ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

GIÁO TRÌNH

CÔNG NGHỆ PHÁT TRIỂN ỨNG DỤNG

Chủ biên: **TS. Nguyễn Thế Vịnh**
Đồng chủ biên: **TS. Nguyễn Toàn Thắng**
Tham gia biên soạn: **ThS. Nguyễn Thu Phương**
ThS. Đoàn Ngọc Phương
ThS. Nguyễn Thị Dung

Thái Nguyên, 2026

ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

GIÁO TRÌNH

CÔNG NGHỆ PHÁT TRIỂN ỨNG DỤNG

Đơn vị phụ trách: Khoa Công nghệ thông tin
Chủ biên: **TS. Nguyễn Thế Vịnh**
Đồng chủ biên: **TS. Nguyễn Toàn Thắng**
Tham gia biên soạn: **ThS. Nguyễn Thu Phương**
ThS. Đoàn Ngọc Phương
ThS. Nguyễn Thị Dung

Thái Nguyên, 2026

Thông tin biên soạn

Họ và tên	Vai trò
TS. Nguyễn Thế Vịnh	Chủ biên
TS. Nguyễn Toàn Thắng	Đồng chủ biên
ThS. Nguyễn Thu Phương	Tham gia biên soạn
ThS. Đoàn Ngọc Phương	Tham gia biên soạn
ThS. Nguyễn Thị Dung	Tham gia biên soạn
Mã số học phần	ADT332
Đơn vị phụ trách	Khoa Công nghệ thông tin

MỤC LỤC

Thông tin biên soạn **i**
Danh mục hình ảnh **vi**
Danh mục bảng biểu **vii**
Danh mục từ viết tắt **ix**
1 Tổng quan công nghệ phát triển ứng dụng và nền tảng ASP.NET Core **1**
Nội dung chính của chương . . . 1
Mục tiêu cần đạt của chương . . . 1
1.1 Khái quát về công nghệ phát triển ứng dụng . . . 1
1.2 Công nghệ phát triển ứng dụng web . . . 2
1.3 Một số hệ sinh thái phát triển web . . . 7
1.4 Lựa chọn ASP.NET Core MVC làm nền tảng chính . . . 7
1.5 Tổng quan .NET và ASP.NET Core MVC . . . 8
1.6 Cài đặt môi trường và thực hành đầu tiên . . . 11
1.7 Sử dụng tác tử AI trong thực hành ASP.NET Core MVC . . . 17
1.8 Thiết lập kiểm thử tự động cho AppDemo . . . 18
1.9 Lộ trình phát triển AppDemo trong giáo trình . . . 19
Tổng kết chương . . . 23
Câu hỏi ôn tập . . . 23
Bài thực hành Chương 1 . . . 24
2 Cấu trúc ứng dụng và mô hình MVC **26**
Nội dung chính của chương . . . 26
Mục tiêu cần đạt của chương . . . 26
2.1 Từ HTML rời rạc đến module MVC có cấu trúc . . . 26
2.2 Cấu hình ứng dụng và luồng request trong Program.cs . . . 28
2.3 Cơ chế Routing trong ASP.NET Core MVC . . . 32
2.4 Controller, model và view model của AppDemo . . . 34
2.5 View, layout, partial view và tài nguyên tĩnh . . . 36
2.6 Quy trình phát triển một lát cắt Company . . . 41
2.7 Tác tử AI hỗ trợ rà soát AppDemo . . . 43
2.8 Kiểm thử các action của CompanyController . . . 43
Tổng kết chương . . . 49
Câu hỏi ôn tập . . . 49
Bài thực hành Chương 2 . . . 49
3 Tương tác giữa Controller, View và Model **51**
Nội dung chính của chương . . . 51
Mục tiêu cần đạt của chương . . . 51

3.1 Từ dữ liệu rời rạc đến model có cấu trúc . . . 51
3.2 Controller chuẩn bị dữ liệu cho từng request . . . 53
3.3 View model và hợp đồng dữ liệu của view . . . 57
3.4 Strongly typed view và Razor . . . 61
3.5 Dùng tác tử AI rà soát luồng dữ liệu . . . 66
3.6 Kiểm thử model do CompanyController trả về . . . 67
Tổng kết chương . . . 74
Câu hỏi ôn tập . . . 74
Bài thực hành Chương 3 . . . 74

4 Kỹ thuật gỡ lỗi trong ứng dụng web ASP.NET Core 76
Nội dung chính của chương . . . 76
Mục tiêu cần đạt của chương . . . 76
4.1 Từ lỗi hiển thị đến quy trình gỡ lỗi có kiểm soát . . . 77
4.2 Breakpoint và kiểm tra dữ liệu runtime . . . 81
4.3 ModelState, log và exception handling . . . 86
4.4 Khoanh vùng lỗi trong AppDemo . . . 92
4.5 Chẩn đoán các vấn đề trong quá trình vận hành ứng dụng . . . 97
4.6 Dùng tác tử AI để gỡ lỗi cùng bài toán AppDemo . . . 98
4.7 Kiểm thử tự động và ngăn lỗi tái phát . . . 99
Tổng kết chương . . . 100
Câu hỏi ôn tập . . . 101
Bài thực hành Chương 4 . . . 101

5 Xử lý biểu mẫu và Model Binding 103
Nội dung chính của chương . . . 103
Mục tiêu cần đạt của chương . . . 103
5.1 Từ request rời rạc đến model binding . . . 103
5.2 Data Annotations và ModelState như hợp đồng dữ liệu . . . 108
5.3 Luồng GET, POST và dữ liệu phụ trợ của form . . . 115
5.4 Dropdown, JSON và ViewModel . . . 120
5.5 Dùng tác tử AI để rà soát form và model binding . . . 127
5.6 Kiểm thử tự động biểu mẫu và Model Binding . . . 128
Tổng kết chương . . . 130
Câu hỏi ôn tập . . . 130
Bài thực hành Chương 5 . . . 131

6 Thiết lập và ánh xạ dữ liệu với Entity Framework Core 132
Nội dung chính của chương . . . 132
Mục tiêu cần đạt của chương . . . 132
6.1 Từ dữ liệu tạm đến cơ sở dữ liệu . . . 132
6.2 Entity, quan hệ và ApplicationDbContext . . . 138
6.3 Đăng ký DbContext, migration và seed data . . . 143
6.4 Truy vấn và ghi dữ liệu bằng DbContext trong controller . . . 149

6.5 Dùng tác tử AI để kiểm tra cấu hình EF Core . . . 157
Tổng kết chương . . . 159
Câu hỏi ôn tập . . . 159
Bài thực hành Chương 6 . . . 159
7 CRUD và truy vấn với Entity Framework Core 161
Nội dung chính của chương . . . 161
Mục tiêu cần đạt của chương . . . 161
7.1 Từ danh sách dữ liệu đến quy trình CRUD . . . 161
7.2 LINQ cho tìm kiếm, sắp xếp và phân trang . . . 164
7.3 Create, Edit và Delete với kiểm tra dữ liệu . . . 170
7.4 CRUD cho dữ liệu có quan hệ và projection . . . 176
7.5 Dùng tác tử AI để rà soát CRUD và LINQ . . . 184
7.6 Kiểm thử tự động CRUD, LINQ và phân trang . . . 186
Tổng kết chương . . . 188
Câu hỏi ôn tập . . . 189
Bài thực hành Chương 7 . . . 189
8 Xác thực người dùng và phân quyền 191
Nội dung chính của chương . . . 191
Mục tiêu cần đạt của chương . . . 191
8.1 Từ ẩn nút giao diện đến bảo vệ action . . . 191
8.2 Cấu hình Identity trong AppDemo . . . 194
8.3 Đăng nhập, đăng ký và đăng xuất . . . 198
8.4 Role Admin và bảo vệ chức năng ghi dữ liệu . . . 204
8.5 Dùng tác tử AI để rà soát xác thực và phân quyền . . . 214
8.6 Kiểm thử tự động xác thực và phân quyền . . . 215
Tổng kết chương . . . 218
Câu hỏi ôn tập . . . 218
Bài thực hành Chương 8 . . . 219
9 Tích hợp trí tuệ nhân tạo vào ứng dụng ASP.NET Core MVC 221
Nội dung chính của chương . . . 221
Mục tiêu cần đạt của chương . . . 221
9.1 Vai trò của AI trong ứng dụng ASP.NET Core MVC . . . 221
9.2 Phân tích bài toán và thiết kế luồng nghiệp vụ . . . 222
9.3 Mở rộng mô hình dữ liệu và ViewModel . . . 223
9.4 Thiết kế kiến trúc dịch vụ AI . . . 225
9.5 Cấu hình và gọi dịch vụ AI . . . 226
9.6 Tích hợp AI vào bộ điều khiển và khung nhìn Razor . . . 233
9.7 An toàn, bảo mật và độ tin cậy . . . 239
9.8 Dùng tác tử AI để rà soát chức năng . . . 241
9.9 Kiểm thử chức năng AI . . . 241
Tổng kết chương . . . 248

Câu hỏi ôn tập và bài tập 248
Bài thực hành Chương 9 248
Kết luận và định hướng mở rộng **251**
Từ điển thuật ngữ Anh–Việt **254**
Câu hỏi thường gặp **256**
Tài liệu tham khảo **258**

DANH MỤC HÌNH ẢNH

Hình 1.1. Mô hình client–server và luồng request/response 4
Hình 1.2. HTTP, URL và hai phương thức GET/POST thường dùng . . 5
Hình 1.3. Trang chủ AppDemo sau bài thực hành Chương 1 14

Hình 2.1. Trang Company minh họa controller, view model, view và layout cùng hoạt động 34
Hình 2.2. Trang Services dùng layout, partial view và CSS chung của AppDemo 40

Hình 3.1. Trang liên hệ nhận model `CompanyContact` thay vì các chuỗi rời rạc 57
Hình 3.2. Trang dịch vụ hiển thị danh sách `CompanyService` có cấu trúc . 61

Hình 4.1. Thông báo validation trên form thêm diễn giả là bằng chứng đầu vào cho quá trình debug 87
Hình 4.2. Form bài trình bày cần kiểm tra cả `ModelState` lẫn dữ liệu dropdown khi validation lỗi 92

Hình 5.1. Form thêm đơn vị tổ chức cho thấy Tag Helper nối field HTML với model `Organizer` 115
Hình 5.2. Form thêm lễ hội dùng dropdown để gửi `OrganizerId` thay vì bắt người dùng nhập khóa ngoại 121

Hình 6.1. Danh sách lễ hội đọc dữ liệu từ SQL Server và hiển thị tên đơn vị tổ chức qua quan hệ 144
Hình 6.2. Dữ liệu bài trình bày được seed và truy vấn cùng thông tin diễn giả liên quan 150

Hình 7.1. Trang diễn giả kết hợp tìm kiếm, sắp xếp và phân trang 171
Hình 7.2. Trang bài trình bày dùng projection để hiển thị dữ liệu từ `Presentation` và `Speaker` 177

Hình 8.1. Luồng xác thực và phân quyền của một request ASP.NET Core 198
Hình 8.2. Trang đăng nhập của AppDemo 199
Hình 8.3. Giao diện thao tác dành cho Admin 204

Hình 9.1. Kết quả gợi ý dùng fallback khi dịch vụ AI chưa khả dụng . . 231
Hình 9.2. Luồng sinh, kiểm duyệt và lưu nội dung do AI hỗ trợ 233
Hình 9.3. Module AI hiển thị rõ trạng thái demo khi chưa có API key . . 234

DANH MỤC BẢNG BIỂU

Bảng 1.1. Một số loại ứng dụng trong lĩnh vực phát triển phần mềm . . . 2
Bảng 1.2. Các khái niệm nền tảng khi chuyển từ console sang ứng dụng web . . . 3
Bảng 1.3. Một số hệ sinh thái tiêu biểu trong phát triển ứng dụng web . . 7
Bảng 1.4. Một số điểm chuyển đổi thường gặp từ Java sang C# 9
Bảng 1.5. Dấu hiệu cần kiểm chứng khi tạo dự án MVC đầu tiên 17
Bảng 1.6. Các trường hợp kiểm tra ban đầu của AppDemo 18
Bảng 1.7. Lộ trình phát triển AppDemo qua chín chương 19
Bảng 1.8. Bộ bằng chứng nền cho dự án web đầu tiên 20
Bảng 1.9. Ma trận hỏi trước khi chọn hoặc đổi công nghệ 21
Bảng 1.10. Ma trận phạm vi môi trường thực hành 21
Bảng 1.11. Khoanh vùng sự cố ở lần chạy đầu 22
Bảng 1.12. Phiếu xác nhận môi trường trước khi chuyển chương 23

Bảng 2.1. Trách nhiệm của các thành phần MVC trong AppDemo 27
Bảng 2.2. Bằng chứng khi lần theo request Company/Contact 29
Bảng 2.3. Ba vòng đời service thường dùng 30
Bảng 2.4. Dấu hiệu chẩn đoán theo tầng xử lý request 31
Bảng 2.5. Quyết định có cần thay đổi Program.cs hay không 31
Bảng 2.6. Phân tích URL theo route mặc định 32
Bảng 2.7. Ý nghĩa các ký hiệu trong route mặc định 32
Bảng 2.8. Một số kết quả action và mục đích sử dụng 35
Bảng 2.9. Phân loại trách nhiệm trong Razor view 38
Bảng 2.10. Chọn cơ chế tái sử dụng giao diện 39
Bảng 2.11. Hợp đồng quan sát được của module Company 41
Bảng 2.12. Bằng chứng và giới hạn của từng phép kiểm tra 43
Bảng 2.13. Các trường hợp kiểm thử `CompanyController` 43
Bảng 2.14. Bất biến kiến trúc của một lát cắt MVC 44
Bảng 2.15. Câu hỏi chọn vòng đời dependency 45
Bảng 2.16. Ma trận kiểm tra một thay đổi cấu trúc 46
Bảng 2.17. Ranh giới của module MVC khi dự án tăng 46
Bảng 2.18. Kiểm tra cấu hình theo thời điểm 47
Bảng 2.19. Bộ kiểm tra tác động cho một module MVC 48
Bảng 2.20. Bằng chứng cho một lát cắt MVC hoàn chỉnh 48

