Tuyệt vời! Dưới đây là toàn bộ nội dung mà bạn yêu cầu, được định dạng bằng Markdown để bạn có thể dễ dàng sao chép và dán vào file Word của mình.

**Tổng Hợp Toàn Bộ Bộ Trắc Nghiệm Ôn Tập ASP.NET Core MVC & EF Core**

**Phần 1: HTML Cơ Bản**

Mức độ: Dễ

Câu hỏi: HTML là viết tắt của từ gì?

A. HyperText Markup Language

B. Hyperlinks and Text Markup Language

C. Home Tool Markup Language

D. HyperText Main Language

Đáp án đúng: A

Giải thích: HTML là ngôn ngữ đánh dấu siêu văn bản, được sử dụng để tạo cấu trúc và nội dung cho các trang web.

Câu hỏi: Thẻ HTML nào được sử dụng để tạo một đoạn văn bản (paragraph)?

A. <p>

B. <d>

C. <para>

D. <text>

Đáp án đúng: A

Giải thích: Thẻ <p> được sử dụng để định nghĩa một đoạn văn bản.

Câu hỏi: Thẻ HTML nào được sử dụng để tạo một tiêu đề cấp 1 (largest heading)?

A. <h1>

B. <head>

C. <heading>

D. <h6>

Đáp án đúng: A

Giải thích: Thẻ <h1> đến <h6> được sử dụng để định nghĩa các tiêu đề, với <h1> là lớn nhất và <h6> là nhỏ nhất.

Câu hỏi: Để tạo một danh sách không có thứ tự (unordered list), bạn sử dụng thẻ nào?

A. <ol>

B. <ul>

C. <li>

D. <dl>

Đáp án đúng: B

Giải thích: Thẻ <ul> định nghĩa một danh sách không có thứ tự, và các mục trong danh sách được định nghĩa bằng thẻ <li>.

Câu hỏi: Thuộc tính HTML nào được sử dụng để chỉ định URL của một liên kết (hyperlink)?

A. src

B. href

C. link

D. url

Đáp án đúng: B

Giải thích: Thuộc tính href trong thẻ <a> (anchor) chỉ định địa chỉ đích của liên kết.

**Phần 2: Giới thiệu ASP.NET & ASP.NET Core**

Mức độ: Dễ

Câu hỏi: ASP.NET Web Forms có những điểm nổi bật nào?

A. Chỉ hỗ trợ VBScript và JavaScript.

B. Biên dịch các trang trước, giúp tăng tốc độ thực hiện.

C. Không quản lý trạng thái trang web.

D. Cơ chế debug kém.

Đáp án đúng: B

Giải thích: ASP.NET Web Forms hỗ trợ đa ngôn ngữ (C#, VB.NET), biên dịch trang trước, quản lý trạng thái trang web, và có cơ chế debug tốt hơn ASP cổ điển.

Câu hỏi: Mô hình kiến trúc nào được ASP.NET MVC sử dụng?

A. Client-Server

B. 3-Tier Architecture

C. Model-View-Controller

D. Peer-to-Peer

Đáp án đúng: C

Giải thích: ASP.NET MVC dựa trên mô hình Model-View-Controller để tách biệt các mối quan tâm.

Câu hỏi: Lợi ích chính của việc sử dụng mô hình MVC trong phát triển ứng dụng web là gì?

A. Tăng băng thông truyền dữ liệu giữa client và server.

B. Khó thực hiện unit test.

C. Có tính mở rộng do có thể thay thế từng thành phần một cách dễ dàng.

D. HTML bị giới hạn và khó sử dụng JavaScript.

Đáp án đúng: C

Giải thích: MVC giúp tách biệt các thành phần, làm cho ứng dụng dễ mở rộng, dễ bảo trì và dễ kiểm thử hơn.

Câu hỏi: ASP.NET Core là một framework như thế nào?

