TUẦN 1. GIỚI THIỆU TỔNG QUAN

Mục tiêu: Sau bài học này, sinh viên có thể:

- Giải thích được kiến trúc **Client-Server**, các thành phần cơ bản của ứng dụng web và quy trình xử lý **HTTP**.
- Trình bày được vai trò và lợi ích của mô hình **MVC** trong phát triển web.
- Nắm vững các khái niệm cơ bản của **C#** (OOP, LINQ, Async/Await) để chuẩn bị cho lập lập trình ASP.NET Core.
- Khởi tạo và chạy thành công một dự án ASP.NET Core MVC đầu tiên.

1.1. Tổng quan về ứng dụng Web và quy trình xử lý thông tin

1.1.1. Mô hình Client-Server

- **Khái niệm:** Web hoạt động dựa trên mô hình Client-Server. **Client** (máy khách) là trình duyệt web (Chrome, Firefox, Edge, Safari...) gửi yêu cầu. **Server** (máy chủ) là máy tính chạy các phần mềm chuyên dụng để xử lý yêu cầu và trả về phản hồi.
- Sơ đồ:

```
+---------+           Request (HTTP/HTTPS)          +--------------+
|  Client | <------------------------------------- |    Server    |
|(Browser)| Response (HTML, CSS, JS, Images, Data) | (Web Server) |
+---------+ -------------------------------------> +--------------+
```

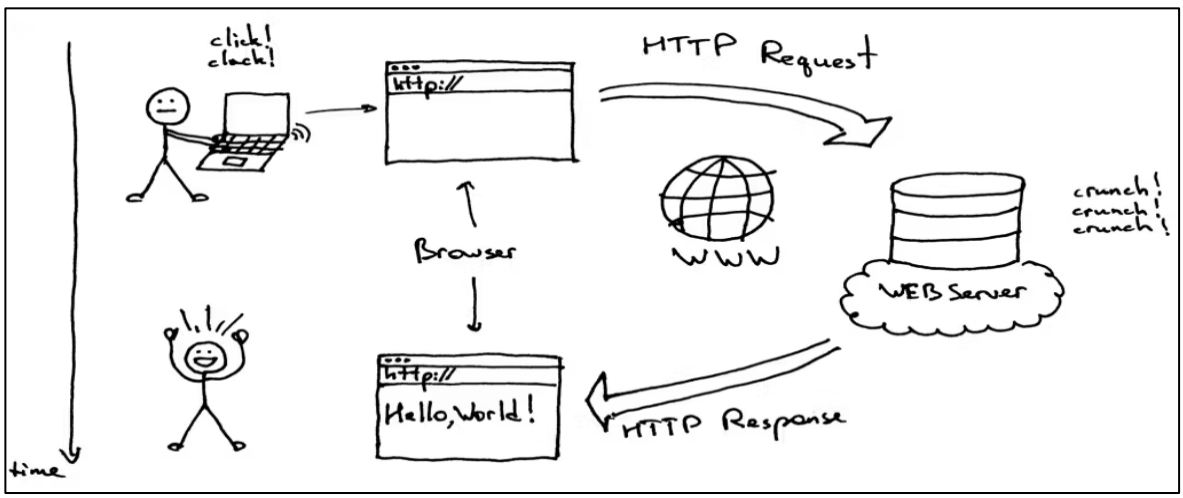


Figure 1-1: Minh họa mô hình Client-Server trong ứng dụng web

- **Ví dụ trong "Cửa hàng Thể thao":** Khi bạn gõ địa chỉ `cuahangthethao.com` vào trình duyệt và nhấn Enter, trình duyệt của bạn là **Client**. Nó gửi một yêu cầu

(Request) đến máy chủ của cửa hàng. Máy chủ xử lý yêu cầu và gửi về phản hồi (Response) là trang chủ của cửa hàng (HTML, CSS, hình ảnh sản phẩm).

1.1.2. Giao thức HTTP và HTTPS

- **HTTP (Hypertext Transfer Protocol):** Giao thức nền tảng để truyền tải dữ liệu trên Web. Đây là giao thức **không trạng thái (stateless)**, nghĩa là mỗi yêu cầu đều độc lập và máy chủ không lưu trữ thông tin về các yêu cầu trước đó.
- **HTTPS (Hypertext Transfer Protocol Secure):** Phiên bản bảo mật của HTTP, sử dụng mã hóa **SSL/TLS** để bảo vệ dữ liệu truyền tải, rất quan trọng cho các trang có thông tin nhạy cảm (thanh toán, đăng nhập).
- Yêu cầu (Request):
  - Gồm: Phương thức HTTP (GET, POST, PUT, DELETE...), URL, Headers (Content-Type, User-Agent...), Body (dữ liệu gửi đi, thường dùng cho POST/PUT).
- Phản hồi (Response):
  - Gồm: Mã trạng thái (200 OK, 404 Not Found, 500 Internal Server Error...), Headers (Content-Type...), Body (dữ liệu trả về, ví dụ HTML, JSON).

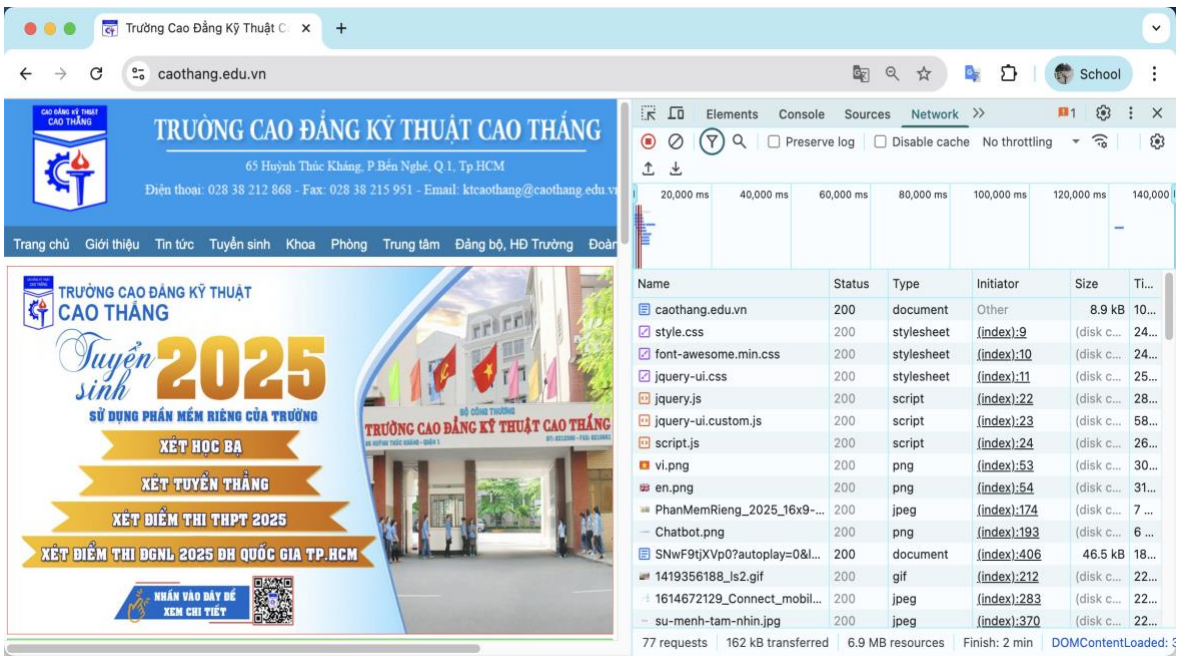


Figure 1-2: Cấu trúc một HTTP Request và Response hiển thị trong trình duyệt (tab Network)

- Ví dụ trong "Cửa hàng Thể thao":
  - Khi bạn xem danh sách sản phẩm, trình duyệt gửi yêu cầu `GET /Products` đến server. Server trả về danh sách sản phẩm (200 OK).

- Khi nhấn "Thêm vào giỏ hàng", trình duyệt gửi yêu cầu `POST /Cart/AddToCart` với dữ liệu sản phẩm. Server xử lý và trả về phản hồi tương ứng.
- Việc thanh toán sẽ luôn được thực hiện qua **HTTPS** để bảo mật thông tin thẻ tín dụng của khách hàng.

1.1.3. Các thành phần cơ bản của một ứng dụng web

– **Front-end (Phía Client):** Phần giao diện người dùng, chạy trên trình duyệt, gồm:
  - **HTML:** Ngôn ngữ đánh dấu cấu trúc nội dung trang.
  - **CSS:** Ngôn ngữ định kiểu, thiết kế giao diện (màu sắc, bố cục, font chữ).
  - **JavaScript:** Ngôn ngữ lập trình để tạo các tương tác động trên trang (kiểm tra form, hiệu ứng...).
– **Back-end (Phía Server):** Phần xử lý logic nghiệp vụ, lưu trữ dữ liệu, giao tiếp với cơ sở dữ liệu. Bao gồm:
  - **Ngôn ngữ lập trình:** C# (ASP.NET Core), Java, Python, Node.js, PHP...
  - **Web Framework:** ASP.NET Core MVC, Spring Boot, Django, Express.js...
  - **Cơ sở dữ liệu (Database):** SQL Server, MySQL, PostgreSQL, MongoDB...

