Model và ViewModel

* Domain model: map dữ liệu tương ứng với db, dữ liệu thật
* View model: model chứa dữ liệu dành cho hiển thị, ví dụ field bên domain model chứa ProductId hoặc cả đối tượng Product, còn view model không hiển thị id nên không cần field ProductId, mà chỉ cần field ProductName, một số field có thể được thêm hoặc bớt so với domain model để phù hợp cho việc hiển thị, có thể có method trong view model
* Edit model: model dành cho chỉnh sửa, thêm một số field cho việc chỉnh sửa, ví dụ: trên view có dropdownlist, thì trong edit model cần thêm field list<product> và productId, và thông tin về validation
* Đôi khi view cần nhiều dữ liệu từ các domain model, việc view model và domain model độc lập cho phép dễ dàng giải quyết điều này
* Best practice

+ Sử dụng strongly typed view

+ Dùng data annotation để validation

+ Chỉ đưa dữ liệu cần thiết vào view model

+ Dùng mapper để chuyển dữ liệu từ model sang view model

+ view model có thể chứa những cách hành xử cụ thể(phương thức) với view

+ sử dụng một view model trong mỗi view

Strongly typed view

* Bind với một kiểu cố định

Xây dựng form Html

* Sử dụng strongly typed view để generate form tự động, dùng @model trong view

Tag Helper

* Thân thiện với phát triển html
* Hỗ trợ gợi ý code
* Code sạch hơn
* Có thể mở rộng

Input Tag Helper

* Trong tag input nếu không có để thuộc tính type thì sẽ generate theo kiểu trong model, ví dụ như nếu là trong model là kiểu string, decimal, double, float thì sẽ gen ra kiểu text, kiểu int, byte, double thì sẽ gen ra kiểu number, kiểu datetime thì sẽ gen ra kiểu datetime-local, kiểu bool thì sẽ gen ra kiểu checkbox…
* Thêm dấu hỏi vào kiểu như decimal? Rate thì sẽ gen ra kiểu không required

Environment Tag Helper

* Xác định qua biến ASPNETCORE\_ENVIRONMENT có thể có các giá trị: development, test, production…

Model Binding

* Là cơ chế truyền dữ liệu từ view đến controller
* ModelState.IsValid: kiểm tra quá trình bind có hợp lệ
* Nơi lấy dữ liệu để binding: giá trị từ html form, route, query strings, body, header, services
* [FromForm], [FromRoute], [FromQuery], [FromBody], [FromHeader], [FromServices], [BindNever], [BindRequired]

Model Validation

* Model validator: kiểm tra tính hợp lệ dữ liệu
* Có client-side và server-side validation
* Client-side validation: tốt hơn cho trải nghiệm người dùng, validate ngay trên trình duyệt nên xử lý nhanh
* Server-side validation: đôi khi javascript bị disabled trên trình duyệt, lỗi không validate được dưới client sẽ do server đảm trách, nhưng sẽ mất thời gian hơn
* Danh sách thuộc tính validation: CreditCard, Compare, EmailAddress, Phone, Range, RegularExpression, Required, StringLength, Url
* <div asp-validation-summary=”ModelOnly”>

Validation Tag Helper

* Dùng để hiển thị validation message cho người dùng
* Có validation message tag helper, và validation summary tag helper
* Summary tag helper có 3 mode: all, ModelOnly, none

+all: tất cả các loại message, bao gồm cả lỗi liên quan thuộc tính

Unobtrusive Client Side validation

* Là cơ chế validation đặt ra trên server, nhưng hỗ trợ cho client
* Viết code một chỗ mà validate cả server và client side
* Dùng các thư viện js như jquery và js unobtrusive, html5 built-in validation
* Jquery, jquery validate, jquery validate unobtrusive

Dependency Injection

* Giúp dễ dàng maintain và thực hiện unit test, linh hoạt hơn trong tổ chức code
* DI Container là đối tượng chịu trách nhiệm tạo re dependencies
* DI Container sẽ khởi tạo và quản lý các service mà ta sẽ dùng trong controller. Ví dụ: HomeController sẽ yêu cầu danh sách product, sẽ có Object ProductService thực hiện, DI Container sẽ gửi cho HomeController một instance của ProductService, HomeController sẽ sử dụng instance này để lấy ra danh sách product, HomeController không quan tâm đến việc tạo mới mà chỉ cần yêu cầu DI Container, DI Container sẽ tự inject thể hiện của đối tượng này vào
* 2 kiểu injection