A. Chỉ chạy trên Windows và là mã nguồn đóng.

B. Cross-platform, hiệu suất cao, mã nguồn mở.

C. Chỉ dùng để xây dựng ứng dụng desktop.

D. Chỉ hỗ trợ ngôn ngữ C#.

Đáp án đúng: B

Giải thích: ASP.NET Core là một framework hiện đại, cross-platform, hiệu suất cao và mã nguồn mở.

Câu hỏi: So với ASP.NET 4.x, ASP.NET Core có những lợi ích nào?

A. Chỉ xây dựng cho Windows.

B. Hiệu suất thấp hơn.

C. Hỗ trợ xây dựng trên Windows, macOS, hoặc Linux.

D. Chỉ có một phiên bản trên mỗi máy.

Đáp án đúng: C

Giải thích: ASP.NET Core là cross-platform, có hiệu suất cao hơn và hỗ trợ nhiều phiên bản trên cùng một máy.

**Phần 3: Fundamentals**

Mức độ: Dễ

Câu hỏi: Trong ASP.NET Core, Middleware là gì?

A. Một lớp xử lý logic nghiệp vụ chính của ứng dụng.

B. Một thành phần xử lý các yêu cầu HTTP trong pipeline.

C. Một cơ sở dữ liệu tạm thời để lưu trữ dữ liệu.

D. Một công cụ để tạo giao diện người dùng.

Đáp án đúng: B

Giải thích: Middleware là các thành phần được sắp xếp trong một pipeline để xử lý các yêu cầu và phản hồi HTTP.

Câu hỏi: Phương thức UseStaticFiles() trong Program.cs (hoặc Startup.cs trước đây) có mục đích gì?

A. Cấu hình định tuyến cho các Controller.

B. Cho phép ứng dụng phục vụ các tệp tĩnh (như CSS, JavaScript, hình ảnh) từ thư mục wwwroot.

C. Thiết lập kết nối cơ sở dữ liệu.

D. Bật tính năng xác thực người dùng.

Đáp án đúng: B

Giải thích: UseStaticFiles() là middleware cho phép phục vụ các tệp tĩnh.

Câu hỏi: Model Binding trong ASP.NET Core MVC có chức năng gì?

A. Xử lý các yêu cầu HTTP đến Controller.

B. Chuyển đổi dữ liệu từ yêu cầu HTTP (form fields, query strings, route data) thành các đối tượng .NET.

C. Hiển thị giao diện người dùng.

D. Quản lý phiên làm việc của người dùng.

Đáp án đúng: B

Giải thích: Model Binding tự động ánh xạ dữ liệu từ request đến các tham số của Action Method.

Câu hỏi: Dependency Injection (DI) trong ASP.NET Core có lợi ích gì?

A. Giảm khả năng kiểm thử của ứng dụng.

B. Tăng sự phụ thuộc giữa các thành phần.

C. Giúp quản lý các phụ thuộc giữa các thành phần một cách linh hoạt và dễ kiểm thử hơn.

D. Chỉ được sử dụng cho các ứng dụng Web API.

Đáp án đúng: C

Giải thích: DI là một mẫu thiết kế giúp giảm sự phụ thuộc chặt chẽ giữa các thành phần, làm cho code dễ bảo trì và kiểm thử hơn.

Câu hỏi: Razor View Engine trong ASP.NET Core MVC được sử dụng để làm gì?

A. Xử lý các yêu cầu cơ sở dữ liệu.

B. Tạo ra các tệp JavaScript động.

C. Render các View bằng cách kết hợp mã C# với HTML.

D. Quản lý các phiên làm việc của người dùng.

Đáp án đúng: C

Giải thích: Razor là một cú pháp đánh dấu cho phép nhúng mã C# vào HTML để tạo ra nội dung web động.

**Phần 4: Views & Tag Helpers**

Mức độ: Dễ

Câu hỏi: Trong ASP.NET Core MVC, các tệp View thường có phần mở rộng là gì?

A. .html

B. .cs

C. .cshtml

D. .razor

Đáp án đúng: C

Giải thích: Các View trong ASP.NET Core MVC sử dụng cú pháp Razor và có phần mở rộng là .cshtml.