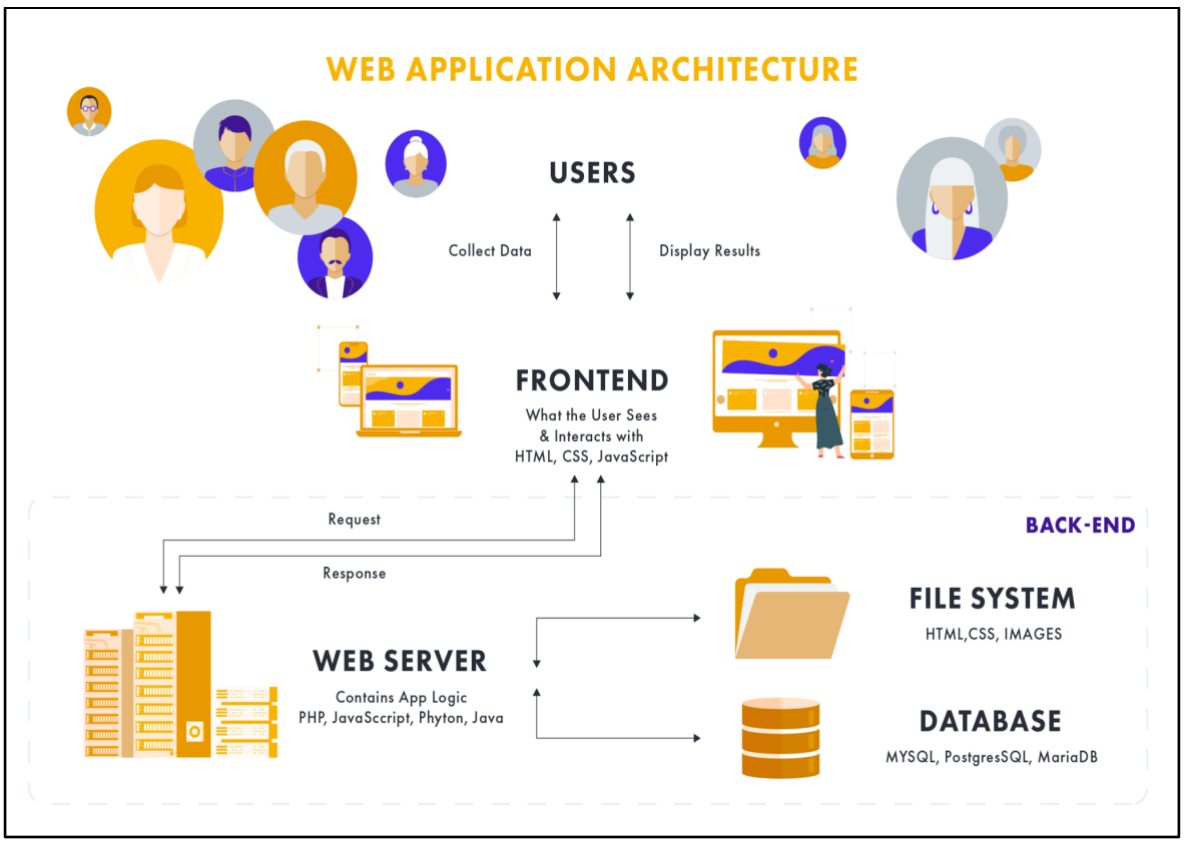


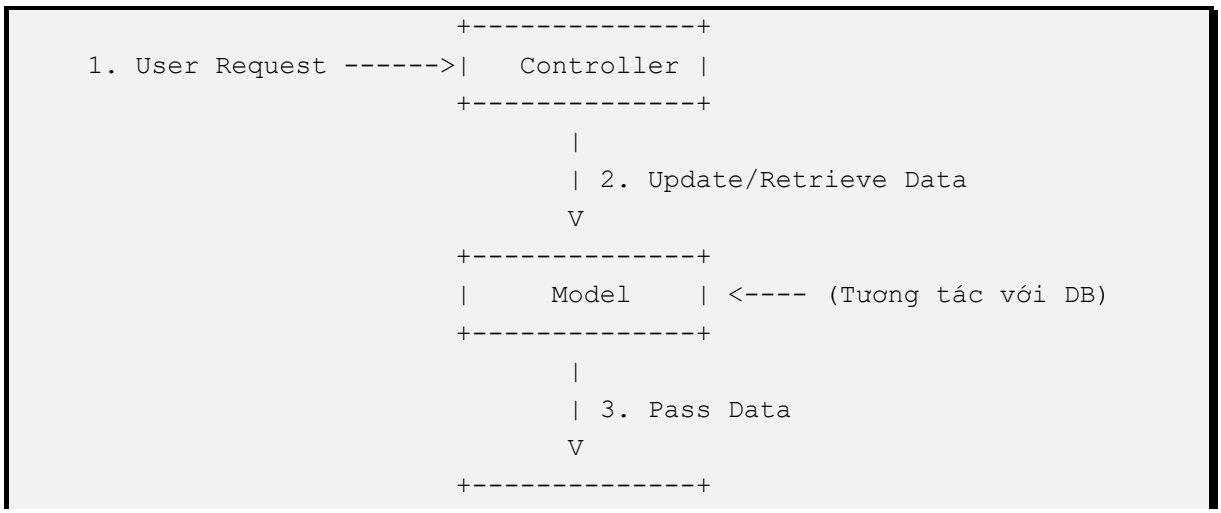
Figure 1-3: Web Application Architecture Diagram

- Ví dụ trong "Cửa hàng Thể thao":
  - **Front-end:** Trang chủ với các sản phẩm được trình bày bằng HTML, CSS (Bootstrap), và JavaScript cho các hiệu ứng thêm vào giỏ hàng.
  - **Back-end:** ASP.NET Core MVC xử lý yêu cầu thêm sản phẩm, kết nối với SQL Server để lưu trữ thông tin giỏ hàng và đơn hàng.

1.2. Giới thiệu mô hình kiến trúc MVC (Model-View-Controller)

1.2.1. Khái niệm và lợi ích

- **MVC** là một mẫu kiến trúc phần mềm phổ biến, giúp tách biệt các thành phần của ứng dụng web thành ba vai trò chính:
  - **Model (Mô hình):** Biểu diễn dữ liệu và logic nghiệp vụ.
  - **View (Giao diện):** Hiển thị dữ liệu cho người dùng.
  - **Controller (Bộ điều khiển):** Xử lý các yêu cầu của người dùng, tương tác với Model và chọn View phù hợp để hiển thị.
- Lợi ích của MVC:
  - **Tách biệt mối quan tâm:** Mỗi thành phần chỉ chịu trách nhiệm cho một phần cụ thể, giúp mã dễ quản lý, bảo trì và mở rộng.
  - **Tái sử dụng mã:** Logic nghiệp vụ trong Model có thể được tái sử dụng bởi nhiều Controller hoặc View.
  - **Dễ kiểm thử:** Việc tách biệt giúp dễ dàng kiểm thử từng phần độc lập.
  - **Phân công công việc rõ ràng:** Các lập trình viên có thể làm việc trên các phần khác nhau của ứng dụng mà ít bị chồng chéo.
- Sơ đồ luồng hoạt động của MVC:



```
                          |     View     |
                          +--------------+
                                 |
  4. Display to User <-----------+
```

- Ví dụ trong "Cửa hàng Thể thao":
  - **Model:** Lớp `Product` (chứa Id, Name, Price, Description...), lớp `Category`.
  - **View:** Trang `List.cshtml` hiển thị danh sách sản phẩm.
  - **Controller:** `ProductController` xử lý yêu cầu xem danh sách sản phẩm.
  - Luồng hoạt động:
    1. Người dùng yêu cầu xem sản phẩm theo danh mục "Bóng đá" (`/Product/List?category=Football`).
    2. `ProductController` nhận yêu cầu, yêu cầu `Product Model` lấy các sản phẩm thuộc danh mục "Bóng đá" từ cơ sở dữ liệu.
    3. `Product Model` trả về dữ liệu sản phẩm.
    4. `ProductController` chuyển dữ liệu đó cho `List View`.
    5. `List View` hiển thị các sản phẩm lên trình duyệt.

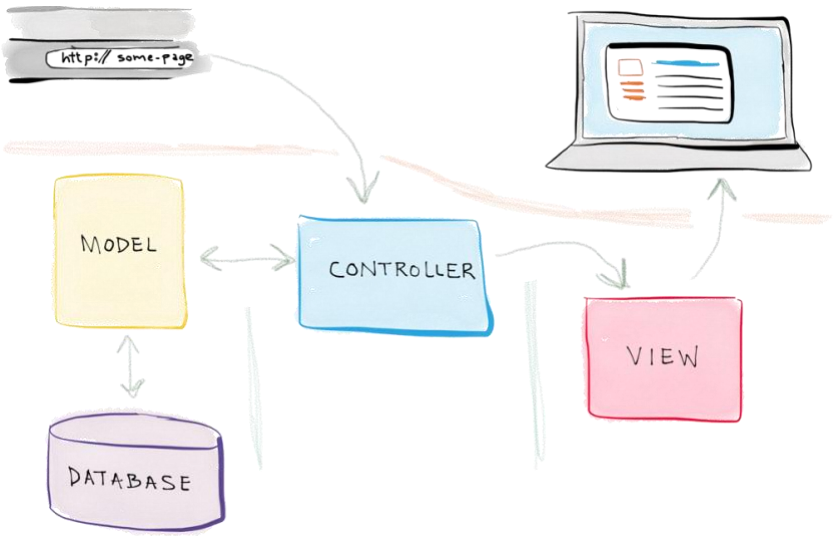


Figure 1-4: Sơ đồ luồng hoạt động của mô hình MVC

1.3. Tổng quan về ASP.NET Core

1.3.1. ASP.NET Core là gì?

- Là một Framework mã nguồn mở và đa nền tảng để xây dựng các ứng dụng web hiện đại (web apps, APIs, microservices) của Microsoft.
- Là thế hệ tiếp theo của ASP.NET Framework.

- **ASP.NET Core MVC** là một phần của ASP.NET Core, tập trung vào việc xây dựng ứng dụng web theo mô hình MVC.

1.3.2. Ưu điểm nổi bật:

- **Đa nền tảng (Cross-platform):** Chạy được trên Windows, macOS, Linux.
- **Mã nguồn mở (Open-source):** Cộng đồng lớn, minh bạch, có thể đóng góp.
- **Hiệu suất cao (High performance):** Được thiết kế để tối ưu tốc độ và khả năng mở rộng.
- **Kiến trúc module (Modular architecture):** Chỉ bao gồm các thành phần cần thiết, giúp ứng dụng nhẹ và nhanh hơn.
- **Hỗ trợ Dependency Injection tích hợp sẵn:** Giúp xây dựng ứng dụng dễ bảo trì và kiểm thử.
- Tích hợp tốt với các công nghệ Front-end hiện đại: Angular, React, Vue.js.

1.3.3. Phiên bản .NET 8 LTS (Long Term Support):

- Bài giảng này sẽ sử dụng .NET 8, phiên bản hỗ trợ dài hạn mới nhất. Điều này đảm bảo bạn học được những tính năng, cải tiến và quy tắc tốt nhất hiện nay.
- **Lưu ý:** Từ .NET 6 trở lên, file `Startup.cs` và `Program.cs` đã được hợp nhất vào một file `Program.cs` duy nhất, giúp cấu hình ứng dụng gọn gàng hơn.

Figure 1-5: Logo .NET 8 LTS và các ưu điểm chính của ASP.NET Core

1.4. Giới thiệu các công cụ phát triển

- **Visual Studio 2022:** Môi trường phát triển tích hợp (IDE) mạnh mẽ nhất cho .NET trên Windows. Cung cấp đầy đủ công cụ để viết code, debug, quản lý dự án.