+Construction Injection: inject lúc khởi tạo

+Action Injection: inject vào method

* Làm giảm sự phụ thuộc(?), code sạch và dễ đọc hơn, test và maintain dễ hơn

Attribute Based Routing

* Cấu hình route trực tiếp trên method

Route constraints

* Thiết lập ràng buộc trên route
* Validation input, giúp routing engine phân biệt được 2 pattern giống nhau, ví dụ

+post/{id:int}

+post/{id:alpha}

Action Selector và Action verbs

* Action selector types: gồm action name, non action, action verbs

Action Results

* Khi controller không được kế thừa từ controllerbase, cần inject IhttpContextAccessor, cần HttpContext để giữ giá trị IhttpContextAccessor.HttpContext, trong hàm index cần viết thủ công để lấy thông tin từ response trả về, phải append thêm chuỗi html
* Khi trả về nhiều loại result thì action phải trả về IactionResult hoặc ActionResult
* Các loại Action result: trả về html, redirect người dùng, trả về file, content result, errors và http codes, resutls liên quan đến security

+html: ViewResult và PartialViewResult

+RedirectUser: chuyển hướng người dùng

+trả về file : trả về nội dung trong file hoặc mảng byte

+contentResult: trả về chuỗi json, đoạn text hoặc giá trị rỗng

+httpcode: ok, 200…, hoặc object với message=hello

+SignInResult, SignOutResult, ChallengeResult: chứng thực thất bại, UnauthorizedResutl

View

Razor view engine

* Cú pháp razor: như C#, trộn với html

Layout và section

* Layout, section, renderbody, viewstart
* Section: cho phép nhúng view vào layout

Cơ chế application startup

* Program.cs : khởi tạo webserver, hosting, entry point, startup.cs : config các service DI, cấu hình request pipeline, middleware
* Các ứng dụng .net core đều là console
* Program.cs:

+ CreateWebHostBuilder: tạo web host, với tham số args

+ .UseStartup : sử dụng cấu hình trong file startup

+ WebHostBuilder()….ConfigureAppConfiguration(hostingContext, config) : để truyền tham số cho Startup.Configure

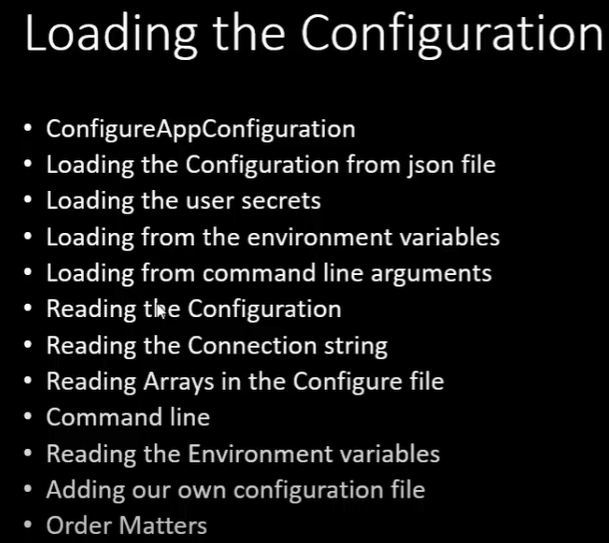
* Startup.cs: gồm ConfigureServices và Configure

+ ConfigureServices: nhận vào tập hợp các service để thiết lập

+ Configure: nhận vào IapplicationBuilder để cấu hình request pipeline, IhostingEnvironment để kiểm tra môi trường: development hay production

+ trong Configure là các middleware

File cấu hình

* Nội dung thường để trong file appsetting.json
* Có thể để chuỗi kết nối db
* Lưu theo dạng cặp name-value, có thể lưu dưới dạng json, xml, ini
* 
* Có thể giấu chuỗi kết nối db trong file secrets.json

Static files

* Là cách sử dụng thư mục wwwroot
* Gồm các loại : html, css, image, javascript
* Có thể tải trực tiếp file mà không cần thông qua middleware, request pepline
* Content root và webroot
* Khi đặt file trong wwwroot thì nghĩa là đã public file
* Webroot : thư mục gốc của ứng dụng web, là thư mục chứa file tĩnh(wwwroot)
* Có thể thiết lập lại thư mục webroot trong program.cs, method CreateWebHostBuilder: .UseWebRoot(“resource”)
* Đây là content root:

