
   - Open Visual Studio and click on **Create a New Project**.
   - In the search bar, type ASP.NET Core Web Application and select it.
   - Click **Next**.
3. **Configure Your Project**:
   - Give your project a name (e.g., **BasicAspNetCoreApp**).
   - Choose the location where you want to save your project.
   - Select **Create**.
4. **Select a Project Template**:
   - Choose the **Web Application (Model-View-Controller)** template, which will create a basic MVC application.
   - Make sure **.NET Core** and the latest **ASP.Net Core version** are selected.
   - Click **Create**.

STEP 3: BUILD YOUR APPLICATION

1. **Open the Solution**:
   - Once the project is created, Visual Studio will automatically open the solution.
2. **Run the Application Locally**:
   - Press **Ctrl + F5** to run the application locally. This will launch the app in a browser window using **IIS Express**.
3. **Customize the Application**:
   - You can modify the default HomeController.cs and Views to add additional pages or content.
   - For example, modify HomeController.cs to return a custom message:

```
public class HomeController : Controller
{
  public IActionResult Index()
  {
    ViewData["Message"] = "Welcome to My Basic ASP.Net Core App!";
    return View();
  }
}
```

4. **Build the Application**:
   - Once your application is working as expected, ensure the application builds without errors by clicking **Build** from the top menu and selecting **Build Solution**.

Step 4: Deploy the Application to a Local Server

You will deploy the application using **IIS Express** which comes with Visual Studio, or you can use **IIS** (Internet Information Services) for a more permanent local server deployment.

Option 1: Using IIS Express (with Visual Studio)

1. **Run the Application with IIS Express**:
   - If your application is running locally on **IIS Express** (which is the default option in Visual Studio), you can easily deploy it by clicking **Start Debugging** (F5) or **Ctrl + F5**.
   - IIS Express automatically runs the application locally, and you can view it by navigating to the URL provided in the browser (usually something like http://localhost:5000).

Option 2: Deploy to Local IIS Server

1. **Install IIS (Internet Information Services)**:

- If IIS is not installed on your machine, open **Control Panel > Programs and Features > Turn Windows features on or off**.
- Check **Internet Information Services (IIS)** and ensure that the **Web Management Tools** and **World Wide Web Services** are selected.
- Click **OK** to install IIS.

2. **Publish Your Application**:
   - In Visual Studio, right-click on the project name in **Solution Explorer** and select **Publish**.
   - Choose **Folder** as the deployment target and specify a folder path (e.g., C:\inetpub\wwwroot\BasicAspNetCoreApp).
   - Click **Publish** to copy your files to the folder.
3. **Configure IIS to Host the Application**:
   - Open **Internet Information Services (IIS) Manager**.
   - Right-click on **Sites** in the left panel and select **Add Website**.
   - In the **Add Website** window, set the following:
     - **Site name**: Enter a name for your site (e.g., **BasicAspNetCoreApp**).
     - **Physical path**: Browse to the folder where you published the application (e.g., C:\inetpub\wwwroot\BasicAspNetCoreApp).
     - **Port**: Choose a port number (e.g., 8080).
   - Click **OK** to create the site.
4. **Access the Application in a Browser**:
   - Open a web browser and go to http://localhost:8080.
   - You should now see your deployed ASP.Net Core application running on your local IIS server.

STEP 5: TESTING AND DEBUGGING

- **Test the Application**:
    - Ensure that the website is running properly by navigating to the local server's URL.
    - Check that all pages are loading correctly, and the content is displayed as expected.
- **Debugging**:
    - If any issues arise, check the **Output** and **Error List** windows in Visual Studio for any errors during publishing or runtime.
    - Ensure that all configurations for the IIS server are correct, including the **Application Pool** settings (use **.NET Core** as the runtime).

CONCLUSION

By following these steps, you've created a basic ASP.Net Core application and deployed it to a local server using IIS Express or a permanent local IIS server. This exercise familiarizes you with the full cycle of web application development, from building the application to deploying it for local testing. You can expand upon this foundation by exploring more complex features, adding functionality, and hosting the application on remote servers for broader access.

EXERCISE

1. Modify your basic ASP.Net Core application to add a new page where users can input their name and display a personalized greeting.
2. Experiment with deploying the application using different configurations (IIS Express vs. IIS).