Câu hỏi: Để truyền dữ liệu từ Controller sang View một cách mạnh mẽ (strongly typed), phương pháp nào được khuyến nghị?

A. ViewData

B. ViewBag

C. ViewModel (Strongly typed data)

D. TempData

Đáp án đúng: C

Giải thích: Sử dụng ViewModel giúp bạn có kiểu dữ liệu rõ ràng, hỗ trợ IntelliSense và kiểm tra lỗi tại thời điểm biên dịch.

Câu hỏi: Tag Helper nào được sử dụng để tạo một thẻ <a> (anchor) và tự động tạo URL dựa trên Controller và Action?

A. Form Tag Helper

B. Input Tag Helper

C. Anchor Tag Helper

D. Label Tag Helper

Đáp án đúng: C

Giải thích: Anchor Tag Helper (<a asp-controller="..." asp-action="...">) giúp tạo các liên kết động.

Câu hỏi: Partial View thường được đặt tên bắt đầu bằng ký tự nào theo quy ước?

A. #

B. $

C. \_

D. @

Đáp án đúng: C

Giải thích: Quy ước đặt tên Partial View bắt đầu bằng dấu gạch dưới (\_) giúp phân biệt chúng với các View thông thường.

Câu hỏi: Tag Helper nào được sử dụng để hiển thị một thông báo lỗi validation cho một thuộc tính cụ thể của Model?

A. Validation Summary Tag Helper

B. Input Tag Helper

C. Validation Message Tag Helper

D. Form Tag Helper

Đáp án đúng: C

Giải thích: Validation Message Tag Helper (<span asp-validation-for="...">) hiển thị lỗi cho từng trường nhập liệu.

Mức độ: Trung bình

Câu hỏi: Phân biệt sự khác nhau cơ bản giữa ViewData và ViewBag khi truyền dữ liệu từ Controller sang View.

A. ViewData yêu cầu ép kiểu, ViewBag không yêu cầu ép kiểu.

B. ViewData không có thuộc tính dictionary, ViewBag có.

C. ViewData chỉ dùng cho chuỗi, ViewBag dùng cho mọi kiểu dữ liệu.

D. ViewData có sẵn trong Razor Pages, ViewBag không.

Đáp án đúng: A

Giải thích: ViewData là một ViewDataDictionary và truy cập thông qua string key, yêu cầu ép kiểu khi lấy dữ liệu không phải chuỗi. ViewBag là một DynamicViewData và truy cập thông qua dot notation, không yêu cầu ép kiểu. Cả hai đều có thể chứa mọi kiểu dữ liệu. ViewBag không có sẵn trong Razor Pages.

Câu hỏi: Khi nào bạn nên sử dụng Partial View thay vì View Component?

A. Khi bạn cần logic nghiệp vụ phức tạp và tham số.

B. Khi bạn muốn tái sử dụng một phần giao diện đơn giản, không có logic phức tạp hoặc phụ thuộc vào dữ liệu từ Model của View cha.

C. Khi bạn muốn hiển thị một phần giao diện động dựa trên truy vấn cơ sở dữ liệu.

D. Khi bạn muốn tạo một widget độc lập.

Đáp án đúng: B

Giải thích: Partial View phù hợp cho việc tái sử dụng markup đơn giản, trong khi View Component mạnh mẽ hơn với logic nghiệp vụ và khả năng có tham số riêng.

Câu hỏi: Tag Helper nào sẽ tự động tạo thuộc tính type của thẻ <input> dựa trên kiểu dữ liệu của Model và các Data Annotation Attributes?

A. Label Tag Helper

B. Form Tag Helper

C. Input Tag Helper

D. Select Tag Helper

Đáp án đúng: C

Giải thích: Input Tag Helper (<input asp-for="...">) tự động xác định type (ví dụ: text, number, email, password, date) dựa trên kiểu dữ liệu và các thuộc tính như [EmailAddress], [DataType(DataType.Password)].