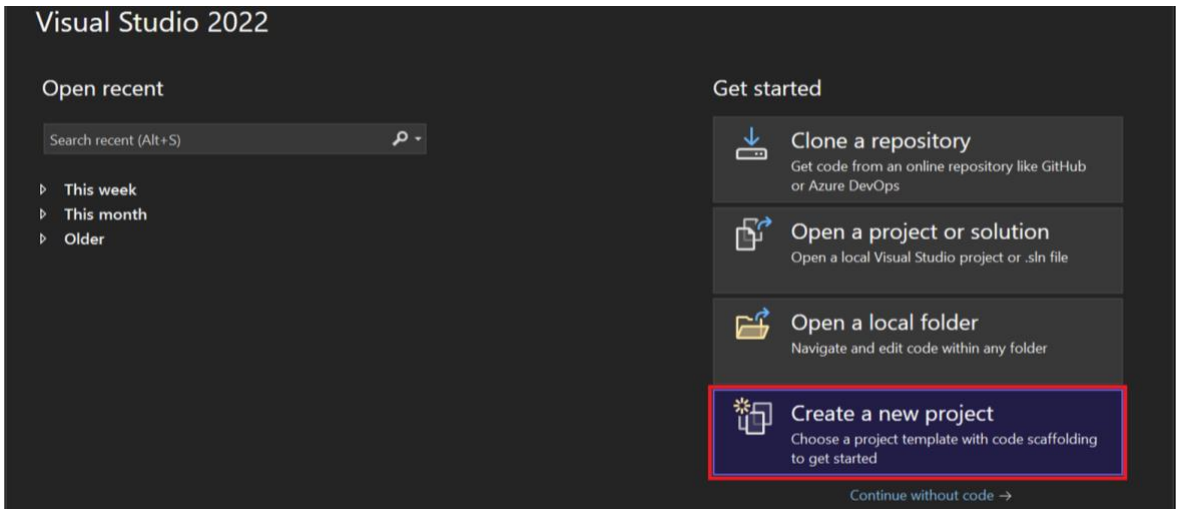


Figure 1-6: Giao diện môi trường phát triển Visual Studio 2022

– **Visual Studio Code:** Trình soạn thảo mã nguồn nhẹ, đa nền tảng, mạnh mẽ. Phù hợp cho cả Windows, macOS, Linux. Cần cài thêm các extension cho C# và .NET.

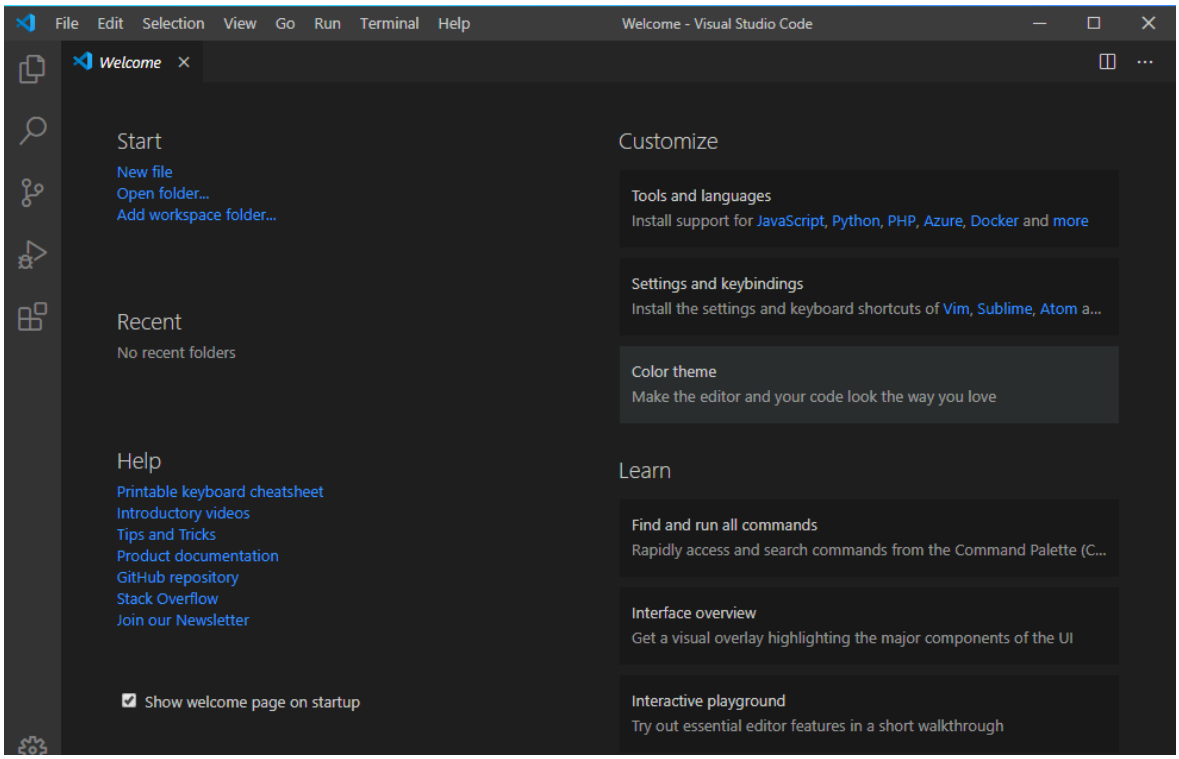


Figure 1-7: Giao diện trình soạn thảo mã nguồn Visual Studio Code

– **.NET CLI (Command Line Interface):** Công cụ dòng lệnh để tạo, build, chạy, publish các dự án .NET. Rất hữu ích cho các tác vụ tự động hóa hoặc khi làm việc trên môi trường terminal.

- **Ví dụ lệnh:** `dotnet new mvc`, `dotnet run`, `dotnet build`, `dotnet –list-sdks`

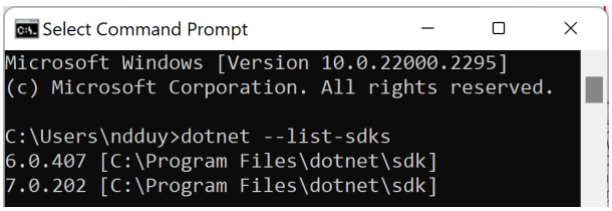


Figure 1-8: Sử dụng .NET CLI để kiểm tra phiên bản .NET SDK

1.5. Giới thiệu ứng dụng mẫu "Cửa hàng Thể thao" (Sports Store)

Để minh họa các khái niệm và kỹ thuật xuyên suốt môn học, chúng ta sẽ xây dựng một ứng dụng web mô phỏng "Cửa hàng Thể thao" với các chức năng chính:

- **Hiển thị danh sách sản phẩm:** Cho phép người dùng xem các sản phẩm thể thao theo danh mục, có phân trang.
- **Chi tiết sản phẩm:** Xem thông tin chi tiết của từng sản phẩm.
- **Giỏ hàng:** Cho phép người dùng thêm/xóa sản phẩm vào giỏ hàng, cập nhật số lượng.
- **Đặt hàng:** Quá trình thanh toán và đặt hàng.
- **Quản trị sản phẩm (Admin Panel):** Dành cho quản trị viên để thêm, sửa, xóa sản phẩm, quản lý danh mục.
- **Hệ thống người dùng:** Đăng ký, đăng nhập, quản lý vai trò (User, Admin).

Ví dụ minh họa giao diện dự kiến:

- Trang chủ:

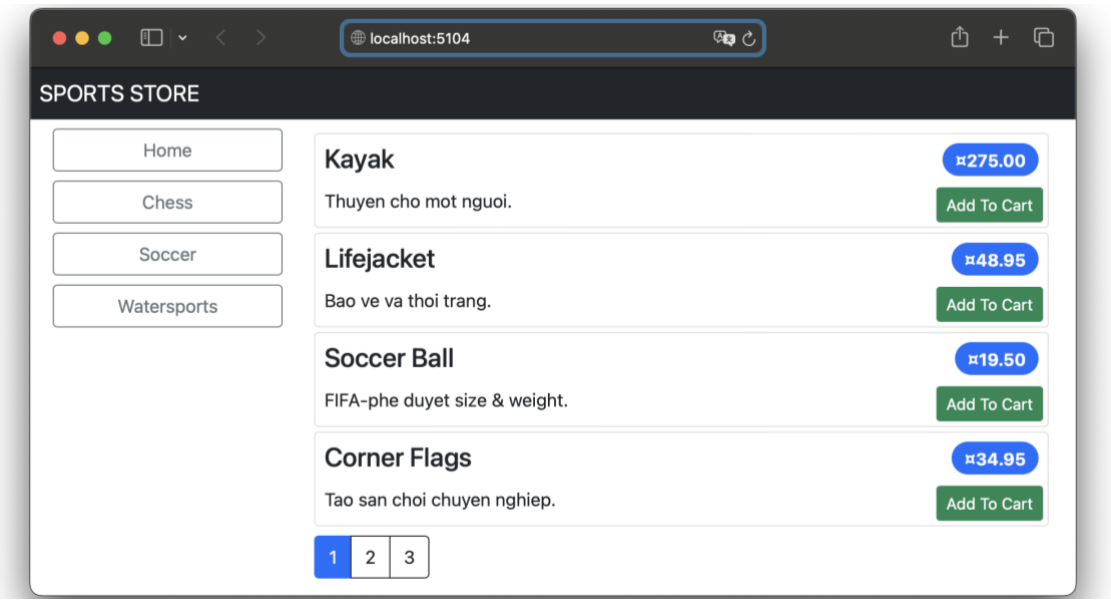


Figure 1-9: Giao diện dự kiến của trang chủ 'Cửa hàng Thể thao'

- Trang quản trị:

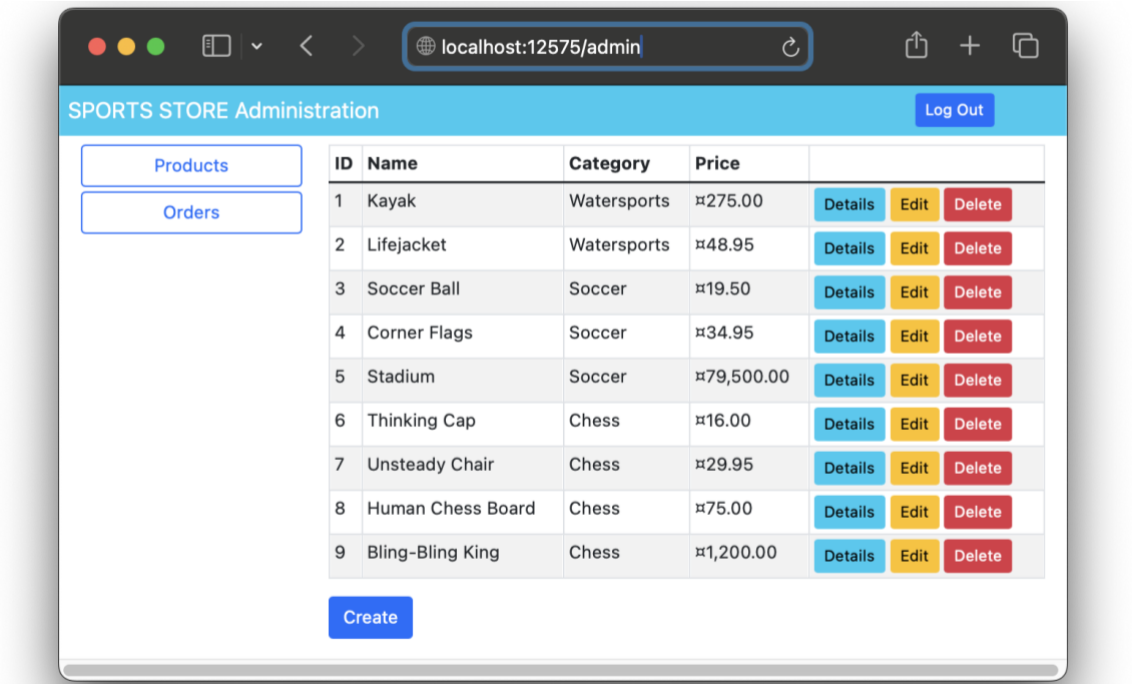


Figure 1-10: Giao diện dự kiến của trang quản trị

– Trang giỏ hàng:

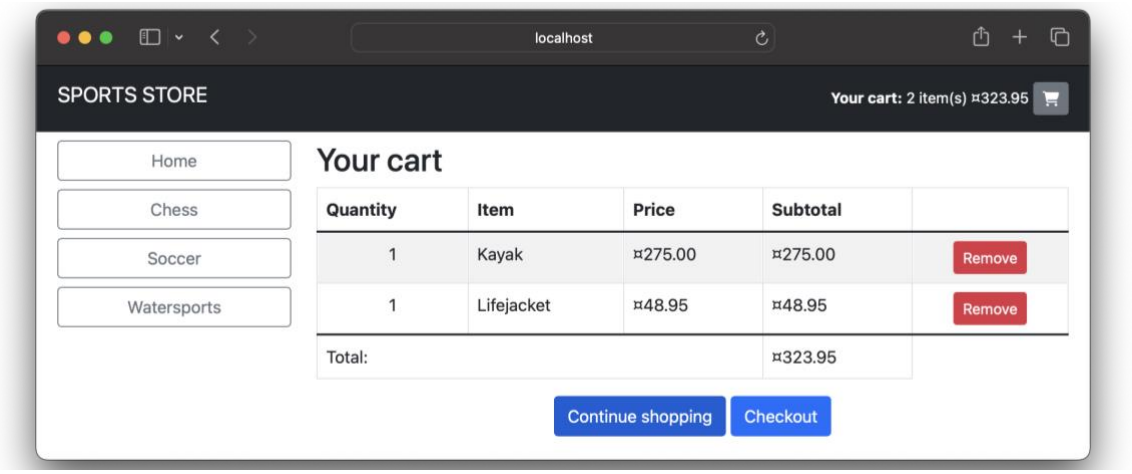


Figure 1-11: Giao diện dự kiến của trang giỏ hàng

1.6. Ngôn ngữ C# cơ bản và các khái niệm cần thiết

ASP.NET Core được xây dựng trên C#. Để học ASP.NET Core hiệu quả, bạn cần nắm vững các khái niệm C# sau:

1.6.1. Lập trình hướng đối tượng (OOP)

– Class và Object:

- **Class (Lớp):** Là một bản thiết kế, khuôn mẫu để tạo ra các đối tượng. Định nghĩa các thuộc tính (dữ liệu) và phương thức (hành vi) mà đối tượng sẽ có.
- **Object (Đối tượng):** Là một thể hiện cụ thể (instance) của một lớp.

- **Ví dụ trong "Cửa hàng Thể thao":**

```
// Định nghĩa Class Product
public class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; } = string.Empty; // Thuộc tính
    public string Description { get; set; } = string.Empty;
    public decimal Price { get; set; }
    public string Category { get; set; } = string.Empty;

    // Constructor (Phương thức khởi tạo)
    public Product(int id, string name, decimal price, string category)
    {
        ProductID = id;
        Name = name;
        Price = price;
        Category = category;
    }

    // Phương thức (Hành vi)
    public void DisplayInfo()
    {
        Console.WriteLine($"ID: {ProductID}, Name: {Name}, Price:
{Price:C}, Category: {Category}");
    }
}

// Tạo Object từ Class Product
Product p1 = new Product(1, "Bóng đá Adidas", 25.00m, "Bóng đá");
// Một Object Product
Product p2 = new Product(2, "Áo đấu đội tuyển A", 50.00m, "Quần áo");
// Một Object Product khác
```

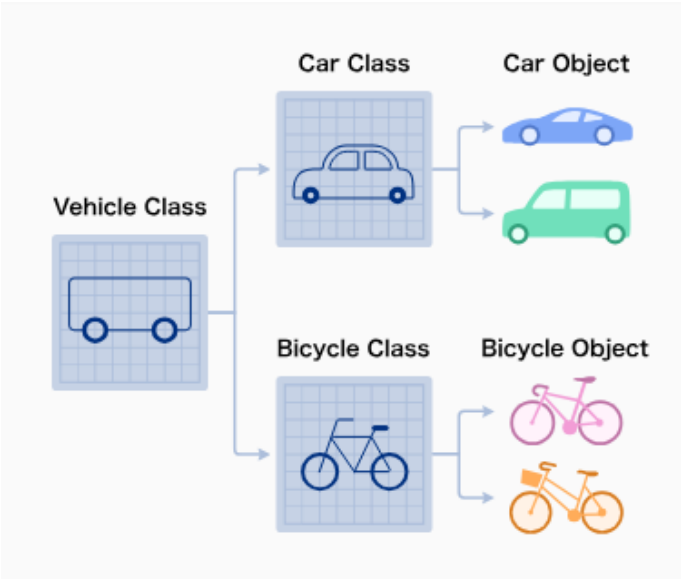


Figure 1-12: Mối quan hệ giữa Class và Object trong OOP

- **Inheritance (Kế thừa):** Cho phép một lớp (lớp con) kế thừa các thuộc tính và phương thức từ một lớp khác (lớp cha), thúc đẩy tái sử dụng mã.
- **Polymorphism (Đa hình):** Cho phép các đối tượng của các lớp khác nhau có thể được coi là đối tượng của một lớp chung, và có thể phản ứng khác nhau với cùng một thông điệp (phương thức).
- **Interface (Giao diện):** Định nghĩa một tập hợp các phương thức mà một lớp phải triển khai. Giúp đạt được tính trừu tượng và tách biệt.
- **Abstract Class (Lớp trừu tượng):** Lớp không thể tạo đối tượng trực tiếp, dùng để định nghĩa các phương thức trừu tượng mà lớp con phải triển khai.

1.6.2. LINQ (Language Integrated Query)

- Cho phép viết các truy vấn dữ liệu trực tiếp trong C# mà không cần dùng chuỗi SQL hay các ngôn ngữ truy vấn khác.
- Hoạt động với nhiều nguồn dữ liệu (Collections, XML, Entity Framework Core).
- Ví dụ LINQ trong "Cửa hàng Thể thao":

```
// Giả sử có một danh sách sản phẩm
List<Product> products = new List<Product>
{
    new Product(1, "Bóng đá Adidas", 25.00m, "Bóng đá"),
    new Product(2, "Áo đấu đội tuyển A", 50.00m, "Quần áo"),
    new Product(3, "Vợt cầu lông Yonex", 120.00m, "Cầu lông"),
    new Product(4, "Bóng chuyền Mikasa", 30.00m, "Bóng chuyền")
};

// Lọc các sản phẩm có giá trên 40.00
var expensiveProducts = from p in products
                        where p.Price > 40.00m
                        select p;

// Hoặc dùng cú pháp Method Syntax (phổ biến hơn trong ASP.NET Core)
var footballProducts = products.Where(p => p.Category == "Bóng
đá").ToList();

foreach (var p in footballProducts)
{
    Console.WriteLine(p.Name); // Output: Bóng đá Adidas
}
```

1.6.3. Asynchronous programming (async/await)

- Cho phép thực hiện các thao tác tốn thời gian (ví dụ: truy vấn cơ sở dữ liệu, gọi API) mà không làm chặn luồng chính của ứng dụng, giúp ứng dụng phản hồi nhanh hơn.
- Sử dụng từ khóa `async` cho phương thức và `await` cho các thao tác bất đồng bộ.
- **Ví dụ trong "Cửa hàng Thể thao":** Khi lấy danh sách sản phẩm từ cơ sở dữ liệu, việc này có thể tốn vài giây. Sử dụng `async/await` giúp ứng dụng không bị "đóng băng" trong thời gian chờ đợi.

```
// Dùng Async/Await:
public async Task<Product> GetProductByIdAsync(int id)
{ /* ... logic DB ... */
     return await Task.FromResult(new Product(id, "Test", 10, "Cat"));
} // Giả lập async

// Trong Controller (sẽ học sau)
// public class ProductController : Controller
// {
//     private readonly IProductRepository _productRepository;
//     public ProductController(IProductRepository productRepository) {
//         _productRepository = productRepository;
//     }

//     public async Task<IActionResult> Details(int id)
//     {
//         Product product = await
_productRepository.GetProductByIdAsync(id);
//         if (product == null) {
//             return NotFound();
//         }
//         return View(product);
//     }
// }
```

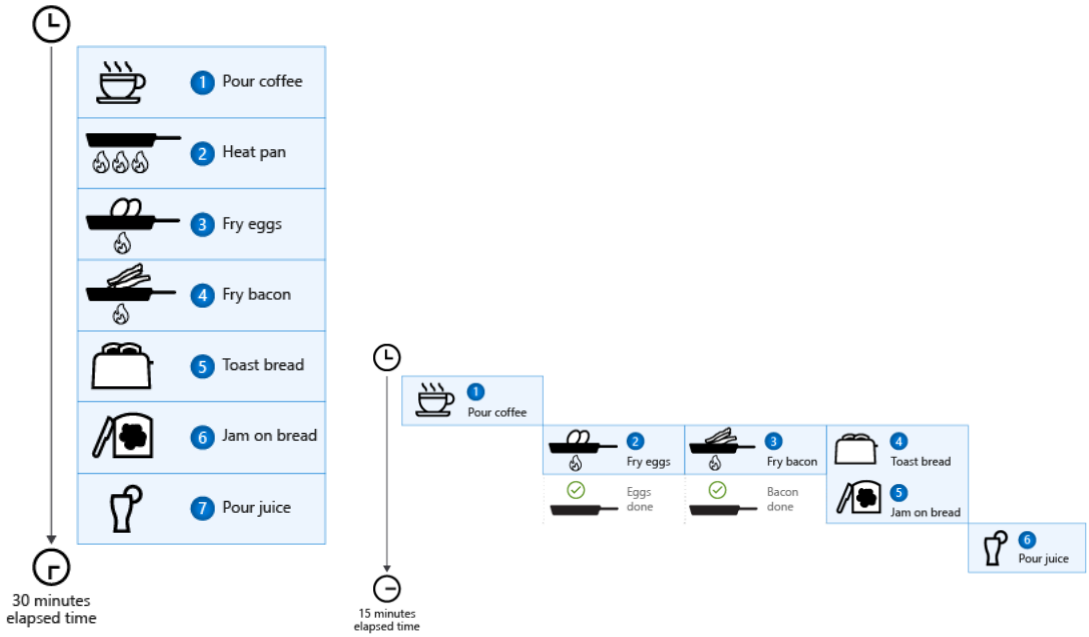


Figure 1-13: Mô hình lập trình bất đồng bộ với async/await[1]

[1] https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/

THỰC HÀNH 01: CÀI ĐẶT MÔI TRƯỜNG PHÁT TRIỂN

Mục tiêu thực hành: Cài đặt môi trường, tạo dự án ASP.NET Core MVC cơ bản, chạy thử, và thực hành các khái niệm C# nền tảng.

Các bước thực hiện:

1.1. Cài đặt môi trường phát triển

– **.NET SDK (phiên bản .NET 8):** Truy cập trang chủ của Microsoft để tải và cài đặt .NET SDK phù hợp với hệ điều hành (Windows, macOS, Linux).

- **Link tải:** `https://dotnet.microsoft.com/download/dotnet/8.0`

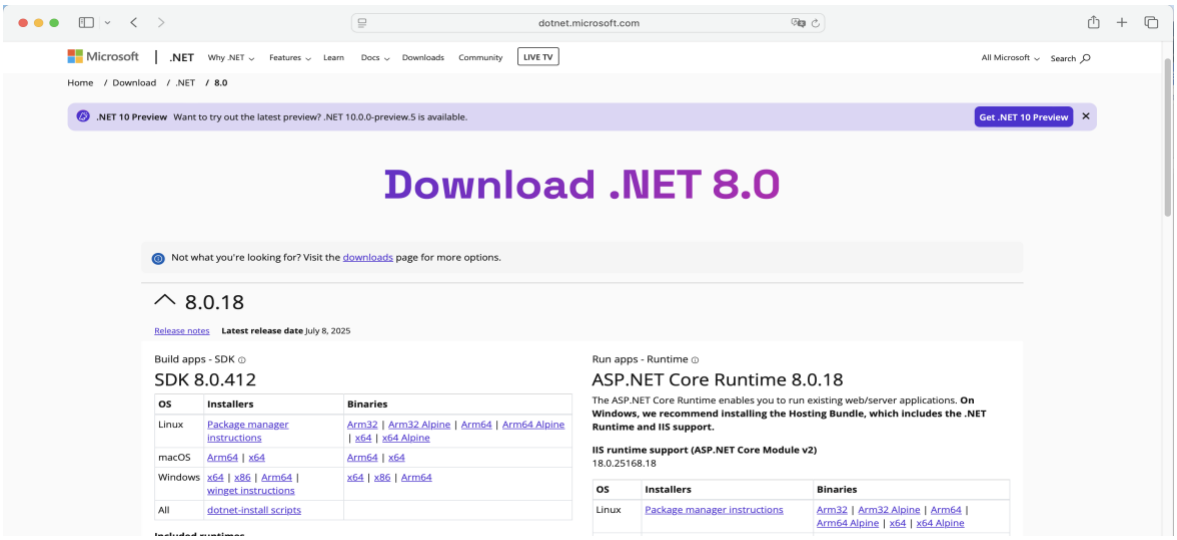


Figure 1-14: Trang tải xuống .NET 8 SDK

- **Kiểm tra cài đặt:** Mở `Command` Prompt/Terminal, gõ `dotnet --version`. Đảm bảo hiển thị phiên bản 8.x.x.

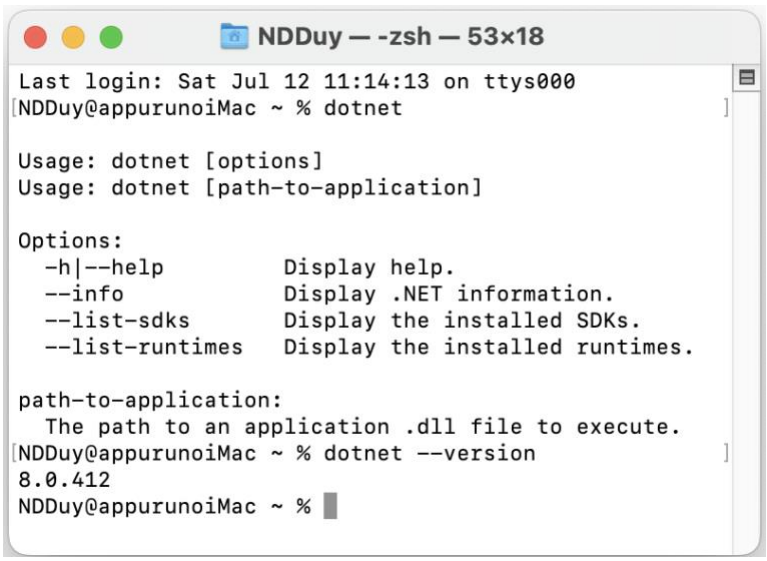


Figure 1-15: Kiểm tra phiên bản .NET SDK đã cài đặt

– **Visual Studio 2022 (Community Edition):** Hướng dẫn tải và cài đặt Visual Studio 2022 Community Edition. Đảm bảo chọn các workload cần thiết như "ASP.NET and web development" và ".NET desktop development".

- **Link tải:** `https://visualstudio.microsoft.com/downloads/`

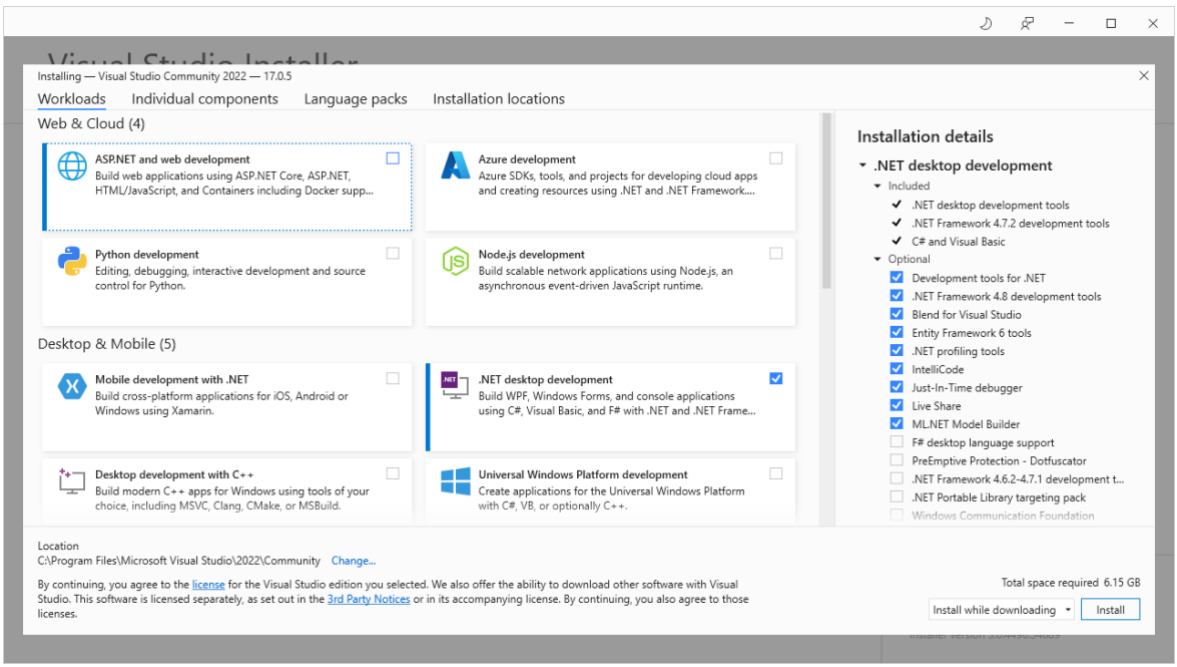


Figure 1-16: Chọn các Workload cần thiết khi cài đặt Visual Studio 2022

– **(Tùy chọn) Visual Studio Code:** Nếu sinh viên muốn dùng VS Code, hướng dẫn cài đặt VS Code và các extension cần thiết (C#, .NET Core SDK).