Bảng 3.1. Lựa chọn cách truyền dữ liệu từ controller sang view 53
Bảng 3.2. Hợp đồng dữ liệu của module Company 54
Bảng 3.3. Câu hỏi chẩn đoán từ action đến view 55
Bảng 3.4. Đối chiếu action, model và strongly typed view 58
Bảng 3.5. Dấu hiệu lựa chọn model hoặc view model 60
Bảng 3.6. Triệu chứng và nơi bắt đầu chẩn đoán 64
Bảng 3.7. Các trường hợp kiểm thử model do action `Index` trả về 67
Bảng 3.8. Phạm vi bằng chứng của các mức kiểm tra 68

Bảng 3.9. Bảng sở hữu dữ liệu của một view model . . . 69
Bảng 3.10. Dấu hiệu hợp đồng view model đang quá rộng hoặc quá hẹp . 70
Bảng 3.11. Ma trận kiểm chứng khi tiến hóa hợp đồng dữ liệu . . . 70
Bảng 3.12. Quyết định tách một phần giao diện . . . 71
Bảng 3.13. Trạng thái cần kiểm tra của strongly typed view . . . 72
Bảng 3.14. Phiếu rà soát một hợp đồng màn hình . . . 72
Bảng 3.15. Kiểm tra đầu ra của strongly typed view . . . 73

Bảng 4.1. Hợp đồng của một bài thực hành cài lỗi . . . 77
Bảng 4.2. Từ triệu chứng giao diện đến tầng cần kiểm tra . . . 80
Bảng 4.3. Công cụ gỡ lỗi và loại bằng chứng thu được . . . 84
Bảng 4.4. Số liệu chẩn đoán ban đầu cho các hiện tượng suy giảm hiệu năng . . . 97
Bảng 4.5. Các trường hợp kiểm thử hồi quy Chương 4 . . . 99
Bảng 4.6. Bằng chứng cho lỗi gián đoạn . . . 100

Bảng 5.1. Luồng GET/POST của một form MVC . . . 106
Bảng 5.2. Phân biệt các lớp lỗi khi xử lý biểu mẫu . . . 109
Bảng 5.3. Ràng buộc dữ liệu thường gặp trong form của `AppDemo` . . . 110
Bảng 5.4. Quyết định chuẩn hóa dữ liệu biểu mẫu . . . 129

Bảng 6.1. Ánh xạ giữa object C# và cơ sở dữ liệu trong EF Core . . . 133
Bảng 6.2. Lựa chọn môi trường cơ sở dữ liệu cho AppDemo . . . 134
Bảng 6.3. Từ migration đến dữ liệu chạy thật trong `AppDemo` . . . 149
Bảng 6.4. Cổng kiểm tra một migration trước khi áp dụng . . . 158

Bảng 7.1. Ma trận action trong một module CRUD . . . 162
Bảng 7.2. Trình tự xây dựng truy vấn LINQ cho trang danh sách . . . 165
Bảng 7.3. Ma trận rà soát CRUD sau khi dùng tác tử AI . . . 186
Bảng 7.4. Chọn bằng chứng cho truy vấn danh sách . . . 187
Bảng 7.5. Lựa chọn xử lý cập nhật đồng thời . . . 187
Bảng 7.6. Câu hỏi chọn chính sách xóa . . . 188

Bảng 8.1. Phân biệt xác thực, phân quyền và role trong `AppDemo` . . . 192
Bảng 8.2. Các điểm bảo vệ bắt buộc với thao tác ghi dữ liệu . . . 194
Bảng 8.3. Các trạng thái quan trọng trong vòng đời tài khoản . . . 216
Bảng 8.4. Ma trận kiểm thử vòng đời và phiên đăng nhập . . . 217
Bảng 8.5. Chọn cơ chế phân quyền theo loại quyết định . . . 218

Bảng 9.1. Một số trường hợp sử dụng AI trong ứng dụng web . . . 222
Bảng 9.2. Tách thao tác gợi ý khỏi thao tác lưu . . . 222
Bảng 9.3. Phân loại lỗi khi gọi dịch vụ AI . . . 231
Bảng 9.4. Bộ tiêu chí đánh giá mô tả do AI hỗ trợ . . . 240
Bảng 9.5. Các cấp kiểm thử của chức năng AI . . . 241
Bảng 9.6. Ma trận kiểm thử tích hợp cho Endpoint gợi ý . . . 246
Bảng 9.7. Hồ sơ thay đổi một chức năng AI . . . 246
Bảng 9.8. Cổng phát hành cho thay đổi AI . . . 247
Bảng 9.9. Phiếu kiểm tra dữ liệu cho một lời gọi AI . . . 247

DANH MỤC TỪ VIẾT TẮT

AI	Artificial Intelligence – Trí tuệ nhân tạo
API	Application Programming Interface – Giao diện lập trình ứng dụng
ASP.NET	Active Server Pages .NET – Nền tảng phát triển web thuộc hệ sinh thái .NET
CI	Continuous Integration – Tích hợp liên tục
CLI	Command-Line Interface – Giao diện dòng lệnh
CRUD	Create, Read, Update, Delete – Thêm, đọc, sửa, xóa dữ liệu
CSRF	Cross-Site Request Forgery – Giả mạo yêu cầu liên trang
CSS	Cascading Style Sheets – Ngôn ngữ định dạng giao diện web
DBMS	Database Management System – Hệ quản trị cơ sở dữ liệu
DI	Dependency Injection – Tiêm phụ thuộc
DTO	Data Transfer Object – Đối tượng truyền dữ liệu
EF Core	Entity Framework Core
GET	Phương thức HTTP dùng để truy xuất tài nguyên
HTML	HyperText Markup Language – Ngôn ngữ đánh dấu siêu văn bản
HTTP	HyperText Transfer Protocol – Giao thức truyền tải siêu văn bản
HTTPS	HyperText Transfer Protocol Secure – HTTP qua kênh truyền được mã hóa
ID	Identifier – Định danh
IDE	Integrated Development Environment – Môi trường phát triển tích hợp
JSON	JavaScript Object Notation
LINQ	Language Integrated Query
MVC	Model–View–Controller
ORM	Object–Relational Mapping – Ánh xạ đối tượng–quan hệ
OWASP	Open Worldwide Application Security Project – Dự án mở về an toàn ứng dụng
PDF	Portable Document Format – Định dạng tài liệu di động
PHP	PHP: Hypertext Preprocessor – Ngôn ngữ lập trình phía máy chủ
POST	Phương thức HTTP dùng để gửi dữ liệu hoặc tạo tài nguyên
PUT	Phương thức HTTP dùng để thay thế hoặc cập nhật tài nguyên
REST	Representational State Transfer
SDK	Software Development Kit – Bộ công cụ phát triển phần mềm
SQL	Structured Query Language – Ngôn ngữ truy vấn có cấu trúc
TLS	Transport Layer Security – Bảo mật tầng truyền tải
URL	Uniform Resource Locator – Địa chỉ tài nguyên thống nhất
VS Code	Visual Studio Code – Trình soạn thảo mã nguồn của Microsoft
XSS	Cross-Site Scripting – Chèn mã kịch bản liên trang

LỜI NÓI ĐẦU

Giáo trình *CÔNG NGHỆ PHÁT TRIỂN ỨNG DỤNG* được biên soạn nhằm trang bị nền tảng và kỹ năng thực hành trong phát triển ứng dụng. Lĩnh vực này bao gồm nhiều loại sản phẩm và hệ sinh thái công nghệ; trong phạm vi học phần, giáo trình tập trung vào ứng dụng web phía máy chủ và chọn ASP.NET Core MVC làm nền tảng thực hành xuyên suốt. Lựa chọn đó giúp người học theo đuổi một mạch kiến thức đủ sâu, đồng thời nhận thức rằng công nghệ phải được lựa chọn theo bài toán, môi trường vận hành và năng lực của đội ngũ.

Giáo trình hướng tới sinh viên ngành Công nghệ thông tin, Kỹ thuật phần mềm và người học các học phần liên quan đến phát triển ứng dụng web. Người học nên có kiến thức nhập môn về lập trình hướng đối tượng, HTML/CSS, cơ sở dữ liệu quan hệ, dòng lệnh và Git.

Trên dự án AppDemo — hệ thống giới thiệu và quản lý sự kiện, hội thảo của một đơn vị công nghệ — nội dung được triển khai từ HTTP, mô hình MVC, giao diện và biểu mẫu đến gỡ lỗi, Entity Framework Core, CRUD, ASP.NET Core Identity và tích hợp dịch vụ trí tuệ nhân tạo có kiểm soát. Cách tổ chức đi từ bài toán thực tế đến khái niệm và phương pháp; mỗi giai đoạn đều gắn với hoạt động xây dựng, kiểm thử và bằng chứng kỹ thuật.

Mặc dù đã nỗ lực rà soát, giáo trình khó tránh khỏi hạn chế. Nhóm tác giả trân trọng cảm ơn các đồng nghiệp và người học đã đóng góp ý kiến, đồng thời mong tiếp tục nhận được trao đổi để giáo trình ngày càng hoàn thiện.

Thái Nguyên, 2026
Nhóm tác giả

CHƯƠNG 1

Tổng quan công nghệ phát triển ứng dụng và nền tảng ASP.NET Core

Nội dung chính của chương

Chương mở đầu đặt phát triển ứng dụng web trong bức tranh rộng hơn của công nghệ phát triển ứng dụng, bên cạnh ứng dụng di động, ứng dụng máy tính và dịch vụ phần mềm. Từ phạm vi đó, sinh viên tìm hiểu các thành phần của một ứng dụng web, nhận biết một số hệ sinh thái tiêu biểu và giải thích vì sao học phần chọn ASP.NET Core MVC làm nền tảng thực hành chính. Các phần tiếp theo nối request, response, HTTP, URL, HTML, CSS và JavaScript với C#, .NET và mô hình MVC; sau đó hướng dẫn tạo dự án, dùng tác tử AI và thiết lập kiểm thử xUnit để kiểm chứng kết quả. Phần nền tảng ASP.NET Core và MVC dựa trên tài liệu Microsoft Learn phiên bản 10.0 [14][21].

Mục tiêu cần đạt của chương

- Phân biệt được ứng dụng web, ứng dụng di động, ứng dụng máy tính và dịch vụ phần mềm; xác định đúng trọng tâm ứng dụng web của học phần.
- Giải thích được vai trò của trình duyệt, máy chủ, cơ sở dữ liệu, HTTP và các công nghệ giao diện trong luồng request–response.
- Nhận biết được một số hệ sinh thái web tiêu biểu và phân biệt ngôn ngữ, nền tảng, môi trường thực thi, framework với công cụ phát triển.
- Giải thích được lý do giáo trình chọn ASP.NET Core MVC theo mục tiêu học tập, thời lượng và dự án AppDemo, không xem đây là lựa chọn duy nhất cho mọi ứng dụng.
- Phân biệt được vai trò của C#, .NET và ASP.NET Core MVC trong việc xây dựng ứng dụng web có cấu trúc.
- Cài đặt được môi trường phát triển; tạo solution, dự án MVC và dự án xUnit; chạy và chỉnh sửa ứng dụng.
- Sử dụng được tác tử AI trong phạm vi rõ ràng và kiểm chứng AppDemo bằng build, test tự động và quan sát trên trình duyệt.

1.1 Khái quát về công nghệ phát triển ứng dụng

Công nghệ phát triển ứng dụng là tập hợp ngôn ngữ, nền tảng thực thi, framework, thư viện, công cụ và quy trình được dùng để tạo ra một sản phẩm phần mềm. Một ứng dụng không được xác định chỉ bởi ngôn ngữ lập trình: cùng một ngôn ngữ có thể tham gia nhiều loại sản phẩm, còn một sản phẩm hoàn chỉnh thường kết hợp mã nguồn, giao diện, dữ liệu, dịch vụ và môi trường triển khai.

Ở mức tổng quan, có thể phân biệt bốn loại ứng dụng tiêu biểu như trình bày trong Bảng 1.1. Việc phân loại dựa trên hai tiêu chí chính: nơi người dùng tương tác với ứng

dụng và cách phần mềm được cung cấp đến người dùng. Ranh giới giữa các loại ứng dụng không tuyệt đối. Chẳng hạn, một ứng dụng di động có thể gọi dịch vụ web, còn một ứng dụng web có thể cung cấp giao diện cho nhiều dịch vụ phần mềm phía sau.