Câu hỏi: Điều gì sẽ xảy ra nếu bạn sử dụng Html.PartialAsync để gọi một Partial View và truyền một Model cho nó?

A. Partial View sẽ nhận được Model đó và không thể truy cập Model của View cha.

B. Partial View sẽ nhận được Model đó và vẫn có thể truy cập Model của View cha.

C. Partial View sẽ không nhận được Model đó và chỉ có thể truy cập Model của View cha.

D. Sẽ xảy ra lỗi biên dịch.

Đáp án đúng: A

Giải thích: Khi bạn truyền một Model cụ thể cho Partial View bằng Html.PartialAsync hoặc <partial model="...">, Partial View đó sẽ sử dụng Model được truyền vào và không kế thừa Model của View cha.

Câu hỏi: Khi sử dụng Anchor Tag Helper, thuộc tính asp-all-route-data được dùng để làm gì?

A. Để chỉ định một giá trị route duy nhất.

B. Để chỉ định tất cả các giá trị route từ một Dictionary.

C. Để chỉ định tên của Controller.

D. Để chỉ định tên của Action.

Đáp án đúng: B

Giải thích: asp-all-route-data cho phép bạn truyền một Dictionary<string, string> chứa nhiều cặp key-value làm route data.

Mức độ: Khó

Câu hỏi: Giải thích sự khác biệt giữa Html.PartialAsync và <partial> Tag Helper khi gọi Partial View. Khi nào bạn nên ưu tiên sử dụng <partial>?

Đáp án:

Html.PartialAsync (HTML Helper):

* Là một phương thức C# được gọi trong Razor View.
* Cú pháp: @await Html.PartialAsync("\_PartialName", model)
* Đã tồn tại từ các phiên bản ASP.NET MVC trước.

**&lt;partial> Tag Helper**:

* Là một Tag Helper, có cú pháp giống HTML hơn.
* Cú pháp: <partial name="\_PartialName" model="model" /> hoặc <partial name="\_PartialName" for="Model.Property" />
* Được giới thiệu trong ASP.NET Core.

**Sự khác biệt chính**:

* **Cú pháp**: Html.PartialAsync là C# code, trong khi <partial> là cú pháp giống HTML.
* **Khả năng đọc**: <partial> thường dễ đọc và dễ hiểu hơn trong các tệp Razor vì nó hòa nhập tốt hơn với HTML.
* **IntelliSense**: Cả hai đều có hỗ trợ IntelliSense tốt.
* **Tính năng**: Về cơ bản, chúng cung cấp chức năng tương tự.

**Khi nào nên ưu tiên sử dụng <partial>**:

* **Tính nhất quán**: Khi bạn đã sử dụng các Tag Helper khác trong dự án của mình, việc sử dụng <partial> sẽ duy trì tính nhất quán về cú pháp.
* **Khả năng đọc**: Đối với các nhà phát triển quen thuộc với HTML hơn là C# trong markup, cú pháp của Tag Helper có thể dễ đọc và dễ bảo trì hơn.
* **Tách biệt mối quan tâm**: Tag Helpers giúp giảm sự pha trộn giữa C# và HTML trong View, làm cho View tập trung hơn vào việc trình bày.

**Giải thích**: <partial> Tag Helper là cách tiếp cận hiện đại và được khuyến nghị hơn trong ASP.NET Core để gọi Partial View do cú pháp rõ ràng và tích hợp tốt hơn với HTML.

Câu hỏi: View Components được thiết kế để giải quyết vấn đề gì mà Partial Views không thể giải quyết một cách hiệu quả? Nêu một ví dụ cụ thể về trường hợp sử dụng View Component.

Đáp án:

Vấn đề View Components giải quyết:

Partial Views chỉ đơn thuần là các đoạn markup có thể tái sử dụng. Chúng không có logic nghiệp vụ riêng, không có khả năng xử lý tham số phức tạp, và không tham gia vào Dependency Injection một cách trực tiếp.

View Components được thiết kế để xử lý các "widget" hoặc các phần giao diện phức tạp, có logic nghiệp vụ riêng, có thể truy vấn dữ liệu, và có thể được kiểm thử độc lập. Chúng giống như một "mini-MVC" trong View.

**Ví dụ cụ thể**:

* **Giỏ hàng (Shopping Cart Summary)**: Một widget giỏ hàng nhỏ hiển thị số lượng mặt hàng và tổng tiền trên mọi trang của một trang web thương mại điện tử.
  + **Tại sao là View Component**: Widget này cần truy vấn cơ sở dữ liệu (hoặc session) để lấy thông tin giỏ hàng, có thể có logic để tính tổng tiền, và cần được cập nhật độc lập với Model của trang chính. Nó có thể được gọi từ Layout View.
* **Menu điều hướng động**: Một menu điều hướng mà các mục của nó được tạo ra dựa trên quyền của người dùng hoặc dữ liệu từ cơ sở dữ liệu.
* **Login Panel**: Một panel đăng nhập/đăng xuất hiển thị trạng thái người dùng và các liên kết tương ứng.

**Giải thích**: View Components cung cấp sự tách biệt mối quan tâm tốt hơn và khả năng kiểm thử cao hơn cho các phần giao diện phức tạp, có logic riêng.

Câu hỏi: Bạn có một form đăng ký với các trường Email và Password. Bạn muốn sử dụng Input Tag Helper và Validation Message Tag Helper để hiển thị các trường này và thông báo lỗi. Hãy viết đoạn mã Razor cho các trường này, giả sử bạn có một RegisterViewModel với các thuộc tính Email và Password được đánh dấu [Required] và [EmailAddress] cho Email, [DataType(DataType.Password)] cho Password.

Đáp án:

C#

@model YourNamespace.ViewModels.RegisterViewModel

<form asp-controller="Account" asp-action="Register" method="post">

<div class="form-group">

<label asp-for="Email"></label>

<input asp-for="Email" class="form-control" />

<span asp-validation-for="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Password"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<button type="submit" class="btn btn-primary">Register</button>

</form>

**Giải thích**:

* @model YourNamespace.ViewModels.RegisterViewModel: Khai báo View này là strongly typed với RegisterViewModel.
* <label asp-for="Email"></label>: Label Tag Helper sẽ tự động tạo <label for="Email">Email Address</label> (nếu Display(Name="Email Address") được sử dụng trong ViewModel) hoặc <label for="Email">Email</label>.
* <input asp-for="Email" class="form-control" />: Input Tag Helper sẽ tạo <input type="email" id="Email" name="Email" class="form-control" data-val="true" data-val-email="..." data-val-required="..." />. Nó tự động thêm type="email" do [EmailAddress] và các thuộc tính validation HTML5.
* <span asp-validation-for="Email" class="text-danger"></span>: Validation Message Tag Helper sẽ hiển thị thông báo lỗi validation cho trường Email nếu có.
* Tương tự cho trường Password, Input Tag Helper sẽ tạo type="password" do [DataType(DataType.Password)]. **Lưu ý**: Để client-side validation hoạt động, bạn cần đảm bảo các script validation (như jQuery Validation) được bao gồm trong trang của bạn.

**Phần 5: Session & State Management**

Mức độ: Dễ

Câu hỏi: Phương pháp quản lý trạng thái nào lưu trữ dữ liệu trên client và có kích thước giới hạn (thường là 4096 bytes)?

A. Session State

B. TempData

C. Cookies

D. HttpContext.Items

Đáp án đúng: C

Giải thích: Cookies là các tệp nhỏ được lưu trữ trên trình duyệt của người dùng.

Câu hỏi: Session State trong ASP.NET Core duy trì trạng thái bằng cách cung cấp một cái gì cho client?