- **Link tải:** `https://code.visualstudio.com/`

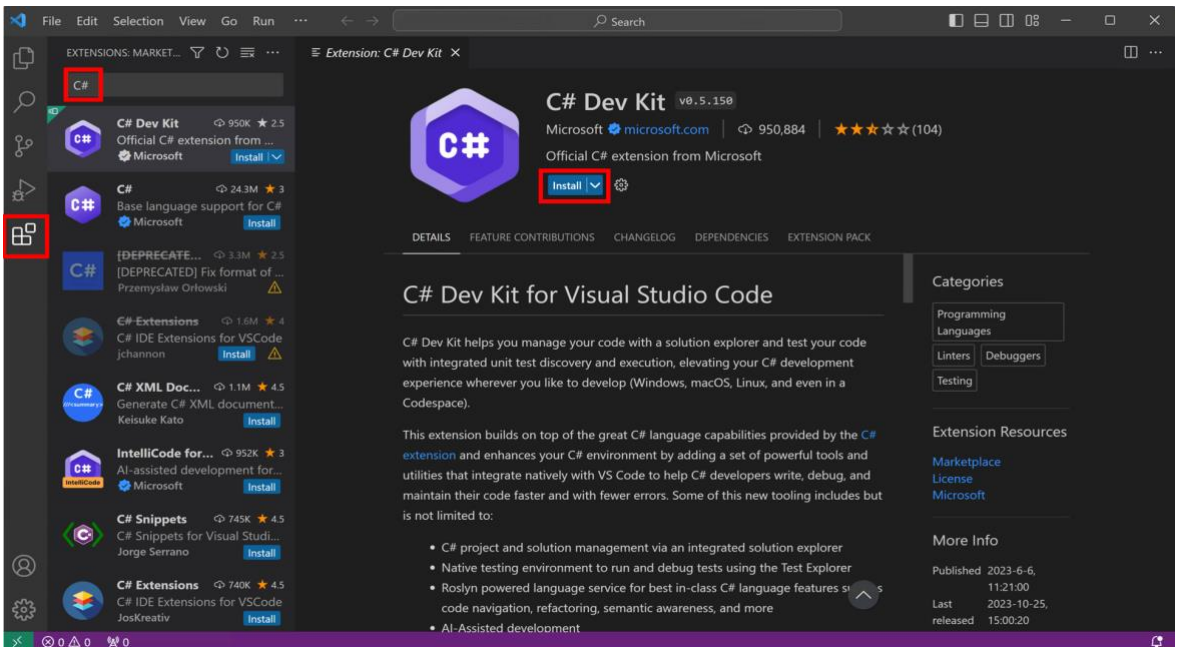


Figure 1-17: Cài đặt các Extension cần thiết cho Visual Studio Code

1.2. Hướng dẫn tạo dự án ASP.NET Core MVC đầu tiên

Cách 1: Sử dụng .NET CLI (Khuyến nghị cho việc hiểu cấu trúc):

- Mở Command Prompt/Terminal.
- Tạo thư mục cho dự án:

```
mkdir SportsStore
cd SportsStore
```

- Tạo dự án MVC mới:

```
dotnet new mvc -n SportsStoreWebApp
```

- Giải thích:
  - `dotnet new`: Lệnh tạo dự án mới.
  - `mvc`: Template cho dự án ASP.NET Core MVC.
  - `-n SportsStoreWebApp`: Tên của dự án (project).

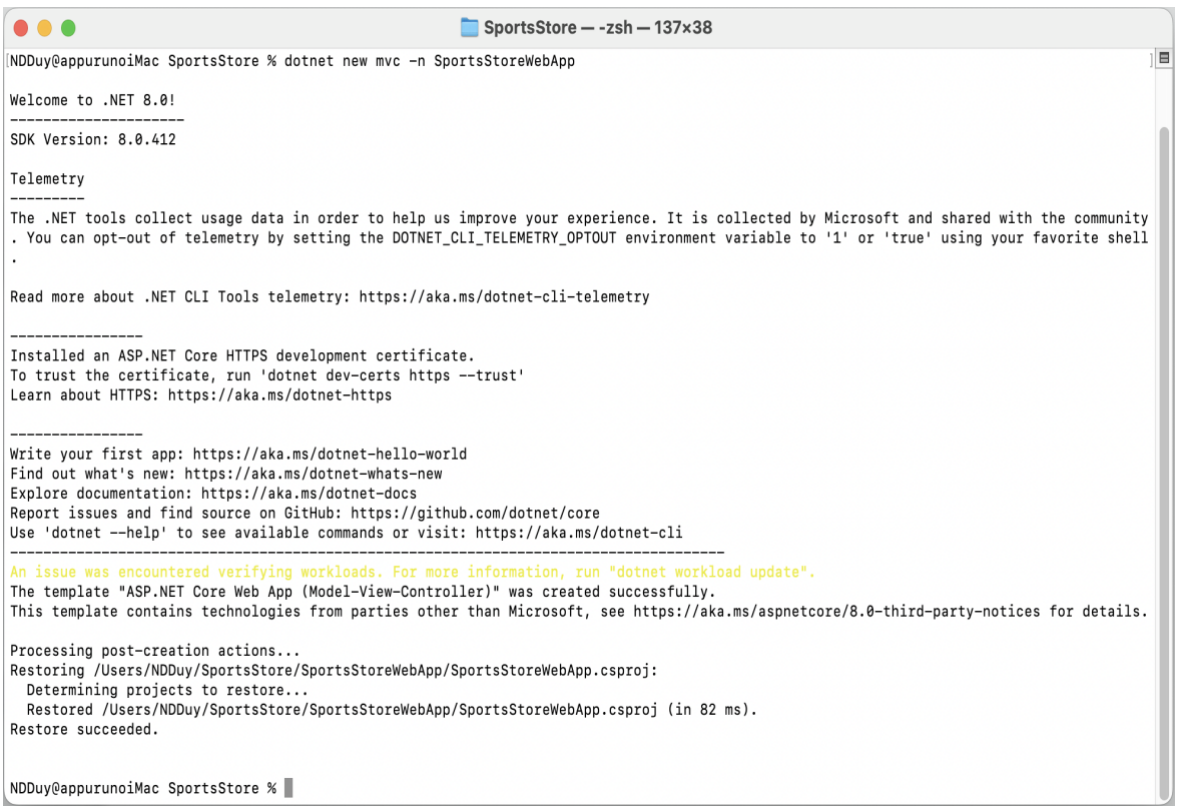


Figure 1-18: Tạo dự án ASP.NET Core MVC với .NET CLI

– Mở dự án trong Visual Studio/VS Code:

- Trong Visual Studio: Chọn `File` -> `Open` -> `Project/Solution...`, duyệt đến thư mục `SportsStoreWebApp` và mở file `SportsStoreWebApp.csproj`.
- Trong VS Code: `File` -> `Open Folder...`, chọn thư mục `SportsStoreWebApp`.
- Chạy ứng dụng:

```
cd SportsStoreWebApp
dotnet run
```

Figure 1-19: Chạy ứng dụng ASP.NET Core từ Command Line

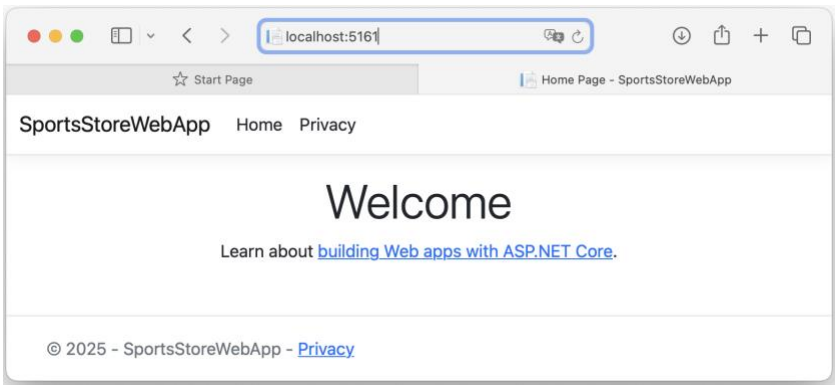


Figure 1-20: Kết quả trang Home Page

Cách 2: Sử dụng Visual Studio 2022 (Nhanh chóng và tích hợp):

– Mở Visual Studio 2022.

– Chọn `Create a new project`.

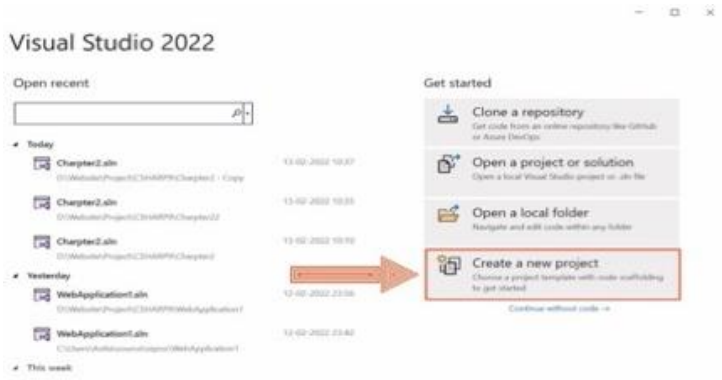


Figure 1-21: Màn hình khởi động Visual Studio - Tạo dự án mới

– Trong cửa sổ tìm kiếm template, gõ `ASP.NET Core Web App (Model-View-Controller)`.

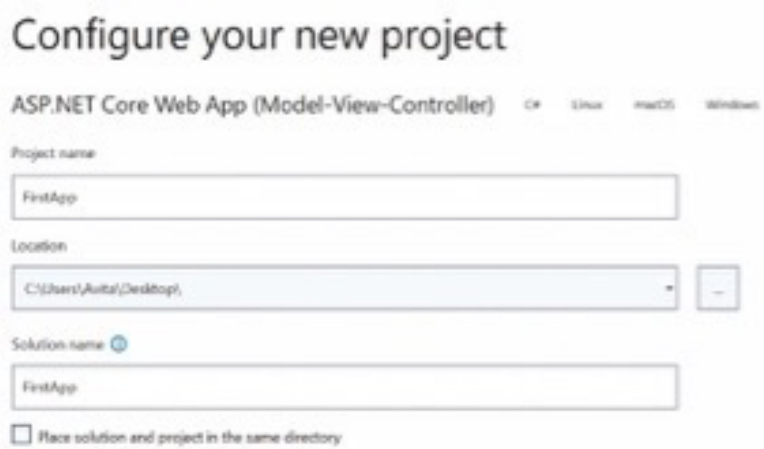
– Chọn template đó và nhấn `Next`.

Figure 1-22: Chọn template ASP.NET Core Web App (Model-View-Controller)

– Đặt tên Project: `SportsStoreWebApp`.

– Chọn Location (thư mục lưu trữ).

– Chọn **.NET 8 (Long Term Support)** và bỏ chọn "Configure for HTTPS" (để đơn giản cho lần đầu). Có thể chọn "Individual Accounts" nếu muốn tích hợp sẵn Identity, nhưng ở tuần này thì không cần.



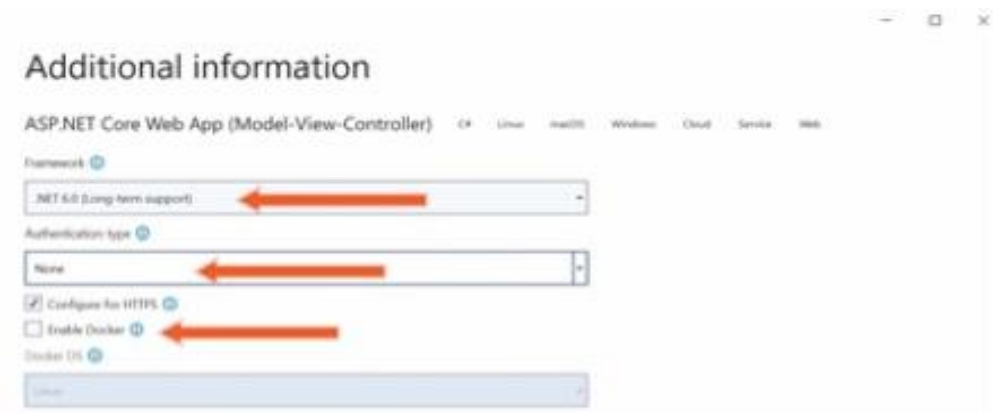


Figure 1-23: Cấu hình dự án mới trong Visual Studio

– Nhấn `Create`.

– Chạy ứng dụng: Nhấn nút `IIS Express` màu xanh lá cây hoặc `F5`.

1.2.1. Khám phá cấu trúc dự án mặc định (.NET 8):

– Mở rộng thư mục `SportsStoreWebApp` trong Solution Explorer (Visual Studio) hoặc File Explorer (VS Code).

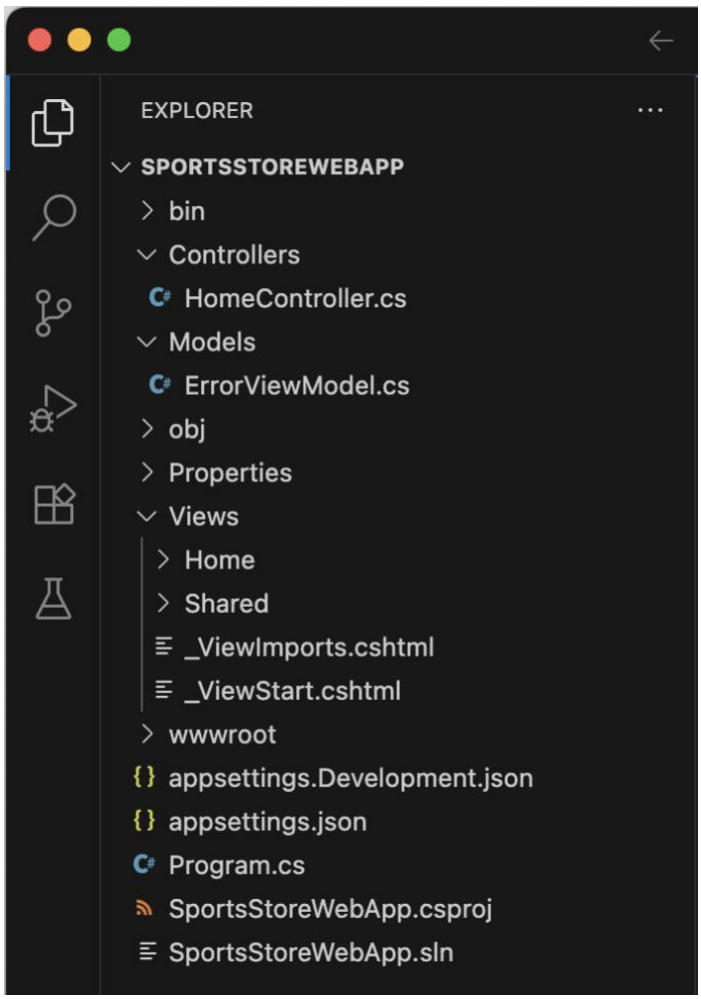


Figure 1-24: Cấu trúc thư mục của dự án ASP.NET Core MVC mặc định

– Tập trung vào **`Program.cs`**:

– Giải thích các dòng:

- `var builder = WebApplication.CreateBuilder(args);`
- `builder.Services.AddControllersWithViews();`
  - Đăng ký dịch vụ MVC.
- `var app = builder.Build();`
- `app.UseStaticFiles();`, `app.UseRouting();`, `app.MapControllerRoute();`

- Các Middleware cơ bản trong Request Pipeline.

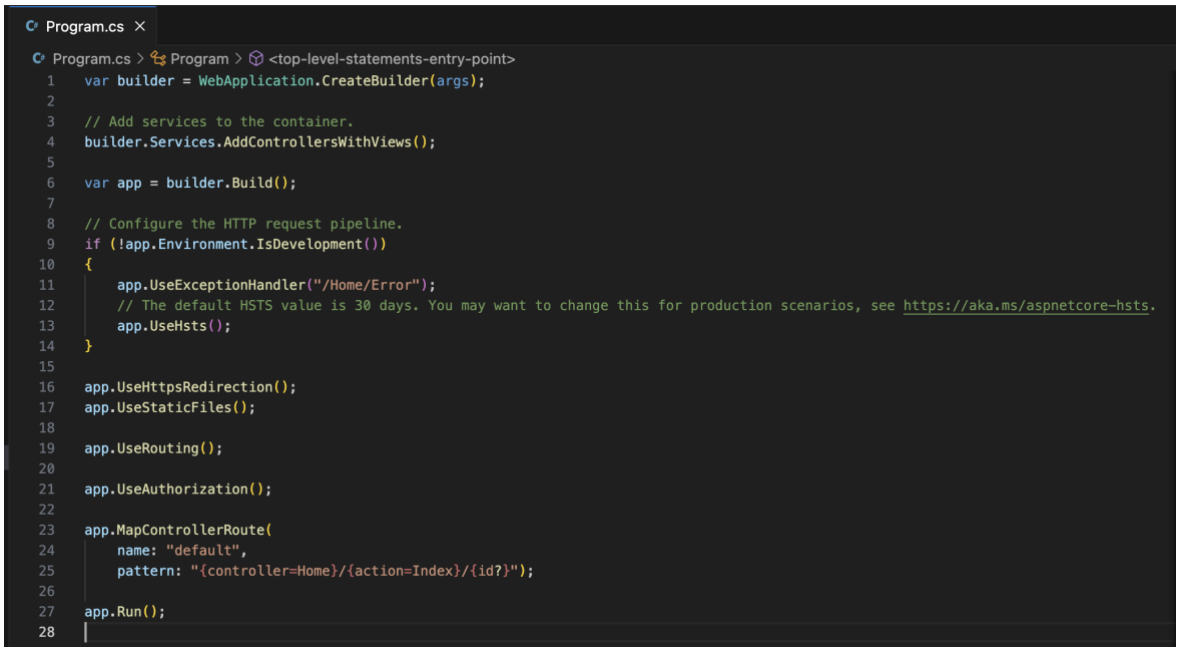


Figure 1-25: File Program.cs - Cấu hình cốt lõi của ứng dụng

1.3. Tạo mới một Controller và một View đơn giản

Bước 1: Tạo TestController.cs

- Trong thư mục `Controllers`, chuột phải -> `Add` -> `Class...` hoặc `Controller...`.
- Chọn `MVC Controller - Empty`.
- Đặt tên: `TestController.cs`

Code:

```
using Microsoft.AspNetCore.Mvc;

namespace SportsStoreWebApp.Controllers
{
    public class TestController : Controller
    {
        // Action Method mặc định khi truy cập /Test
        public IActionResult Index()
        {
            ViewBag.Message = "Chào mừng bạn đến với Cửa hàng Thể thao!
Đây là trang Test!";
            return View(); // Trả về View có tên Index
        }

        // Một Action Method khác: /Test/HelloWorld
        public IActionResult HelloWorld()
        {
```

```
            return Content("Xin chào từ Action HelloWorld của
TestController!");
        }

        // Action nhận tham số: /Test/Welcome?name=DavidTeo
        public IActionResult Welcome(string name = "Khách")
        {
            return Content($"Chào mừng {name} đến với trang Test!");
        }
    }
}
```

Bước 2: Tạo Index.cshtml cho TestController

- Trong thư mục `Views`, chuột phải -> `Add` -> `New Folder`. Đặt tên là `Test`.
- Trong thư mục Test vừa tạo, chuột phải -> `Add` -> `View...` -> `Razor View - Empty`.
- Đặt tên là `Index.cshtml`.

Code:

```
@{
    ViewData["Title"] = "Trang Test"; // Dữ liệu cho tiêu đề trang
}
<h1>@ViewBag.Message</h1>
<p>Bạn có thể truy cập các đường dẫn sau:</p>
<ul>
    <li><a href="/Test/HelloWorld">/Test/HelloWorld</a></li>
    <li><a href="/Test/Welcome?name=Teo">/Test/Welcome?name=Teo</a></li>
</ul>
```

Bước 3: Chạy và kiểm tra:

- Chạy lại ứng dụng (`dotnet run` hoặc `F5`).
- Truy cập `https://localhost:xxxx/Test` (hoặc `/test`).
- Truy cập `https://localhost:xxxx/Test/HelloWorld`.
- Truy cập `https://localhost:xxxx/Test/Welcome?name=NVTeo`.

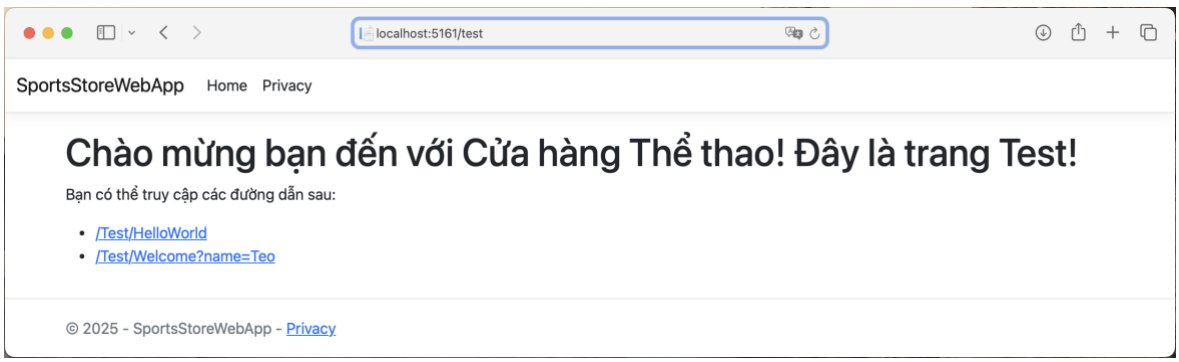


Figure 1-26: Kết quả hiển thị của trang TestController Index

1.4. Thực hành các cú pháp LINQ cơ bản và viết hàm async/await:

Bước 1: Thêm một lớp Model cơ bản cho Sản phẩm (trong thư mục Models):

– Tạo file `Product.cs` trong thư mục `Models`.

Code:

```
namespace SportsStoreWebApp.Models
{
    public class Product
    {
        public int ProductID { get; set; }
        public string Name { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
        public decimal Price { get; set; }
        public string Category { get; set; } = string.Empty;
    }
}
```

Bước 2: Viết code thực hành LINQ và Async/Await trong một file Console App tạm thời hoặc trong một phương thức trong Program.cs (chỉ để minh họa):

– Trong `Program.cs`, trước dòng `app.Run();`, thêm một đoạn code để minh họa:

```
// --- Bắt đầu phần thực hành C# cơ bản ---
Console.WriteLine("--- Thực hành C# cơ bản ---");

// Tạo danh sách sản phẩm mẫu
List<SportsStoreWebApp.Models.Product> sampleProducts = new
List<SportsStoreWebApp.Models.Product>
{
    new SportsStoreWebApp.Models.Product { ProductID = 1, Name = "Bóng đá World
Cup", Description = "Bóng đá chính hãng", Price = 50.00m, Category = "Bóng đá" },
    new SportsStoreWebApp.Models.Product { ProductID = 2, Name = "Áo đấu CLB A",
Description = "Áo đấu cho người hâm mộ", Price = 75.50m, Category = "Quần áo" },
    new SportsStoreWebApp.Models.Product { ProductID = 3, Name = "Vợt Tennis
Pro", Description = "Vợt chuyên nghiệp", Price = 150.00m, Category = "Tennis" },
    new SportsStoreWebApp.Models.Product { ProductID = 4, Name = "Giày chạy bộ
ABC", Description = "Giày thể thao nhẹ", Price = 99.99m, Category = "Giày" },
    new SportsStoreWebApp.Models.Product { ProductID = 5, Name = "Bóng rổ NBA",
Description = "Bóng rổ tiêu chuẩn", Price = 45.00m, Category = "Bóng rổ" }
};

Console.WriteLine("\n--- LINQ: Lọc sản phẩm có giá trên 70 ---");
var expensiveProducts = sampleProducts.Where(p => p.Price > 70.00m);
foreach (var p in expensiveProducts)
{
    Console.WriteLine($"- {p.Name} ({p.Price:C})");
}

Console.WriteLine("\n--- LINQ: Lấy sản phẩm đầu tiên thuộc danh mục 'Bóng đá' ---
");
```

```
var firstFootballProduct = sampleProducts.FirstOrDefault(p => p.Category == "Bóng
đá");
if (firstFootballProduct != null)
{
    Console.WriteLine($"- {firstFootballProduct.Name}");
}
else
{
    Console.WriteLine("Không tìm thấy sản phẩm bóng đá.");
}

Console.WriteLine("\n--- Async/Await: Mô phỏng thao tác bất đồng bộ ---");
async Task SimulateDataFetchAsync()
{
    Console.WriteLine("Đang bắt đầu lấy dữ liệu (mất 2 giây)...");
    await Task.Delay(2000); // Mô phỏng thao tác tốn thời gian
    Console.WriteLine("Đã lấy xong dữ liệu.");
}

// Gọi hàm bất đồng bộ
await SimulateDataFetchAsync(); // Cần `await` ở đây vì hàm Main của .NET 6+ đã
là async

Console.WriteLine("--- Kết thúc thực hành C# cơ bản ---\n");
// --- Kết thúc phần thực hành C# cơ bản ---
```

Figure 1-27: Kết quả console

Kết thúc buổi thực hành: Yêu cầu sinh viên trình bày kết quả chạy các URL đã tạo và giải thích ngắn gọn về code LINQ/async/await.

TÀI LIỆU THAM KHẢO

Giới thiệu tổng quan & c# cơ bản

Tài liệu chính thức của Microsoft:

[1]. **ASP.NET Core Documentation:** Đây là nguồn tài liệu số 1 và đầy đủ nhất về ASP.NET Core. Nó bao gồm mọi thứ từ kiến trúc cơ bản đến các tính năng nâng cao: https://learn.microsoft.com/en-us/aspnet/core/

[2]. **C# Documentation:** Nguồn tài liệu toàn diện về ngôn ngữ C#, bao gồm cả các khái niệm OOP, LINQ và **Async**/Await: https://learn.microsoft.com/en-us/dotnet/csharp/

Sách:

[3]. Mark J. Price (2023), *C# 12 and .NET 8 – Modern Cross-Platform Development*, Packt Publishing: Cuốn sách này cung cấp một nền tảng vững chắc về C# và .NET, rất phù hợp cho người mới bắt đầu.

Khóa học trực tuyến (Tìm kiếm trên các nền tảng):

[4]. Các khóa học về "C# Programming for Beginners" hoặc "ASP.NET Core MVC Fundamentals" **trên** Udemy, Pluralsight.

Cấu trúc cơ bản và kiến trúc ứng dụng

Tài liệu chính thức của Microsoft:

[5]. **ASP.NET Core Fundamentals:** Tập trung vào các khái niệm về Startup (nay là Program.cs), Middleware, Routing, Dependency Injection: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/

[6]. **Dependency Injection in ASP.NET Core:** Giải thích sâu về DI và các vòng đời dịch vụ: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection/

[7]. **Routing in ASP.NET Core:** Hướng dẫn cấu hình định tuyến: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing/

Sách:

[8]. Adam Freeman (2016), *Pro ASP.NET Core MVC 6th*, Apress: Đây là một trong những cuốn sách chuyên sâu và toàn diện nhất về ASP.NET Core MVC, bao gồm cả kiến trúc và cấu hình ứng dụng.

Models – Controllers – Views

Tài liệu chính thức của Microsoft:

[9]. **ASP.NET Core MVC Overview:** Tổng quan về kiến trúc MVC trong ASP.NET Core: https://learn.microsoft.com/en-us/aspnet/core/mvc/

[10]. **Model Binding in ASP.NET Core MVC:** Giải thích cách dữ liệu được ánh xạ vào

Model: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding/

[11]. **Validation in ASP.NET Core MVC:** Hướng dẫn sử dụng Data Annotations và Model Validation: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation/

[12]. **Razor Syntax Reference:** Hướng dẫn cú pháp Razor: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor

[13]. **Layout in ASP.NET Core:** Cách tạo và sử dụng Layout: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout

[14]. **Tag Helpers in ASP.NET Core:** Hướng dẫn sử dụng Tag Helpers: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro

[15]. **View Components in ASP.NET Core:** Giải thích về View Component: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/view-components

Sách:

[16]. Adam Freeman (2016), Pro ASP.NET Core MVC 6th, Apress (tiếp tục sử dụng).

Tương tác dữ liệu với Entity Framework Core

Tài liệu chính thức của Microsoft:

[17]. **Entity Framework Core Documentation:** Tài liệu chính thức cho EF Core, bao gồm Code-First, Querying, Updating, Deleting và Migrations: https://learn.microsoft.com/en-us/ef/core/

[18]. **Getting Started with EF Core:** Hướng dẫn từng bước cơ bản: https://learn.microsoft.com/en-us/ef/core/get-started/

Sách:

[19]. Brian L. Gorman (2020), *Practical Entity Framework*, Apress: Cuốn sách chuyên sâu về EF Core.

Xác thực & Phân quyền

Tài liệu chính thức của Microsoft Learn

- Tổng quan về Xác thực và Phân quyền trong ASP.NET Core:
  - Authentication (Xác thực):

[20]. **ASP.NET Core authentication overview**: Giải thích các khái niệm cơ bản về xác thực, các lược đồ xác thực (authentication schemes) và cách hoạt động của middleware xác thực: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/

  - Authorization (Phân quyền):

[21]. **Authorization in ASP.NET Core**: Giới thiệu tổng quan về phân quyền, sự khác biệt với xác thực, và các phương pháp phân quyền khác nhau (dựa trên vai trò, dựa trên chính sách): https://learn.microsoft.com/en-us/aspnet/core/security/authorization/introduction

Xây dựng và sử dụng Webservices (RESTful API)

Tài liệu chính thức của Microsoft:

[22]. **Create web APIs with ASP.NET Core:** Hướng dẫn xây dựng API Controller: https://learn.microsoft.com/en-us/aspnet/core/web-api/

[23]. **Authentication and Authorization in ASP.NET Core:** Các phương pháp bảo mật: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/

[24]. **JWT Bearer Authentication:** Hướng dẫn sử dụng JWT: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/jwt-bearer

[25]. **Get started with Swashbuckle** and **ASP.NET Core:** Tài liệu hóa API với Swagger/OpenAPI: https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger

[26]. **Enable Cross-Origin Requests (CORS) in ASP.NET Core:** Hướng dẫn cấu hình CORS: https://learn.microsoft.com/en-us/aspnet/core/security/cors

Công cụ:

[27]. **Postman:** Công cụ phổ biến để kiểm thử API: https://www.postman.com/

[28]. **Fetch API (MDN Web Docs):** Tài liệu về Fetch API trong JavaScript để tiêu thụ API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Lập trình Front-end với Angular

Tài liệu chính thức của Angular:

[29]. **Angular Documentation:** Nguồn tài liệu chính thức và đầy đủ nhất về Angular: https://angular.io/docs

[30]. **Angular CLI:** Hướng dẫn sử dụng công cụ dòng lệnh của Angular: https://angular.io/cli

[31]. **HttpClient in Angular:** Cách gọi API từ Angular: https://angular.io/guide/http

Sách:

[32]. Doguhan Uluca (2020), *Angular for Enterprise-Ready Web Applications*, Packt Publishing: Sách này tập trung vào việc xây dựng ứng dụng Angular chất lượng cao.

Khóa học trực tuyến:

[33]. Các khóa học về "Angular Crash Course" hoặc "Angular complete guide" trên Udemy, Pluralsight.