Bảng 1.1. *Một số loại ứng dụng trong lĩnh vực phát triển phần mềm*

Loại ứng dụng	Cách người dùng tiếp cận	Ví dụ và vấn đề đặc trưng
Ứng dụng web	Dùng trình duyệt hoặc client giao tiếp với máy chủ qua mạng	Cổng thông tin, cửa hàng trực tuyến; cần xử lý request, dữ liệu, phiên và bảo mật web
Ứng dụng di động	Cài trên điện thoại hoặc máy tính bảng	Ứng dụng đặt lịch, bản đồ; cần quan tâm thiết bị, kết nối mạng và vòng đời ứng dụng
Ứng dụng máy tính	Cài và chạy trực tiếp trên hệ điều hành máy cá nhân	Phần mềm soạn thảo, quản lý nội bộ; thường tích hợp tệp, thiết bị và giao diện hệ điều hành
Dịch vụ phần mềm	Cung cấp chức năng qua API, tác vụ nền hoặc trao đổi thông điệp	Dịch vụ thanh toán, gửi thư, xử lý dữ liệu; chú trọng hợp đồng giao tiếp, độ tin cậy và khả năng tích hợp

Dù hình thức phân phối khác nhau, quá trình phát triển các loại ứng dụng đều phải trả lời những câu hỏi chung: yêu cầu nào cần đáp ứng, trách nhiệm được tổ chức ra sao, dữ liệu được kiểm tra và lưu trữ thế nào, hệ thống được bảo vệ và kiểm thử bằng bằng chứng nào. Các kiến thức chung đó tạo nền tảng để người học chuyển sang công nghệ khác sau học phần.

Giáo trình không triển khai sâu cả bốn loại ứng dụng. Xuất phát từ nhu cầu thực tiễn trong phát triển ứng dụng web hiện nay, ASP.NET Core MVC được lựa chọn làm công nghệ trọng tâm của giáo trình và được dùng để biến các khái niệm thành một dự án chạy được. Vì vậy, các loại ứng dụng còn lại đóng vai trò xác định bối cảnh của tên học phần, không phải những nhánh thực hành song song.

1.2 Công nghệ phát triển ứng dụng web

Từ lập trình hướng đối tượng đến ứng dụng web

Trước khi học học phần này, các bạn đã được học lập trình hướng đối tượng. Kiến thức đó vẫn là nền tảng, vì ứng dụng web cũng phải mô tả các đối tượng của bài toán: sản phẩm, khách hàng, đơn hàng, sinh viên, lớp học hoặc tài khoản người dùng. Lớp vẫn dùng để mô tả cấu trúc dữ liệu, thuộc tính vẫn biểu diễn trạng thái, phương thức vẫn biểu diễn hành vi, còn quan hệ giữa các lớp vẫn giúp ta hiểu nghiệp vụ.

Vấn đề là môi trường chạy đã thay đổi. Trong chương trình console, người dùng thường mở chương trình, nhập dữ liệu, chương trình xử lý theo một luồng tương đối liên tục, rồi in kết quả ra cùng cửa sổ. Cách nghĩ này phù hợp với bài toán nhỏ vì chỉ có một người đang tương tác trực tiếp với chương trình. Nếu cần sửa lỗi, ta thường nhìn vào luồng lệnh từ trên xuống dưới.

Ứng dụng web vận hành theo cách khác. Thay vì chạy trực tiếp một file chương trình, người dùng mở trình duyệt, truy cập địa chỉ, bấm liên kết, gửi biểu mẫu hoặc tải dữ liệu. Nhiều thao tác như vậy có thể làm trình duyệt tạo một hay nhiều request gửi đến server; thao tác chỉ thay đổi trạng thái cục bộ bằng JavaScript thì không nhất thiết gửi request. Trong cùng thời điểm, server có thể xử lý request của nhiều người dùng ở những màn hình và trạng thái khác nhau. Tư duy hướng đối tượng vẫn cần thiết, nhưng

nay phải giải quyết thêm ba yêu cầu: phản hồi đúng từng request, tách biệt dữ liệu giữa người dùng và giữ mã nguồn dễ bảo trì khi ứng dụng phát triển.

Bảng 1.2 đặt cạnh cách nghĩ console và cách nghĩ web để sinh viên thấy bài toán đã đổi trước khi học framework. Console phù hợp với một người dùng làm việc trực tiếp trên một máy; ứng dụng web phải nhận nhiều request, trả response cho trình duyệt và chạy ổn định trên nhiều môi trường. Vì vậy chương đi từ nền tảng client, server, HTTP, URL, HTML/CSS đến C#, .NET, ASP.NET Core MVC, rồi kết thúc bằng bài thực hành tạo dự án và dùng tác tử AI có kiểm soát [5].

Bảng 1.2. *Các khái niệm nền tảng khi chuyển từ console sang ứng dụng web*

Khái niệm	Trong cách nghĩ cũ	Trong ứng dụng web
Luồng chạy	Chương trình chạy liên tục trên một máy	Nhiều thao tác web tạo một hoặc nhiều request riêng gửi tới server
Đầu vào	Người dùng nhập trực tiếp vào console	Trình duyệt gửi URL, query string, form hoặc body của request
Đầu ra	Kết quả in ngay ra cửa sổ dòng lệnh	Server trả response, trình duyệt dựng thành HTML/CSS người dùng nhìn thấy
Kiểm tra lỗi	Theo dõi thứ tự dòng lệnh	Lần theo URL, route, controller, action, view và response

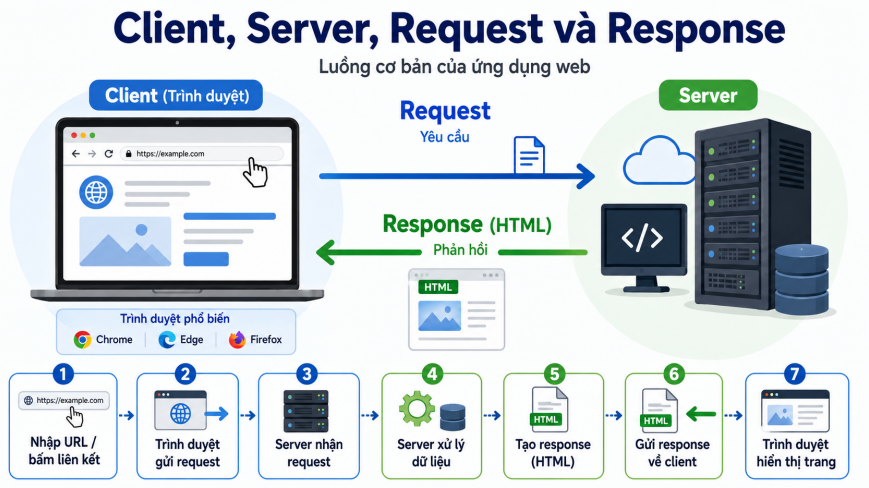
Client, server, request và response

Từ thay đổi trên, mô hình client–server xuất hiện như một cách chia trách nhiệm. Nếu mọi thứ đều nằm trong một chương trình duy nhất trên máy người dùng, việc cập nhật, chia sẻ dữ liệu và phục vụ nhiều người sẽ rất khó. Ứng dụng web tách hệ thống thành hai phía: client là nơi người dùng thao tác, thường là trình duyệt web như Chrome, Edge hoặc Firefox; server là nơi chạy logic ứng dụng, kiểm tra dữ liệu, truy cập cơ sở dữ liệu và quyết định kết quả trả về.

Giữa hai phía đó cần có một nhịp trao đổi rõ ràng. Khi người dùng nhập địa chỉ hoặc bấm liên kết, trình duyệt tạo ra một request gửi đến server. Request có thể hiểu là một yêu cầu có cấu trúc: người dùng muốn tài nguyên nào, gửi kèm dữ liệu gì, dùng phương thức nào và chấp nhận kiểu kết quả nào. Server nhận request, xử lý theo logic của ứng dụng rồi trả response. Response đầu tiên thường chứa tài liệu HTML. Khi phân tích HTML, trình duyệt có thể gửi thêm request để tải CSS, JavaScript, hình ảnh và các tài nguyên liên quan trước khi hoàn thiện giao diện.

Ví dụ, khi người dùng truy cập địa chỉ `https://ictu.edu.vn`, trình duyệt gửi request đến server của website `ictu.edu.vn`. Server xử lý yêu cầu, tạo response rồi gửi lại; trình duyệt tiếp tục đọc nội dung và hiển thị trang cho người dùng.

Luồng trao đổi tổng quát giữa trình duyệt và server được tóm tắt trong Hình 1.1.



Hình 1.1. *Mô hình client–server và luồng request/response*

Điểm quan trọng cần rút ra là ứng dụng web xử lý theo từng request. Mỗi lần người dùng mở trang, gửi form hoặc bấm nút, một request mới được tạo ra. Do đó, khi viết ứng dụng web, ta không hỏi "chương trình chạy đến dòng nào rồi?" như trong console, mà hỏi "request này đi vào đâu, được xử lý bởi thành phần nào, và response nào sẽ được trả về?". Câu hỏi này sẽ dẫn trực tiếp đến Controller, Action, View và Routing ở các chương sau.

HTTP, URL, GET và POST

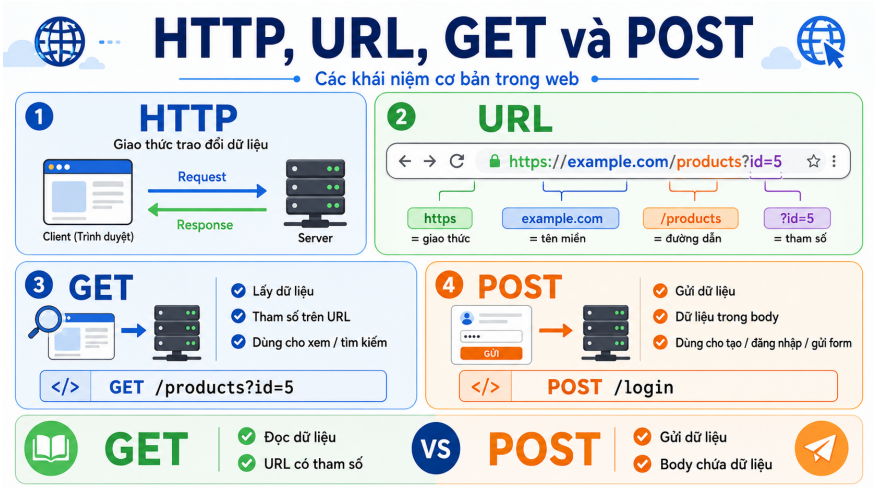
Khi client và server tách rời nhau, vấn đề tiếp theo là làm sao hai phía hiểu được nhau. Trình duyệt có thể chạy trên Windows, macOS hoặc điện thoại; server có thể viết bằng C#, JavaScript, PHP hoặc Python. Nếu mỗi nền tảng tự đặt một cách nói riêng, web sẽ không thể hoạt động thống nhất. HTTP ra đời để giải quyết vấn đề đó: nó quy định cách client gửi request và cách server trả response.

HTTP là viết tắt của *HyperText Transfer Protocol*. Đây là giao thức tầng ứng dụng theo mô hình request–response; mỗi thông điệp là request hoặc response và phương thức request biểu đạt ý định của client [5].

Web cần một quy ước giao tiếp thống nhất, không phụ thuộc vào công nghệ phía server; HTTP đảm nhiệm vai trò đó. Khi người dùng bấm liên kết, trình duyệt không cần biết server được viết bằng ngôn ngữ nào hay dữ liệu được lưu ở đâu. Trình duyệt tạo request HTTP theo quy ước, còn server xử lý request và trả về response HTTP. Nhờ giao thức chung, cùng một trình duyệt có thể làm việc với nhiều website mà không phải cài phần mềm riêng cho từng hệ thống.

Trong phần này, trọng tâm là hiểu ngữ nghĩa của `GET` và `POST`. `GET` thường yêu cầu đọc hoặc hiển thị tài nguyên và không nên tạo thay đổi nghiệp vụ khi được gọi; vì vậy không được xóa dữ liệu chỉ bằng một liên kết GET. `POST` thường dùng khi client gửi dữ liệu để server xử lý, chẳng hạn tạo bản ghi hoặc thực hiện một thao tác nghiệp vụ. Trong API, `PUT`, `PATCH` và `DELETE` còn được dùng cho cập nhật hoặc xóa. Phân biệt phương thức theo ý nghĩa giúp routing, bảo mật và kiểm thử phản ánh đúng hành vi của ứng dụng.

URL giải quyết một vấn đề khác: làm sao chỉ đúng tài nguyên hoặc chức năng cần gọi. Một URL thường gồm tên miền, đường dẫn và đôi khi có tham số. Chẳng hạn, `https://example.com` thường chỉ trang chính của website; `https://example.com/contact` có thể chỉ trang liên hệ; còn `https://example.com/products?page=2` cho biết người dùng đang muốn xem trang thứ hai của danh sách sản phẩm. Khi làm bài thực hành, nếu một trang không mở được, các bạn nên bắt đầu từ ba câu hỏi: URL đã đúng chưa, server có đang chạy không, và request đó có khớp với route/action nào trong ứng dụng không. Hình 1.2 tóm tắt quan hệ giữa HTTP, URL và hai phương thức thường gặp.



Hình 1.2. *HTTP, URL và hai phương thức GET/POST thường dùng*

HTML, CSS và JavaScript ở phía trình duyệt

Sau khi server xử lý xong request, nó phải trả về thứ mà trình duyệt có thể hiển thị. Đây là lý do HTML và CSS xuất hiện trong chương đầu. Server không gửi trực tiếp một đối tượng C# cho người dùng nhìn thấy; nó thường tạo ra HTML để mô tả nội dung và cấu trúc trang. Trình duyệt đọc HTML, kết hợp với CSS, rồi dựng thành giao diện.

Trong học phần này, HTML và CSS được học theo nhu cầu xây dựng giao diện cho các file `.cshtml` ở các chương sau. Trọng tâm là hiểu vai trò của giao diện trong luồng request/response: View nhận dữ liệu từ Controller, sinh ra HTML, trình duyệt hiển thị HTML đó cho người dùng.

Có thể hình dung HTML như bộ khung của trang. Tiêu đề thường được đặt trong các thẻ `h1`, `h2`; đoạn văn bản đặt trong `p`; liên kết dùng thẻ `a`; biểu mẫu dùng `form`; ô nhập liệu dùng `input`; hộp chọn dùng `select`; nút bấm dùng `button`; còn dữ liệu dạng danh sách nhiều dòng, nhiều cột thường được trình bày bằng `table`, `tr`, `th` và `td`. Khi cần gom một nhóm nội dung thành một khối, thẻ `div` thường được sử dụng.

Một số thuộc tính HTML sẽ xuất hiện thường xuyên trong các file giao diện. Thuộc tính `href` cho biết đích của liên kết; `class` nối phần tử với lớp CSS; `name` đặt tên cho dữ liệu biểu mẫu gửi lên server; `type` mô tả kiểu ô nhập hoặc nút. Nếu `name` không khớp hợp đồng đầu vào, server có thể không binding được dữ liệu ở các chương sau.

Ví dụ một đoạn HTML cho giao diện danh sách sản phẩm:

```
<h1>Danh sách sản phẩm</h1>

<a href="/products/create">Thêm mới</a>

<table class="table">
    <tr>
        <th>Tên sản phẩm</th>
        <th>Đơn giá</th>
    </tr>
    <tr>
        <td>Ban phim</td>
        <td>250000</td>
    </tr>
</table>
```

CSS giải quyết phần trình bày: cùng một dữ liệu nhưng nếu không có khoảng cách, đường viền, màu chữ và cách hiển thị bảng phù hợp thì người dùng khó đọc và khó thao tác. Khi làm bài thực hành, các bạn có thể dùng class CSS để định dạng các phần tử HTML:

```
.table {
    width: 100%;
    border-collapse: collapse;
}
```

JavaScript đảm nhiệm hành vi tương tác ở phía trình duyệt, chẳng hạn ẩn hoặc hiện một vùng nội dung, kiểm tra sơ bộ dữ liệu hay gửi request mà không tải lại toàn bộ trang. Một thao tác JavaScript chỉ thay đổi trạng thái cục bộ có thể không liên hệ với server; khi cần dữ liệu hoặc chức năng phía server, mã JavaScript vẫn phải dùng HTTP để trao đổi. Trong giáo trình này, JavaScript được giới thiệu ở mức đủ để phân biệt trách nhiệm của trình duyệt với xử lý C# phía server; trọng tâm thực hành vẫn là ASP.NET Core MVC.

Trong một giao diện MVC tốt, phần trình bày cần bảo đảm các yêu cầu trực tiếp của bài toán: nội dung đúng, bảng rõ ràng, form nhập liệu đủ trường và các liên kết thao tác dễ nhận biết.

HTTPS, cache HTTP và ranh giới tin cậy

HTTPS là HTTP chạy qua TLS. TLS dùng chứng thư số (*certificate*) để hỗ trợ client xác thực server, mã hóa dữ liệu trên đường truyền và phát hiện nội dung bị thay đổi [39]. HTTPS bảo vệ kênh truyền, nhưng không bảo đảm toàn bộ website an toàn: ứng dụng vẫn có thể kiểm tra mật khẩu kém, phân quyền sai hoặc xử lý dữ liệu đầu vào không an toàn. Vì vậy, dữ liệu do client gửi lên vẫn phải được server kiểm tra tính hợp lệ (*validation*), xác thực và phân quyền.

Trong môi trường phát triển, lệnh `dotnet dev-certs https --trust` tạo và tin cậy chứng thư số cục bộ. Khi đưa ứng dụng vào môi trường triển khai chính thức (*production*), chứng thư phải phù hợp với tên miền và được quản lý vòng đời. Các cơ chế cấu hình chi tiết như HSTS sẽ được học khi người học đã hiểu middleware và triển khai ứng dụng.

Cache HTTP cho phép trình duyệt hoặc proxy dùng lại response khi chính sách cho phép. Điều này có thể giảm thời gian tải và băng thông, nhưng dữ liệu cũ sẽ tồn tại có

chủ ý. Ở mức nhập môn, cần nhớ rằng tài nguyên công khai ít thay đổi dễ cache hơn trang có dữ liệu cá nhân; server có thể trả mã `304 Not Modified` khi tài nguyên chưa đổi. Các header, cache key và chính sách vô hiệu hóa là nội dung đọc thêm sau khi đã nắm vững request–response [4].

Mọi dữ liệu do client gửi lên, bao gồm query string, form, hidden field, cookie, header và phần thân (*body*) JSON, đều phải được server kiểm tra. Việc một giá trị bị ẩn trên giao diện hoặc nằm trong cookie không làm cho nó mặc nhiên đáng tin cậy. Developer Tools hoặc một HTTP client vẫn có thể tạo request khác với request mà giao diện thông thường sinh ra.

1.3 Một số hệ sinh thái phát triển web

Các hệ sinh thái web hiện đại cùng phải giải quyết việc nhận request, xử lý nghiệp vụ, truy cập dữ liệu và trả response, nhưng mỗi tên gọi trong một hệ sinh thái có vai trò khác nhau. Ngôn ngữ dùng để diễn đạt chương trình; nền tảng hoặc môi trường thực thi cung cấp nơi chạy và thư viện nền; framework đưa ra cấu trúc cùng các thành phần phát triển; công cụ hỗ trợ tạo, biên dịch, kiểm thử và vận hành sản phẩm.

Bảng 1.3 giới thiệu một số công nghệ tiêu biểu để người học nhận biết các vai trò đó, không phải bảng xếp hạng hoặc yêu cầu học chi tiết mọi lựa chọn. Các mô tả được đối chiếu với tài liệu chính thức của ASP.NET Core, Django, Node.js và Laravel [14][2][32][11].

Bảng 1.3. *Một số hệ sinh thái tiêu biểu trong phát triển ứng dụng web*

Hệ sinh thái	Công nghệ tiêu biểu	Vai trò
.NET	C#, .NET, ASP.NET Core MVC	C# là ngôn ngữ; .NET là nền tảng; ASP.NET Core phục vụ phát triển web; MVC tổ chức ứng dụng theo Model–View–Controller
Python	Python, Django	Python là ngôn ngữ; Django là framework phát triển ứng dụng web trong hệ sinh thái Python
JavaScript phía máy chủ	JavaScript/TypeScript, Node.js, Express/NestJS	Node.js là môi trường thực thi; ứng dụng thường kết hợp thêm web framework
PHP	PHP, Laravel	PHP là ngôn ngữ; Laravel là web framework cung cấp MVC và các thành phần thực dụng cho ứng dụng web

Trong dự án thực tế, từng hệ sinh thái còn có nhiều framework, thư viện và công cụ dữ liệu khác. Việc lựa chọn phụ thuộc yêu cầu sản phẩm, năng lực đội ngũ và môi trường vận hành. Mục này chỉ tạo bản đồ khái niệm để người học không đồng nhất ngôn ngữ với framework hoặc môi trường thực thi.

1.4 Lựa chọn ASP.NET Core MVC làm nền tảng chính

Phát triển ứng dụng là một lĩnh vực rộng, gồm nhiều loại sản phẩm, nền tảng và hệ sinh thái. Giáo trình sử dụng ASP.NET Core MVC làm nền tảng chính để trình bày có hệ thống quy trình xây dựng ứng dụng web, không nhằm khẳng định đây là công nghệ phù hợp với mọi bài toán.

Việc lựa chọn này hướng đến bốn mục tiêu:

1. Hình thành tư duy tổ chức ứng dụng. Mô hình MVC giúp người học nhận biết vai trò của Model, View và Controller, từ đó hiểu nguyên tắc phân tách trách nhiệm.
2. Hiểu chuỗi xử lý của ứng dụng web. Người học có thể lần theo request từ trình duyệt qua routing, controller, xử lý nghiệp vụ, truy cập dữ liệu và response; các nội dung về biểu mẫu, model binding, validation, xác thực và phân quyền được nối thành một quy trình.
3. Phát triển sản phẩm theo lộ trình xuyên suốt. AppDemo được mở rộng qua chín chương, giúp người học quan sát một ứng dụng tiến hóa từ cấu trúc ban đầu đến dữ liệu, bảo mật, kiểm thử và tích hợp AI.
4. Tạo nền tảng tiếp cận công nghệ khác. Kiến thức về HTTP, tổ chức mã nguồn, hợp đồng dữ liệu, bảo mật và kiểm thử có thể được đối chiếu khi người học chuyển sang framework hoặc hệ sinh thái khác.

ASP.NET Core còn cung cấp dependency injection, quản lý cấu hình, truy cập dữ liệu với Entity Framework Core, xác thực và phân quyền với ASP.NET Core Identity, Web API và hỗ trợ kiểm thử. Nhờ đó, các hoạt động chính của quá trình phát triển phần mềm có thể được quan sát, thực hành và kiểm chứng trên một nền tảng thống nhất [14][21].

1.5 Tổng quan .NET và ASP.NET Core MVC

C# trong hệ sinh thái .NET

Đến đây, ta đã biết ứng dụng web cần một phía server để nhận request, xử lý nghiệp vụ và tạo response. Câu hỏi tiếp theo là viết phần server đó bằng ngôn ngữ và nền tảng nào. Trong giáo trình này, câu trả lời là C# và .NET.

Trước khi học ASP.NET Core, cần hiểu vai trò của C#. C# là ngôn ngữ chính của hệ sinh thái .NET, có kiểu dữ liệu chặt chẽ, hỗ trợ hướng đối tượng và làm việc tự nhiên với thư viện chuẩn. Trong ứng dụng MVC, controller, model và Razor view đều dựa trên C#, nên nếu chỉ nhớ lệnh tạo dự án hoặc biết sửa HTML, sinh viên sẽ khó tự phát triển chức năng mới.

C# không chỉ dùng để viết website. Nó còn dùng cho console, desktop, API, dịch vụ nền, ứng dụng di động, game, cloud service và nhiều hệ thống doanh nghiệp. Điểm quan trọng là các loại ứng dụng này chia sẻ cùng .NET SDK, cách build, package và kiểm thử. Với web, C# giúp mô hình hóa nghiệp vụ bằng class, phát hiện lỗi sớm nhờ kiểu dữ liệu và viết lớp dữ liệu gọn bằng property `get; set;`. Trong giáo trình này, C# được học đúng lúc bài toán cần đến: ban đầu là class, method, collection và property; các chương sau mới mở rộng sang validation, truy vấn dữ liệu, dependency injection và cấu hình.

Chuyển từ Java sang C#

Ở học phần lập trình hướng đối tượng, các bạn đã thực hành bằng Java. Vì vậy, khi chuyển sang C#, điều quan trọng không phải là học lại lập trình từ đầu mà là nhận ra những ý tưởng OOP quen thuộc được viết bằng cú pháp mới như thế nào. Java và C# đều là ngôn ngữ hướng đối tượng, kiểu tĩnh, dùng lớp, đối tượng, phương thức và danh sách. Phần cần chú ý trong chương này là cách C# diễn đạt dữ liệu của đối tượng bằng property `get; set;`, quy ước đặt tên, file `Program.cs` và cách tạo danh sách.

Trong Java, lớp dữ liệu thường kết hợp trường `private` với getter/setter. C# diễn đạt cùng hợp đồng dữ liệu gọn hơn bằng property [22]:

```
public class Student
{
    public string FullName { get; set; }
    public int Age { get; set; }

}
```

`FullName` và `Age` trong ví dụ C# là property, không phải các biến public thông thường. Cú pháp truy cập trông trực tiếp, nhưng thao tác đọc và ghi vẫn đi qua `get` và `set`. Model trong ASP.NET Core MVC thường dùng property để framework có thể gán dữ liệu từ form hoặc từ nguồn dữ liệu. Do đó, dòng `public string Name { get; set; }` nên được hiểu là cách C# biểu diễn trạng thái của đối tượng gọn hơn mẫu getter/setter quen thuộc trong Java.

Bảng 1.4 gom lại các điểm chuyển đổi cốt lõi để khi đọc mã C# từ nền tảng Java, các bạn đối chiếu đúng khái niệm thay vì học thuộc từng dòng cú pháp.

Bảng 1.4. *Một số điểm chuyển đổi thường gặp từ Java sang C#*

Khái niệm	Trong Java	Trong C#
Chuỗi	`String`	`string` trong mã thông thường
Logic đúng/sai	`boolean`	`bool`
In ra console	`System.out.println(...)`	`Console.WriteLine(...)`
Không gian tên	`package`, `import`	`namespace`, `using`
Dữ liệu của đối tượng	`private` field và getter/setter	Property, ví dụ `public string Name { get; set; }`
Quy ước đặt tên	`displayName()`	`DisplayName()` và `FullName`

Với chương trình console, Java thường có phương thức `main`. C# cũng hỗ trợ `Main`, nhưng các phiên bản .NET hiện nay cho phép dùng top-level statements để giảm mã khung:

```
Console.WriteLine("Hello C#");
```

Điều này giải thích vì sao `Program.cs` có thể không khai báo rõ lớp `Program` và phương thức `Main`. Sinh viên chỉ cần nhận ra cách viết mới của các khái niệm OOP đã học: lớp, đối tượng, kiểu dữ liệu, phương thức, property và collection; các chương sau sẽ đặt chúng vào Controller, Model và View.