A. Một chuỗi kết nối cơ sở dữ liệu.

B. Một file cấu hình.

C. Một cookie chứa Session ID.

D. Một đối tượng JavaScript.

Đáp án đúng: C

Giải thích: Session ID trong cookie được gửi đến ứng dụng với mỗi yêu cầu để ứng dụng có thể tìm nạp dữ liệu phiên tương ứng.

Câu hỏi: Theo mặc định, thời gian chờ (timeout) của Session State trong ASP.NET Core là bao nhiêu phút?

A. 10 phút

B. 15 phút

C. 20 phút

D. 30 phút

Đáp án đúng: C

Giải thích: Thời gian chờ mặc định của Session State là 20 phút.

Câu hỏi: TempData được sử dụng chủ yếu cho mục đích gì?

A. Lưu trữ dữ liệu vĩnh viễn trên server.

B. Lưu trữ dữ liệu chỉ tồn tại trong một yêu cầu duy nhất.

C. Lưu trữ dữ liệu cần thiết cho việc chuyển hướng (redirection) và tồn tại trong tối đa hai yêu cầu (yêu cầu hiện tại và yêu cầu tiếp theo sau redirect).

D. Lưu trữ dữ liệu nhạy cảm của người dùng.

Đáp án đúng: C

Giải thích: TempData rất hữu ích cho các thông báo (ví dụ: "Đã lưu thành công") sau khi chuyển hướng.

Câu hỏi: Để kích hoạt Session State trong ứng dụng ASP.NET Core, bạn cần gọi phương thức nào trong ConfigureServices?

A. UseSession()

B. AddSession()

C. EnableSession()

D. MapSession()

Đáp án đúng: B

Giải thích: AddSession() được gọi trong ConfigureServices để thêm các dịch vụ Session vào DI container.

Mức độ: Trung bình

Câu hỏi: Điều gì xảy ra với dữ liệu Session State nếu người dùng đóng trình duyệt mà không đăng xuất?

A. Dữ liệu Session sẽ được lưu trữ vĩnh viễn trên server.

B. Cookie Session sẽ bị xóa khi phiên trình duyệt kết thúc, và dữ liệu Session trên server sẽ bị xóa sau khi hết thời gian chờ.

C. Dữ liệu Session sẽ tự động được chuyển sang Cookies.

D. Ứng dụng sẽ tự động tạo một Session mới cho người dùng đó khi họ mở lại trình duyệt.

Đáp án đúng: B

Giải thích: Cookie phiên bị xóa khi trình duyệt đóng, nhưng dữ liệu trên server vẫn tồn tại cho đến khi hết thời gian chờ của phiên.

Câu hỏi: Để dữ liệu trong TempData tồn tại thêm một yêu cầu nữa sau khi đã được đọc, bạn cần sử dụng phương thức nào?

A. TempData.Clear()

B. TempData.Remove("Key")

C. TempData.Keep("Key") hoặc TempData.Keep()

D. TempData.Peek("Key")

Đáp án đúng: C

Giải thích: Keep() đánh dấu các khóa trong TempData để giữ lại cho yêu cầu tiếp theo.

Câu hỏi: Khi cấu hình Session State, thuộc tính IsEssential của options.Cookie có ý nghĩa gì nếu được đặt là false (mặc định)?

A. Cookie Session là bắt buộc và luôn được gửi.

B. Cookie Session không được đánh dấu là thiết yếu, Session State sẽ không hoạt động trừ khi người dùng cho phép theo dõi.

C. Cookie Session sẽ bị xóa ngay lập tức sau khi được tạo.

D. Cookie Session chỉ hoạt động trên các kết nối HTTPS.

Đáp án đúng: B

Giải thích: Nếu IsEssential là false, cookie phiên không được coi là thiết yếu theo quy định GDPR, và Session State sẽ không hoạt động nếu người dùng không đồng ý với việc theo dõi.

Câu hỏi: Phương pháp nào sau đây KHÔNG phải là một cách để thiết lập dịch vụ Distributed Caching cho Session State trong ASP.NET Core?