Đến đây, ba thành phần đã được nối với nhau: web trao đổi dữ liệu bằng request/response; trình duyệt thường hiển thị response bằng HTML/CSS; phần server của giáo trình được viết bằng C# trên nền tảng .NET. ASP.NET Core là framework tổ chức các thành phần này thành một ứng dụng web có cấu trúc.

ASP.NET Core trong hệ sinh thái .NET

.NET là nền tảng phát triển mã nguồn mở, đa nền tảng, gồm runtime, thư viện, compiler, SDK và các application stack như ASP.NET Core [18]. Vì vậy, sinh viên không chỉ học một thư viện riêng lẻ mà học cả ngôn ngữ C#, công cụ dòng lệnh, cơ chế build, quản lý gói và cách chạy ứng dụng.

ASP.NET là nhánh của .NET dành cho phát triển ứng dụng web. Phiên bản ASP.NET truyền thống ra đời từ năm 2002 và gắn chặt với .NET Framework trên Windows. Nó từng có nhiều mô hình quen thuộc như Web Forms, MVC và Web API. Những công nghệ này đặt nền móng cho lập trình web bằng C#, nhưng khi ứng dụng web hiện đại bắt đầu cần chạy trên cloud, container, Linux, DevOps pipeline và kiến trúc microservices, cách tiếp cận cũ bộc lộ giới hạn: phụ thuộc môi trường Windows nhiều hơn, nặng hơn, và kém linh hoạt hơn khi triển khai trên nhiều hạ tầng.

ASP.NET Core là framework web mã nguồn mở, đa nền tảng trong hệ sinh thái .NET. Nó hỗ trợ website, MVC, API và các dịch vụ backend trên Windows, Linux và macOS [14].

Trong phạm vi giáo trình này, trọng tâm là ASP.NET Core MVC. MVC tách ứng dụng thành Model, View và Controller để phân chia trách nhiệm; request được định tuyến tới controller, controller làm việc với model và chọn view để tạo giao diện [21]. Cấu trúc này giúp sinh viên trả lời ba câu hỏi: request đi vào đâu, dữ liệu được chuẩn bị thế nào và giao diện được tạo ở đâu.

Ở các bài thực hành đầu tiên, trọng tâm là tạo một dự án MVC, thêm controller, thêm action, tạo view tương ứng và truyền dữ liệu từ Controller sang View bằng `ViewBag` hoặc `ViewData`. Khi làm chủ được chuỗi thao tác này, sinh viên bắt đầu nhìn ứng dụng MVC như một luồng request có tổ chức thay vì một tập file rời rạc.

Các khả năng quan trọng của ASP.NET Core

Khi ứng dụng chỉ có vài trang, ta có thể viết mã đơn giản mà vẫn chạy được. Khi hệ thống có đăng nhập, phân quyền, log, xử lý lỗi, cơ sở dữ liệu, API và triển khai nhiều môi trường, vấn đề chuyển thành tổ chức luồng xử lý nhất quán. ASP.NET Core cung cấp kiến trúc chung cho MVC, REST API, ứng dụng realtime và backend service, đồng thời hỗ trợ Dependency Injection để controller không phụ thuộc cứng vào các lớp cụ thể.

Bên cạnh MVC, ASP.NET Core còn có Razor Pages cho ứng dụng thiên về từng trang, Minimal API cho REST API nhỏ, Blazor cho giao diện tương tác bằng C#, và gRPC cho giao tiếp hiệu suất cao giữa dịch vụ. Middleware pipeline xử lý request theo nhiều bước như log, lỗi, file tĩnh, routing, xác thực và phân quyền trước khi đến logic nghiệp vụ. Các mô hình này không phải danh sách tính năng rời rạc; mỗi mô hình là câu trả lời cho một áp lực thiết kế khác nhau.

Kestrel là web server mặc định của ASP.NET Core. Nó tồn tại vì ứng dụng web cần một thành phần lắng nghe kết nối HTTP thật sự, nhận request từ mạng và chuyển request đó vào pipeline của ứng dụng. Trong môi trường phát triển, khi các bạn chạy lệnh `dotnet run`, ứng dụng thường được phục vụ trực tiếp qua Kestrel. Trong môi trường triển khai thực tế, Kestrel có thể chạy độc lập hoặc đứng sau các web server/reverse proxy như IIS, Nginx hoặc Apache. Cách đứng sau reverse proxy giúp hệ thống tận dụng thêm các khả năng như cân bằng tải, quản lý chứng chỉ HTTPS, nén dữ liệu hoặc điều phối request giữa nhiều dịch vụ.

ASP.NET Core cũng có hệ thống cấu hình linh hoạt theo môi trường vì cùng một ứng dụng thường chạy trong nhiều ngữ cảnh khác nhau: máy của các bạn, máy kiểm thử, server thật hoặc cloud. Chuỗi kết nối, khóa API, mức ghi log và tham số bảo mật không nên giống nhau ở mọi nơi. Nếu cấu hình bị viết cứng trong mã nguồn, việc triển khai sẽ dễ sai và khó bảo vệ thông tin nhạy cảm. Vì vậy, framework cung cấp cách đọc cấu hình theo môi trường, tích hợp logging, hỗ trợ HTTPS và nhiều điểm mở rộng để theo dõi, chẩn đoán, bảo vệ ứng dụng trong quá trình vận hành.

1.6 Cài đặt môi trường và thực hành đầu tiên

Sau khi hiểu ứng dụng web cần server, request, response, HTML và một framework để tổ chức mã nguồn, bước tiếp theo là chuẩn bị môi trường thực hành. VS Code là nơi đọc và sửa mã; .NET SDK cung cấp công cụ tạo dự án, build và chạy ứng dụng; C# Dev Kit hỗ trợ làm việc với dự án C# trong VS Code [41].

Trong hệ sinh thái .NET, lập trình viên cũng có thể dùng Visual Studio như một IDE đầy đủ cho thiết kế, lập trình, gỡ lỗi và kiểm thử. Tuy nhiên, giáo trình này chọn VS Code kết hợp với .NET CLI vì cách làm này buộc các bạn nhìn thấy những bước nền tảng: tạo dự án bằng template, build mã nguồn, chạy server, đọc URL được sinh ra và mở dự án như một thư mục mã nguồn. Khi hiểu các bước này, các bạn sẽ dễ xử lý lỗi hơn so với chỉ bấm một nút chạy mà không biết phía sau đã xảy ra điều gì.

Thành phần cần cài

Bộ công cụ thực hành gồm các thành phần sau. Mỗi thành phần giải quyết một phần cụ thể của quy trình phát triển:

- Visual Studio Code: nơi mở thư mục dự án, đọc cấu trúc `Controllers`, `Models`, `Views` và chỉnh sửa mã nguồn.
- .NET SDK: bộ công cụ cung cấp lệnh `dotnet`; nếu thiếu SDK, máy không thể tạo, build hoặc chạy dự án ASP.NET Core.
- C# Dev Kit: extension bắt buộc để VS Code hiểu dự án C#, hỗ trợ điều hướng, gợi ý và phát hiện lỗi; các phụ thuộc C# cần thiết được cài cùng extension này.
- Công cụ giao diện tùy chọn: extension HTML/CSS, icon theme hoặc formatter có thể hỗ trợ thao tác, nhưng không phải điều kiện để học; formatter HTML cần được kiểm tra trước khi áp dụng cho Razor `.cshtml`.
- Trình duyệt web: client dùng để gửi request thật đến ứng dụng đang chạy và quan sát response được hiển thị.

Biết trước môi trường cơ sở dữ liệu từ Chương 6

ASP.NET Core và .NET chạy trên Windows, macOS và Linux, nhưng SQL Server LocalDB thì chỉ hỗ trợ Windows. Vì vậy, từ Chương 6 giáo trình quy định hai tuyến thực hành chính thức. Môi trường thực hành trên Windows dùng LocalDB với Windows Authentication làm phương án mặc định. Người học dùng macOS/Linux, hoặc Windows không có LocalDB, chạy SQL Server trong Docker theo hướng dẫn ở Chương 6.

Hai tuyến dùng chung entity, `ApplicationDbContext`, bản di trú (migration), controller và EF Core SQL Server provider. Khác biệt nằm ở cách khởi động SQL Server, cơ chế xác thực và nguồn cung cấp connection string. SQLite in-memory chỉ phục vụ kiểm thử cô lập; không được xem là bản thay thế hoàn toàn cho SQL Server. Nhờ quy ước này, lựa chọn LocalDB không làm mất tính đa nền tảng đã nêu ở đầu chương.

Cài đặt trên Windows

Bước 1: Cài Visual Studio Code

VS Code được cài trước để các bạn có môi trường mở thư mục mã nguồn, chỉnh sửa file và dùng terminal tích hợp. Nếu chưa có trình soạn thảo ổn định, việc kiểm tra cấu trúc dự án và lỗi trong quá trình thực hành sẽ khó theo dõi.

Truy cập trang chính thức:

```
https://code.visualstudio.com/
```

Tải bản Windows, chạy file cài đặt. Khi cài, nên chọn các tùy chọn:

- Add to PATH, nếu trình cài đặt hiển thị tùy chọn này.
- Open with Code trong menu chuột phải, nếu có.
- Register Code as an editor for supported file types, nếu có.

Bước 2: Cài .NET SDK

.NET SDK là thành phần quan trọng nhất để biến mã nguồn thành ứng dụng chạy được. SDK khác Runtime ở chỗ Runtime chỉ đủ để chạy một ứng dụng đã được build sẵn, còn SDK có thêm công cụ tạo dự án, biên dịch, phục hồi package và chạy ứng dụng trong quá trình phát triển. Vì các bạn cần tự tạo và sửa dự án, bắt buộc phải cài SDK.

Truy cập trang tải .NET chính thức và chọn SDK phù hợp với hệ điều hành [23]:

```
https://dotnet.microsoft.com/download
```

Không chỉ tải `Runtime`, vì Runtime chỉ chạy ứng dụng đã build; bài thực hành cần SDK để tạo dự án, build và chạy mã nguồn.

Sau khi cài, mở Command Prompt, PowerShell hoặc Terminal trong VS Code và kiểm tra:

```
dotnet --version
```

Nếu lệnh trả về số phiên bản, SDK đã được cài thành công. Nếu báo không nhận lệnh `dotnet`, xử lý theo thứ tự:

1. Đóng toàn bộ Command Prompt, PowerShell và VS Code rồi mở lại.
2. Nếu vẫn lỗi, mở Start Menu, tìm `Edit the system environment variables`.
3. Chọn `Environment Variables`.
4. Trong phần `System variables`, chọn `Path`, bấm `Edit`.
5. Kiểm tra có đường dẫn `C:\Program Files\dotnet\`. Nếu chưa có, bấm `New` và thêm đường dẫn này.
6. Bấm `OK`, mở terminal mới và chạy lại `dotnet --version`.

Bước 3: Cài extension trong VS Code

Sau khi có SDK, VS Code vẫn chỉ là trình soạn thảo tổng quát. Các extension giúp VS Code hiểu ngữ cảnh C# và ASP.NET Core tốt hơn: biết dự án đang dùng SDK nào, gợi ý namespace, báo lỗi cú pháp, hỗ trợ mở nhanh class và làm việc với file View.

Mở VS Code, chọn biểu tượng Extensions ở thanh bên trái. Thành phần bắt buộc là `C# Dev Kit` của Microsoft; extension C# và các phụ thuộc cần thiết sẽ được cài cùng gói. Các công cụ giao diện còn lại chỉ là tùy chọn:

- Bắt buộc: `C# Dev Kit` của Microsoft.
- Tùy chọn: extension hỗ trợ HTML/CSS và một icon theme để dễ quan sát cây thư mục.
- Tùy chọn, cần kiểm tra trước khi dùng: formatter cho HTML/CSS/JavaScript; không áp dụng máy móc formatter HTML cho Razor `.cshtml`.

Sau khi cài extension, nên khởi động lại VS Code.

Bước 4: Kiểm tra lệnh code

Lệnh `code` giúp mở thư mục hiện tại bằng VS Code trực tiếp từ terminal. Điều này làm cho quy trình thực hành rõ ràng hơn: tạo dự án ở đâu thì mở đúng thư mục đó, tránh lỗi viết file ở sai vị trí.

Trong terminal, chạy:

```
code --version
```

Trên Windows, cần chọn `Add to PATH` khi cài VS Code rồi đóng và mở lại terminal. Nếu vẫn không nhận lệnh, có thể cài lại với tùy chọn đó hoặc mở dự án bằng `File > Open Folder`; không cần tìm lệnh dành riêng cho macOS trong Command Palette.

Ghi chú cho máy macOS
Nếu dùng Mac, vẫn cài VS Code, .NET SDK và các extension giống Windows. Với máy Apple Silicon, tải bản Arm64. Nếu Terminal không nhận lệnh ngắn, thêm vào `~/.zprofile`:

```
export PATH="$PATH:/usr/local/share/dotnet:$HOME/.dotnet/tools"
```

Nếu lệnh `code` chưa chạy, mở VS Code, nhấn `Command + Shift + P`, chạy `Shell Command: Install 'code' command in PATH`.

Tạo solution và hai dự án đầu tiên

Sau khi cài đặt, người học tạo một solution chứa cả ứng dụng MVC và dự án kiểm thử. Quy trình đầy đủ dưới đây bảo đảm các lệnh build/test ở cuối chương có đúng đối tượng để thực thi. Với .NET 10, `dotnet new sln` mặc định tạo file `.slnx`.

```
mkdir AppDemo
cd AppDemo
dotnet new sln -n AppDemo
dotnet new mvc -n AppDemo
dotnet new xunit -n AppDemo.Tests
dotnet sln AppDemo.slnx add AppDemo/AppDemo.csproj
dotnet sln AppDemo.slnx add AppDemo.Tests/AppDemo.Tests.csproj
dotnet add AppDemo.Tests/AppDemo.Tests.csproj reference AppDemo/AppDemo.csproj
dotnet build AppDemo.slnx
dotnet test AppDemo.slnx
dotnet run --project AppDemo/AppDemo.csproj
```

Kết quả có cấu trúc tối thiểu:

```
AppDemo/
|-- AppDemo.slnx
|-- AppDemo/
|   `-- AppDemo.csproj
`-- AppDemo.Tests/
    `-- AppDemo.Tests.csproj
```

Các phần tiếp theo viết đường dẫn như `Views/Home/Index.cshtml` theo gốc dự án `AppDemo`. Nếu terminal vẫn đứng ở thư mục solution, hãy thêm tiền tố `AppDemo/` hoặc dùng tham số `--project AppDemo/AppDemo.csproj` khi chạy.

Quy trình tạo dự án theo template MVC và chạy bằng CLI phù hợp với hướng dẫn ASP.NET Core MVC chính thức [16]. Khi ứng dụng chạy, terminal hiển thị địa chỉ

dạng `http://localhost:xxxx` hoặc `https://localhost:xxxx`; mở địa chỉ đó để kiểm tra server đã nhận và trả response.



Hình 1.3. *Trang chủ AppDemo sau bài thực hành Chương 1*

Hình 1.3 hiển thị thanh điều hướng gồm trang chủ, trang Privacy và trang Giới thiệu; vùng nội dung chính giới thiệu công ty tổ chức sự kiện, còn chân trang chứa thông tin AppDemo. Giao diện xác nhận ứng dụng đã trả và kết xuất HTML; `dotnet build` và `dotnet test` mới xác nhận solution biên dịch và hạ tầng kiểm thử hoạt động.

Chạy ứng dụng bằng `dotnet run` *và* `dotnet watch`

Khi mới học, nhiều bạn gặp tình huống sửa file view nhưng trình duyệt vẫn hiển thị nội dung cũ, hoặc không biết khi nào phải dừng server rồi chạy lại. Cần tách rõ hai chế độ làm việc. Lệnh `dotnet run` build dự án một lần rồi khởi động ứng dụng. Cách này phù hợp khi cần kiểm tra dự án có biên dịch được không, chạy thử một phiên ổn định, hoặc chuẩn bị chụp màn hình cho giáo trình. Nếu thay đổi file trong khi ứng dụng đang chạy bằng `dotnet run`, một số thay đổi giao diện có thể xuất hiện sau khi tải lại trình duyệt, nhưng với người mới học, cách kiểm chứng chắc chắn nhất vẫn là dừng server bằng `Ctrl + C`, chạy lại `dotnet run`, rồi mở lại đúng URL localhost.

```
dotnet run
```

Trong quá trình thực hành hằng ngày, `dotnet watch run` phù hợp hơn vì nó theo dõi các file trong dự án. Khi các bạn sửa `.cshtml`, CSS hoặc một số file C#, công cụ này sẽ cố gắng áp dụng Hot Reload hoặc tự khởi động lại ứng dụng nếu cần. Nhờ đó, vòng lặp học trở nên ngắn hơn: sửa file, lưu file, nhìn terminal báo cập nhật, rồi tải lại trình duyệt để kiểm tra kết quả. Nếu terminal báo không thể áp dụng Hot Reload, hãy để `dotnet watch` restart ứng dụng, hoặc dừng bằng `Ctrl + C` và chạy lại từ đầu.

```
dotnet watch run
```

Với các file `.cshtml`, cần nhớ rằng các bạn đang sửa mã nguồn tạo ra HTML ở phía server, không phải sửa trực tiếp trang đang nằm trong trình duyệt. Vì vậy, sau khi lưu file, trình duyệt vẫn cần được tải lại để gửi request mới và nhận response HTML mới. Trong giáo trình này, khi cần chụp ảnh giao diện hoặc kiểm chứng một chức năng, nên dùng `dotnet watch run` trong giai đoạn chỉnh sửa nhanh; trước khi chốt ảnh hoặc chép code vào bài, chạy lại `dotnet run` hoặc `dotnet build` để chắc chắn dự án không còn lỗi.

Mở dự án trong VS Code

Sau khi dự án chạy được, các bạn cần mở đúng thư mục dự án để nhìn thấy mối quan hệ giữa mã nguồn và trang web vừa chạy. Đây là bước nối giữa phần cài đặt và các chương sau: từ đây các bạn sẽ thêm controller, tạo view, sửa file cấu hình và quan sát kết quả trên trình duyệt.

Từ thư mục dự án, chạy:

```
code .
```

Sau khi chạy lệnh `code .`, VS Code sẽ mở toàn bộ thư mục dự án hiện tại thành một workspace. Các bạn cần nhận biết một số khu vực chính trong cửa sổ làm việc để thao tác đúng khi thực hành:

- Activity Bar: thanh biểu tượng dọc bên trái, dùng để chuyển nhanh giữa Explorer, Search, Source Control, Run/Debug và Extensions.
- Explorer: khu vực quản lý thư mục và tệp của dự án. Khi tạo controller, model hoặc view, các bạn cần kiểm tra đúng vị trí trong cây thư mục này.
- Cây thư mục dự án: hiển thị các thư mục quen thuộc của dự án MVC như `Controllers`, `Models`, `Views`, `wwwroot` và các tệp cấu hình.
- Editor Tabs: các tab tệp đang mở, giúp chuyển nhanh giữa `Program.cs`, controller và các view `.cshtml`.
- Vùng soạn thảo mã: nơi viết và chỉnh sửa mã nguồn.
- Panel phụ bên dưới: thường dùng để mở Terminal, xem lỗi trong Problems hoặc xem Output khi chạy ứng dụng.
- Nút Run/Debug: dùng để chạy hoặc gỡ lỗi dự án trong VS Code khi môi trường đã được cấu hình.

Nếu chưa dùng được lệnh `code .`, các bạn vẫn có thể mở dự án bằng menu `File > Open Folder` trong VS Code và chọn đúng thư mục `AppDemo`.

Thay đổi trang chủ và thêm liên kết điều hướng

Sau khi dự án MVC đầu tiên chạy được, các bạn nên thực hiện một thay đổi nhỏ để thấy rõ quan hệ giữa file mã nguồn và giao diện trên trình duyệt. Trang mặc định mà trình duyệt đang hiển thị không phải được tạo ra một cách bí mật; nó đến từ action `Index` trong `HomeController` và file view `Views/Home/Index.cshtml`. Khi sửa file view này rồi tải lại trình duyệt, các bạn sẽ thấy response HTML thay đổi.

Mở file `Views/Home/Index.cshtml` và thay nội dung mặc định bằng:

Mã Razor – `Views/Home/Index.cshtml`:

```
@{
    ViewData["Title"] = "Trang chủ";
}
```

```
<div class="text-center">
    <h1>Công ty tổ chức sự kiện</h1>
    <p>Dịch vụ tổ chức sự kiện, lễ hội và hội thảo.</p>
</div>
```

Sau đó lưu file và tải lại trang `/` hoặc `/Home/Index` trên trình duyệt. Nếu đang chạy bằng `dotnet watch run`, terminal thường báo đã nhận thay đổi và trình duyệt chỉ cần gửi lại request. Nếu đang chạy bằng `dotnet run` mà nội dung chưa đổi, hãy dừng server bằng `Ctrl + C`, chạy lại `dotnet run`, rồi mở lại URL localhost. Khi nội dung mới xuất hiện, các bạn đã kiểm chứng được luồng cơ bản: trình duyệt gửi request đến action `Index`, action trả về view, view sinh ra HTML và trình duyệt hiển thị HTML đó.

Tiếp theo, thêm một trang mới để thấy navigation hoạt động như thế nào. Mở file `Controllers/HomeController.cs`, thêm action `About` trong lớp `HomeController`:

```
public IActionResult About()
{
    return View();
}
```

Tạo file `Views/Home/About.cshtml` với nội dung:

Mã Razor – `Views/Home/About.cshtml`:

```
@{
    ViewData["Title"] = "Giới thiệu";
}

<h1>Giới thiệu</h1>
<p>Đây là trang giới thiệu đầu tiên được thêm vào dự án MVC.</p>
```

Nếu chạy lại ứng dụng và truy cập `/Home/About`, ASP.NET Core MVC sẽ tìm action `About` trong `HomeController`, sau đó tìm view `Views/Home/About.cshtml`. Đây là quy ước quan trọng: tên controller, tên action và vị trí file view phải khớp với nhau.

Cuối cùng, thêm liên kết đến trang mới vào thanh điều hướng. Mở file `Views/Shared/_Layout.cshtml`, tìm danh sách `navbar-nav` và thêm một mục mới bên cạnh `Home` và `Privacy`:

Mã Razor – trích đoạn `Views/Shared/_Layout.cshtml`:

```
<li class="nav-item">
    <a class="nav-link text-dark"
       asp-area=""
       asp-controller="Home"
       asp-action="About">Giới thiệu</a>
</li>
```

Các thuộc tính `asp-controller` và `asp-action` là Tag Helper của ASP.NET Core. Chúng giúp tạo liên kết dựa trên controller và action thay vì viết cứng URL. Khi người dùng bấm liên kết `Giới thiệu`, trình duyệt gửi request đến action `About`; server xử lý và trả về view tương ứng. Bài thực hành nhỏ này giúp các bạn nhìn thấy đầy đủ vòng lặp đầu tiên: sửa mã nguồn, chạy ứng dụng, gửi request, nhận response và kiểm tra kết quả trên giao diện.

Bảng 1.5 cho thấy dự án MVC đầu tiên không được kiểm chứng bằng cảm giác "đã chạy", mà bằng các dấu hiệu quan sát được từ server, URL, navigation và giao diện.

Bảng 1.5. *Dấu hiệu cần kiểm chứng khi tạo dự án MVC đầu tiên*

Dấu hiệu	Nếu chưa đạt	Cách kiểm chứng
Server khởi động	Trình duyệt không mở được trang hoặc báo không kết nối	Đọc terminal, xác định đúng URL và cổng đang nghe
Trang chủ hiển thị	dự án có thể build nhưng response chưa đúng mong đợi	Mở trang chủ và kiểm tra nội dung đầu tiên được render
Navigation hoạt động	Link có thể trỏ sai controller hoặc action	Bấm từng link và quan sát URL, tiêu đề, nội dung trang
Thay đổi giao diện có hiệu lực	CSS hoặc view có thể sửa sai file	Làm mới trình duyệt và đối chiếu với file vừa chỉnh
Vòng lặp sửa nhanh hoạt động	Mỗi lần sửa view đều phải dừng và chạy lại thủ công	Chạy `dotnet watch run`, lưu file `.cshtml`, quan sát terminal và tải lại trình duyệt

1.7 Sử dụng tác tử AI trong thực hành ASP.NET Core MVC

Sau khi tự tạo dự án, sửa trang chủ, thêm action, tạo view và cập nhật thanh điều hướng, sinh viên đã có đủ kiến thức để đánh giá một thay đổi MVC cơ bản. Codex IDE extension cho phép làm việc cạnh mã nguồn, đưa file đang mở vào ngữ cảnh và xem thay đổi ngay trong trình soạn thảo [33]. Người học vẫn phải xác định mục tiêu, xem lệnh và diff, chạy kiểm tra rồi quyết định giữ thay đổi nào.

1. Mở VS Code.
2. Cài đặt và đăng nhập Codex extension theo hướng dẫn của giảng viên.
3. Mở một thư mục làm việc trống, ví dụ `AspNetPractice`.
4. Mở khung chat của Codex trong VS Code.
5. Gõ câu lệnh đầu vào (prompt) yêu cầu tạo ứng dụng MVC bằng .NET 10 và chạy thử chương trình.

Prompt nên nêu công nghệ, phạm vi, điều cấm và bằng chứng hoàn thành. Ví dụ:

```
Trong AppDemo .NET 10 MVC, hãy thêm trang About:
- thêm action About vào HomeController;
- tạo Views/Home/About.cshtml và liên kết trong _Layout.cshtml;
- không thêm cơ sở dữ liệu hoặc package;
- chạy dotnet build và báo các file đã sửa.
Sau đó giải thích request /Home/About đi qua những file nào.
```

Yêu cầu này giới hạn thay đổi vào đúng ba vị trí và quy định lệnh kiểm tra. Nếu công cụ báo thiếu SDK hoặc template không hỗ trợ framework, sinh viên phải đọc lỗi trước khi cho phép sửa tiếp.

Kết quả trên trình duyệt chưa phải bằng chứng duy nhất. Trong quá trình làm việc với Codex, sinh viên cần quan sát bốn yếu tố:

1. Codex đã chạy những lệnh terminal nào?
2. Codex đã sửa những file nào?
3. Ứng dụng có build và chạy được không?
4. Thay đổi trên trình duyệt có đúng với yêu cầu ban đầu không?

Nếu Codex đặt view sai thư mục, phản hồi phải chỉ rõ quy ước bị vi phạm, giới hạn phạm vi cần sửa và yêu cầu chạy build lần nữa. Cách thực hành này rèn đồng thời năng lực đọc cấu trúc MVC và năng lực giao nhiệm vụ cho tác tử AI có kiểm soát.

1.8 Thiết lập kiểm thử tự động cho AppDemo

Sau khi build và quan sát ứng dụng trên trình duyệt, sinh viên cần bổ sung một phép kiểm tra có thể chạy lại mà không phụ thuộc hoàn toàn vào thao tác thủ công. Dự án `AppDemo.Tests` được tạo bằng xUnit [42] và tham chiếu đến `AppDemo`. Mục tiêu trước hết là kiểm tra dự án kiểm thử và test runner đã được cấu hình đúng; sau đó người học chuyển từ test hạ tầng sang kiểm tra một hành vi cụ thể của ứng dụng.

unit test thường cô lập một hành vi nhỏ của chương trình. Tuy nhiên, test đầu tiên trong chương chỉ xác nhận rằng dự án kiểm thử, package xUnit và test runner đã được cấu hình đúng. Nó chưa kiểm tra quy tắc nghiệp vụ; giới hạn này cần được nói rõ để sinh viên không đồng nhất mọi test tự động với unit test có giá trị nghiệp vụ.

Bảng 1.6. *Các trường hợp kiểm tra ban đầu của AppDemo*

Mã	Điều kiện	Hành động	Mong đợi
TC01	Dự án kiểm thử tham chiếu AppDemo	Chạy `dotnet test`	Runner tìm và chạy test
TC02	Một assertion sai có chủ ý	Chạy lại test	Hiện `Failed` và vị trí lỗi
TC03	`ErrorViewModel` có mã request	Đọc `ShowRequestId`	Trả về `true`

Từ thư mục chứa solution, kiểm tra quan hệ dự án đã tạo ở phần trước:

```
dotnet sln AppDemo.slnx list
dotnet list AppDemo.Tests/AppDemo.Tests.csproj reference
```

Kết quả phải liệt kê cả hai dự án và reference đến `AppDemo.csproj`. Mỗi test nên trình bày theo Arrange–Act–Assert: chuẩn bị dữ liệu, thực hiện hành vi và kiểm tra kết quả. Tên `Method_Condition_ExpectedResult` giúp người đọc nhận ra ngay hành vi đang được kiểm chứng. Test dưới đây đã được chạy trong dự án đi kèm giáo trình:

```
[Fact]
public void TestRunner_WhenExecuted_RunsXunitTest()
{
    Assert.True(true);
}
```

Trước hết chạy `dotnet build AppDemo/AppDemo.csproj` để tách lỗi biên dịch khỏi lỗi kiểm thử, rồi chạy `dotnet test AppDemo.slnx`. Kết quả `Passed` cho biết assertion đã thỏa mãn. Với `Failed`, cần đọc tên test, giá trị mong đợi–thực tế và stack trace trước khi sửa mã.

Để quan sát cơ chế báo lỗi, sinh viên có thể đổi tạm `true` thành `false`, chạy lại test, giải thích thông báo rồi hoàn tác. Sau đó, viết test hành vi đầu tiên:

```
[Fact]
public void ShowRequestId_WhenRequestIdExists_ReturnsTrue()
```

```
{
    var model = new ErrorViewModel { RequestId = "request-01" };

    Assert.True(model.ShowRequestId);
}
```

Test này chứng minh một quy tắc cụ thể của AppDemo thay vì chỉ chứng minh test runner khởi động được.

1.9 Lộ trình phát triển AppDemo trong giáo trình

`AppDemo` là hệ thống giới thiệu và quản lý sự kiện, hội thảo của một đơn vị công nghệ. Toàn bộ giáo trình sử dụng một solution `AppDemo.slnx`, một dự án web `AppDemo` và một dự án kiểm thử `AppDemo.Tests`. Sản phẩm được mở rộng theo từng giai đoạn như Bảng 1.7; người học không tạo lại dự án mới khi chuyển chương.

Bảng 1.7. *Lộ trình phát triển AppDemo qua chín chương*

Giai đoạn	Chương	Năng lực được bổ sung vào AppDemo
Khởi tạo	1	Solution, ứng dụng MVC, dự án kiểm thử và quy trình kiểm thử cơ bản
Website giới thiệu	2–3	Các trang `Company`, routing, controller, view, model và view model
Nghiệp vụ sự kiện	4–5	Miền `Organizer`, `Festival`, `Speaker`, `Presentation`; gỡ lỗi, form, model binding và validation trên tầng dữ liệu dựng sẵn
Quản lý dữ liệu	6–7	EF Core, quan hệ, migration, seed, CRUD và truy vấn LINQ
Bảo vệ chức năng	8	Identity, đăng nhập, role `Admin` và phân quyền phía server
Hỗ trợ thông minh	9	AI gợi ý mô tả cho `Presentation`; người quản trị duyệt trước khi lưu

Các trang `Company` vẫn là phần giới thiệu của sản phẩm khi miền quản lý sự kiện được bổ sung. Riêng đầu Chương 4, giảng viên cung cấp mốc `chapter-04-start` phát triển từ `chapter-03-completed`. Mốc này có sẵn entity, cơ sở dữ liệu và tình huống lỗi để sinh viên học gỡ lỗi trước khi tự thiết lập EF Core ở Chương 6; các thành phần dựng sẵn chưa thuộc phạm vi đánh giá của Chương 4–5.

Trước khi tổng kết chương, cần đặt môi trường vừa cài vào một chuẩn có thể tái hiện. Một ứng dụng chạy được trên máy của người viết chưa đủ chứng minh người khác có thể mở kho mã, khôi phục package và chạy cùng kết quả. Chuẩn tối thiểu gồm phiên bản SDK, lệnh khởi động, URL kiểm tra, cấu hình nào được phép đưa vào Git và bằng chứng response trả về. Chuẩn này không thay thế hướng dẫn cài đặt; nó biến hướng dẫn thành một hợp đồng kỹ thuật có thể kiểm tra.

Hồ sơ nền cho một lần thực hành. Nhóm học tập nên lưu các bằng chứng ngắn thay vì chụp toàn bộ màn hình. Tệp dự án trả lời ứng dụng dùng target framework nào; kết quả `dotnet --info` cho biết SDK thực tế; log khởi động cho biết môi trường và địa chỉ; request trên trình duyệt cho biết endpoint có phản hồi. Nếu một máy thất bại, bốn nguồn này giúp phân biệt thiếu SDK, lỗi khôi phục package, xung đột cổng và lỗi trong ứng dụng.

Bảng 1.8. *Bộ bằng chứng nền cho dự án web đầu tiên*

Bằng chứng	Câu hỏi trả lời	Tiêu chí chấp nhận
Phiên bản SDK	Máy đang dùng SDK nào và có phù hợp tệp dự án không?	Lệnh CLI nhận đúng SDK, không phụ thuộc một bản cài không được ghi nhận.
Tệp solution	Những dự án nào thuộc cùng bài thực hành?	Lệnh build từ thư mục gốc tìm thấy đúng các dự án cần thiết.
Kết quả restore	Package có thể được khôi phục từ nguồn đã cấu hình không?	Không có package thiếu, nguồn lạ hoặc cảnh báo bị bỏ qua.
Kết quả build	Mã nguồn có biên dịch độc lập với IDE không?	Build thành công từ CLI và lỗi, nếu có, trỏ đúng file cùng dòng.
Log khởi động	Ứng dụng chạy ở môi trường và cổng nào?	Log nêu rõ URL, môi trường và không in bí mật.
Request kiểm tra	Tuyến gốc có nhận request thật không?	Trình duyệt nhận status thành công và HTML đúng ứng dụng.
Thay đổi giao diện	File nào tạo ra nội dung vừa sửa?	Refresh cho thấy đúng thay đổi, không phải nội dung từ cache hoặc dự án khác.
Trạng thái Git	Những file nào thực sự thuộc bản sửa?	Không có thư mục build, bí mật hoặc file tạm trong commit.
Hướng dẫn chạy	Người khác cần thực hiện những lệnh nào?	Một người học khác có thể chạy lại mà không hỏi thêm đường dẫn riêng trên máy.
Mốc hoàn thành	Trạng thái nào sẽ được dùng làm đầu vào chương sau?	Commit có tên rõ, build sạch và URL kiểm tra đã được ghi lại.

Đọc ứng dụng như một hệ thống có ranh giới. Khi người dùng mở trang chủ, trình duyệt chịu trách nhiệm gửi request và kết xuất HTML, server chịu trách nhiệm chọn endpoint và tạo response, còn hệ điều hành cùng .NET cung cấp tiến trình thực thi. Sau này cơ sở dữ liệu chịu trách nhiệm lưu trạng thái bền vững và dịch vụ AI chịu trách nhiệm sinh gợi ý trong giới hạn hợp đồng. Tách ranh giới như vậy giúp tránh hai suy luận sai: JavaScript không thể thay thế mọi kiểm tra ở server, và việc server trả HTTP 200 không tự chứng minh dữ liệu hay nội dung AI là đúng.

Một cách luyện tư duy ranh giới là theo dõi một thay đổi nhỏ qua các lớp. Yêu cầu "thêm liên kết Company vào thanh điều hướng" bắt đầu từ nhu cầu người dùng, đi vào layout, trỏ tới URL, được routing chuyển tới action, rồi action trả view. Nếu liên kết hiện nhưng bấm vào lỗi, phần HTML đã hoàn thành nhưng hợp đồng URL chưa hoàn thành. Nếu URL chạy khi gõ trực tiếp nhưng liên kết không hiện, server có thể đúng trong khi layout hoặc trình duyệt đang sai. Một triệu chứng giao diện vì thế không đủ để kết luận toàn hệ thống.

Chọn công nghệ bằng ràng buộc thay vì theo xu hướng. Trong một bài toán mới, cần ghi rõ loại giao diện, dữ liệu, môi trường triển khai, năng lực nhóm và yêu cầu bảo trì. ASP.NET Core MVC phù hợp với giáo trình vì cho phép quan sát rõ request, routing, controller, model và view trong cùng một dự án. Điều đó không có nghĩa mọi hệ thống đều phải dùng MVC. API độc lập phù hợp khi nhiều client dùng chung dữ liệu; ứng dụng kết xuất phía server phù hợp khi cần luồng HTML rõ và triển khai gọn; xử lý nền phù hợp với công việc không cần giữ request mở. Quyết định tốt nêu được điều kiện, không chỉ nêu tên framework.

Bảng 1.9. *Ma trận hỏi trước khi chọn hoặc đổi công nghệ*

Ràng buộc	Câu hỏi cần làm rõ	Hệ quả đối với thiết kế
Loại client	Chỉ trình duyệt hay còn ứng dụng di động và hệ thống khác?	Một client đơn giản có thể dùng MVC; nhiều client làm hợp đồng API quan trọng hơn.
Trạng thái dữ liệu	Dữ liệu tạm hay phải tồn tại sau khi tiến trình dừng?	Trạng thái bền vững cần cơ sở dữ liệu, migration và cơ chế sao lưu.
Độ trễ	Người dùng cần phản hồi ngay hay chấp nhận xử lý nền?	Công việc lâu cần hàng đợi hoặc trạng thái tiến độ, không giữ request vô hạn.
Bảo mật	Ai được xem và thay đổi dữ liệu?	Quyền phải được kiểm tra ở server, không chỉ ẩn nút giao diện.
Khả năng vận hành	Nhóm quan sát lỗi bằng cách nào?	Log, health check và chỉ số phải được tính từ đầu thay vì bổ sung sau sự cố.
Năng lực nhóm	Nhóm đã biết ngôn ngữ, công cụ và cách kiểm thử nào?	Công nghệ quen thuộc giảm rủi ro nhưng vẫn phải được đánh giá theo yêu cầu.
Tuổi thọ hệ thống	Bài demo ngắn hay sản phẩm cần bảo trì nhiều năm?	Tuổi thọ dài đòi hỏi ranh giới module, nâng cấp dependency và tài liệu quyết định.
Chi phí thay đổi	Nếu nhà cung cấp hoặc giao diện đổi thì tầng nào bị ảnh hưởng?	Hợp đồng và abstraction nên đặt tại ranh giới biến động thực sự.

Đường cơ sở về hiệu năng và an toàn. Ở Chương 1 chưa cần tối ưu truy vấn hay cấu hình triển khai phức tạp, nhưng có thể ghi lại thời gian khởi động, thời gian tải trang đầu tiên và kích thước response. Các số này chỉ là đường cơ sở để phát hiện thay đổi bất thường về sau, không phải cuộc thi lấy số nhỏ nhất. Tương tự, không đưa mật khẩu, API key hay chuỗi kết nối thật vào ảnh chụp và commit. Thói quen đo đúng thứ cần đo và che đúng dữ liệu cần che nên xuất hiện ngay từ bài đầu tiên.

Cuối cùng, một mốc chương tốt cần trả lời được ba câu hỏi. Thứ nhất, người khác có thể chạy lại dự án không? Thứ hai, mỗi thay đổi có bằng chứng ở đúng tầng không? Thứ ba, kho mã có đủ sạch để làm đầu vào cho chương kế tiếp không? Khi ba câu trả lời đều có bằng chứng, môi trường phát triển mới thực sự sẵn sàng cho việc học MVC.

Xác định phạm vi môi trường được hỗ trợ. Câu "chạy trên máy em" chưa mô tả được phạm vi hỗ trợ. Nhóm cần chốt hệ điều hành, kiến trúc CPU, phiên bản SDK, trình duyệt và tuyến cơ sở dữ liệu sẽ dùng. Không nhất thiết kiểm thử mọi tổ hợp, nhưng phải biết tổ hợp nào là chuẩn của lớp, tổ hợp nào chỉ là phương án thay thế và tổ hợp nào chưa được xác nhận. Khi người dùng báo lỗi trên môi trường ngoài phạm vi, nhóm vẫn ghi nhận nhưng không được suy diễn rằng lỗi thuộc mã nguồn trước khi tái hiện.

Bảng 1.10. *Ma trận phạm vi môi trường thực hành*

Thành phần	Tuyến chuẩn	Tuyến thay thế	Bằng chứng
SDK .NET	Phiên bản target của giáo trình.	SDK mới hơn nếu vẫn build đúng target.	`dotnet --info`, build.
Hệ điều hành	Windows theo phòng máy.	Linux/macOS với khác biệt được ghi rõ.	Log môi trường và đường dẫn.
Trình duyệt	Một trình duyệt Chromium hiện hành.	Firefox hoặc trình duyệt khác để đối chứng.	Network, Console, ảnh kết quả.

IDE/editor	VS Code cùng CLI.	Visual Studio hoặc editor khác.	Lệnh build không phụ thuộc IDE.
HTTPS cục bộ	Chứng thư phát triển được tin cậy.	HTTP cục bộ khi bài không yêu cầu cookie an toàn.	URL và cảnh báo trình duyệt.
Cơ sở dữ liệu	Tuyến được chốt cho Chương 6.	Container hoặc máy chủ dùng chung có kiểm soát.	Kết nối, phiên bản, dữ liệu thử.
Kho mã	Git với quy ước nhánh của lớp.	Bản nén chỉ cho tình huống đặc biệt.	Commit và trạng thái sạch.

Ma trận hỗ trợ giúp xử lý khác biệt mà không biến tài liệu thành danh sách mẹo cho mọi máy. Nếu một lệnh chỉ đúng trên PowerShell, cần cung cấp ý nghĩa của lệnh để người dùng shell khác có thể thực hiện tương đương. Nếu đường dẫn có khoảng trắng, lệnh mẫu phải được kiểm tra trong chính điều kiện đó. Tài liệu tốt mô tả mục tiêu và bằng chứng, không chỉ sao chép chuỗi ký tự từ một máy duy nhất.

Phân loại lỗi khởi động trước khi sửa. Các lỗi SDK, restore, build, bind cổng và request đầu tiên xảy ra ở những giai đoạn khác nhau. Nếu `dotnet restore` thất bại, chỉnh controller không có tác dụng. Nếu build thành công nhưng trình duyệt không kết nối, cần đọc log tiến trình và địa chỉ lắng nghe trước khi sửa route. Nếu server trả 404, tiến trình đã nhận request và phạm vi mới chuyển sang endpoint/routing.

Bảng 1.11. *Khoanh vùng sự cố ở lần chạy đầu*

Triệu chứng	Ranh giới nghi ngờ đầu tiên	Kiểm tra tối thiểu
Không nhận lệnh `dotnet`	Cài đặt SDK và biến môi trường.	`dotnet --info`; mở terminal mới sau cài đặt.
Restore lỗi	Nguồn package, mạng hoặc phiên bản.	Đọc package và nguồn lỗi; không xóa cache tùy tiện.
Build lỗi	Mã nguồn, target hoặc dependency.	Lỗi đầu tiên có file/dòng; build lại từ thư mục gốc.
Cổng đã dùng	Tiến trình khác hoặc cấu hình launch.	Log bind cổng và danh sách tiến trình liên quan.
Không kết nối	Server chưa chạy, URL/scheme sai.	Log “Now listening”, HTTP/HTTPS và cổng thật.
404	Request tới server nhưng không có endpoint khớp.	URL, method và danh sách route của ứng dụng.
500	Endpoint chạy nhưng phát sinh lỗi.	Log exception cùng mã tương quan, không đoán từ màn hình.
Giao diện cũ	Cache, sai dự án hoặc tiến trình chưa tải lại.	Network, đường dẫn file và PID tiến trình đang phục vụ.

Bàn giao mốc đầu tiên. Một người khác trong nhóm nên clone kho mã vào thư mục mới, chạy đúng chuỗi lệnh được ghi và xác nhận URL. Bài kiểm tra này phát hiện dependency ngầm như file cấu hình chưa commit, đường dẫn tuyệt đối hoặc package chỉ tồn tại trong cache cá nhân. Không cần xóa môi trường đang làm việc; một thư mục kiểm tra sạch đã đủ cho mục tiêu tái hiện và an toàn hơn thao tác phá hủy cấu hình hiện tại.

Hồ sơ bàn giao gồm README ngắn, commit đã kiểm chứng, các URL chính và giới hạn đã biết. Ảnh chụp chỉ minh họa kết quả; lệnh, log và test mới là bằng chứng có thể chạy lại. Khi mốc đầu vào rõ, các chương sau có thể tập trung vào kiến trúc và dữ liệu thay vì lặp lại việc đoán môi trường của từng máy.

Đọc log theo mốc thời gian và mã tương quan. Khi nhiều request xuất hiện gần nhau, một dòng lỗi riêng lẻ không đủ cho biết nó thuộc thao tác nào. Ngay từ ứng dụng đầu tiên, nên giữ thời điểm, mức log, nguồn phát và một mã tương quan cho chuỗi xử lý. Mã này không chứa dữ liệu cá nhân; nó chỉ nối request với các sự kiện liên quan. Khi báo lỗi, người học ghi URL, thời điểm và mã tương quan thay vì sao chép toàn bộ log không có phạm vi. Cách ghi này tạo nền cho việc quan sát các tầng phức tạp hơn ở những chương sau.

Bảng 1.12. *Phiếu xác nhận môi trường trước khi chuyển chương*

Kiểm tra	Bằng chứng đạt	Nếu chưa đạt
Khôi phục và build	Lệnh tại thư mục gốc kết thúc thành công.	Xử lý lỗi đầu tiên, không che cảnh báo bằng cách bỏ qua toàn bộ.
Khởi động	Log cho biết đúng môi trường và URL.	Kiểm tra cấu hình launch, cổng và tiến trình đang chạy.
Request cơ sở	URL chính trả đúng status và nội dung.	Đối chiếu scheme, route và log cùng thời điểm.
Kho mã	Commit không chứa output build hoặc bí mật.	Sửa quy tắc bỏ qua và thay bí mật đã lộ.
Bàn giao	Máy/thư mục sạch chạy theo README.	Ghi lại dependency ngầm rồi kiểm chứng lại từ đầu.

Phiếu xác nhận không thay thế việc hiểu công cụ. Nó tạo một điểm kết thúc có thể kiểm tra: môi trường, mã nguồn và hướng dẫn đều mô tả cùng một trạng thái. Khi trạng thái đó được gắn với commit cụ thể, mọi thay đổi về sau có một đường cơ sở để so sánh.

Tổng kết chương

Chương bắt đầu từ bức tranh chung của công nghệ phát triển ứng dụng, phân biệt ứng dụng web với ứng dụng di động, ứng dụng máy tính và dịch vụ phần mềm. Từ đó, trọng tâm được thu hẹp vào công nghệ web với client, server, request, response, HTTP, URL, HTML, CSS và JavaScript. Việc đối chiếu các hệ sinh thái giúp phân biệt ngôn ngữ, nền tảng, môi trường thực thi và framework trước khi giải thích lựa chọn ASP.NET Core MVC của học phần.

Trọng tâm của chương là hiểu ASP.NET Core MVC như một cách tổ chức ứng dụng web: Controller nhận request, Model biểu diễn dữ liệu, View sinh giao diện HTML, còn routing giúp request đi đến đúng action. Phần thực hành đưa các khái niệm đó vào một vòng lặp nhỏ nhưng đầy đủ: cài công cụ, tạo dự án MVC, chạy ứng dụng, sửa trang chủ, thêm action, tạo view và thêm liên kết điều hướng.

Phần cuối đặt tác tử AI vào một quy trình có kiểm soát: người học xác định yêu cầu, giới hạn phạm vi, xem thay đổi và kiểm chứng bằng build. Sau bước rà soát bằng AI, xUnit cung cấp bằng chứng thực thi lặp lại. Cả hai công cụ đảm nhiệm các vai trò khác nhau và đều dựa trên hiểu biết kỹ thuật của người học.

Câu hỏi ôn tập

1. Phân biệt ứng dụng web, ứng dụng di động, ứng dụng máy tính và dịch vụ phần mềm theo cách người dùng tiếp cận và cách phần mềm được cung cấp.
2. Vì sao chương trình console không đủ để mô tả cách hoạt động của một ứng dụng web nhiều người dùng?
3. Client và server có vai trò khác nhau như thế nào trong mô hình ứng dụng web?

4. Request và response là gì? Hãy mô tả luồng xử lý khi người dùng truy cập một URL trên trình duyệt.
5. HTTP là viết tắt của cụm từ nào, và vì sao web cần một giao thức chung như HTTP?
6. Phân biệt `GET` và `POST`. Mỗi phương thức thường phù hợp với loại thao tác nào?
7. HTML và CSS có vai trò gì trong response mà server trả về cho trình duyệt?
8. Vì sao các bạn đã học Java có thể chuyển sang C# tương đối thuận lợi? Nêu ba điểm giống hoặc khác quan trọng giữa Java và C#.
9. Trong ASP.NET Core MVC, Controller, Model và View lần lượt giải quyết vấn đề gì?
10. Khi thêm một trang `About` vào dự án MVC đầu tiên, cần sửa hoặc tạo những file nào? Vì sao các tên controller, action và view phải khớp với nhau?
11. Vì sao không nên giao toàn bộ việc tạo và sửa dự án cho tác tử AI bằng một prompt chung chung? Hãy liên hệ với phạm vi thay đổi, khả năng kiểm chứng và trách nhiệm của người lập trình.
12. Kiểm thử thủ công và kiểm thử tự động khác nhau ở khả năng lặp lại và cách tạo bằng chứng như thế nào?
13. Vì sao test `TestRunner_WhenExecuted_RunsXunitTest` mới chỉ xác nhận hạ tầng xUnit, chưa phải unit test nghiệp vụ?
14. So sánh ASP.NET Core với một lựa chọn web khác theo bốn tiêu chí: ngôn ngữ, loại nền tảng, cách tổ chức ứng dụng và công cụ truy cập dữ liệu.

Bài thực hành Chương 1

Mục đích

Tạo được solution ASP.NET Core MVC đầu tiên, quan sát luồng request–response, thực hiện một thay đổi giao diện có kiểm chứng và phân biệt test hạ tầng với test hành vi.

Yêu cầu chung

Máy có .NET 10 SDK, VS Code và C# Dev Kit; sản phẩm không dùng cơ sở dữ liệu hoặc package ngoài template. Mỗi bài phải lưu lệnh đã chạy và kết quả build/test. Sinh viên nộp mã nguồn hoặc liên kết commit/tag, tệp ghi lệnh build/test và phần giải thích request. Bài đạt khi solution build được, các test hiện có đều qua, ba URL `/`, `/Home/About`, `/Home/Services` trả đúng trang, navigation hoạt động và báo cáo phân biệt rõ kiểm tra giao diện với kiểm thử tự động.

Bài 1.1 – Bài giải mẫu có hướng dẫn

Tạo solution `AppDemo`, dự án MVC và dự án xUnit theo Mục 1.6. Sửa trang chủ, thêm action/view `About` và liên kết `Giới thiệu`. Thực hiện theo bốn mốc kiểm tra: (1) `dotnet --version` trả về phiên bản 10; (2) `dotnet build AppDemo.slnx` thành công; (3) `dotnet test AppDemo.slnx` chạy được; (4) truy cập `/Home/About` và kiểm tra kết quả. Nếu một mốc thất bại, dừng tại đó và ghi thông báo lỗi trước khi sửa.

Bài 1.2 – Bài tập cơ bản

Thêm action và view `Services` vào `HomeController`; giao diện liệt kê ba dịch vụ của đơn vị tổ chức sự kiện. Thêm liên kết vào layout, chạy lại ứng dụng và giải thích bằng một đoạn ngắn đường đi của request `/Home/Services`. Không dùng `ViewBag`, model

hoặc cơ sở dữ liệu ở bài này.

Bài 1.3 – Bài tập nâng cao

Dùng tác tử AI để đề xuất thay đổi tiêu đề và bố cục trang `Services`, nhưng giới hạn phạm vi ở view và CSS. Trước khi chấp nhận, đọc diff, loại mọi package hoặc chức năng ngoài yêu cầu, rồi chạy build/test. Dùng Developer Tools xác định request tài liệu HTML của trang và ghi lại method, URL, status code, content type; giải thích vì sao dữ liệu quan sát được chưa chứng minh toàn bộ website an toàn.

Mã nguồn thực hành của chương

Mã nguồn hoàn chỉnh của Chương 1 được cung cấp tại: `https://github.com/NguyenPhuong86/CongNghePhatTrienUD/tree/chapter-01-completed` [31].