A. Distributed Memory Cache

B. Distributed SQL Server Cache

C. Distributed Redis Cache

D. Distributed File System Cache

Đáp án đúng: D

Giải thích: Các tùy chọn Distributed Caching được cung cấp bao gồm Memory, SQL Server, Redis và NCache. File System Cache không phải là một tùy chọn mặc định.

Câu hỏi: Điều gì sẽ xảy ra nếu một Session không có ít nhất một giá trị được đặt để duy trì phiên trên các yêu cầu?

A. Session sẽ được lưu trữ vĩnh viễn.

B. Session sẽ được giữ lại nhưng Session ID sẽ thay đổi.

C. Session trống sẽ không được giữ lại, và một Session ID mới sẽ được tạo cho mỗi yêu cầu mới.

D. Ứng dụng sẽ gặp lỗi.

Đáp án đúng: C

Giải thích: ASP.NET Core không giữ lại các phiên trống.

Mức độ: Khó

Câu hỏi: Giải thích tại sao việc lưu trữ dữ liệu Session State trong bộ nhớ (in-memory) của server (sử dụng AddDistributedMemoryCache()) không phải là giải pháp lý tưởng cho các ứng dụng web được triển khai trên nhiều server (load-balanced environment). Đề xuất một giải pháp thay thế.

Đáp án:

Tại sao không lý tưởng:

* **Không đồng bộ hóa**: Khi ứng dụng được triển khai trên nhiều server (ví dụ: trong môi trường load-balanced), mỗi server sẽ có bộ nhớ riêng. Nếu dữ liệu Session được lưu trữ in-memory trên một server, và yêu cầu tiếp theo của cùng một người dùng lại được định tuyến đến một server khác, server thứ hai sẽ không tìm thấy dữ liệu Session đó, dẫn đến mất trạng thái của người dùng.
* **Mất dữ liệu khi khởi động lại server**: Nếu server lưu trữ dữ liệu Session in-memory bị khởi động lại hoặc gặp sự cố, tất cả dữ liệu Session trên server đó sẽ bị mất.
* **Không mở rộng (Scalability)**: Giải pháp in-memory không mở rộng tốt vì nó không chia sẻ trạng thái giữa các server, gây khó khăn trong việc mở rộng ứng dụng theo chiều ngang.

Giải pháp thay thế:

Sử dụng một Distributed Cache (bộ nhớ đệm phân tán) làm backing store cho Session State. Các tùy chọn phổ biến bao gồm:

* **Distributed SQL Server Cache**: Lưu trữ dữ liệu Session trong một cơ sở dữ liệu SQL Server.
* **Distributed Redis Cache**: Lưu trữ dữ liệu Session trong Redis, một kho dữ liệu trong bộ nhớ mã nguồn mở, thường được sử dụng làm bộ nhớ đệm phân tán.
* **Distributed NCache Cache**: Một bộ nhớ đệm phân tán trong bộ nhớ khác được phát triển cho .NET. Các giải pháp này cho phép dữ liệu Session được chia sẻ và truy cập bởi tất cả các server trong cụm, đảm bảo tính nhất quán của trạng thái người dùng ngay cả khi yêu cầu được định tuyến đến các server khác nhau.

**Giải thích**: Để đảm bảo tính nhất quán và khả năng mở rộng trong môi trường đa server, việc sử dụng một bộ nhớ đệm phân tán là cần thiết cho Session State.

Câu hỏi: Bạn muốn lưu trữ một đối tượng phức tạp (ví dụ: UserPreferences bao gồm Theme, Language, FontSize) vào Session State. Hãy viết một đoạn code mẫu để lưu trữ và đọc đối tượng này từ Session State, giả sử bạn đã cấu hình Session State đúng cách.

Đáp án:

Định nghĩa đối tượng:

C#

public class UserPreferences

{

public string Theme { get; set; }

public string Language { get; set; }

public int FontSize { get; set; }

}

**Lưu trữ vào Session**:

C#

// Trong Controller Action hoặc nơi có HttpContext

public IActionResult SetPreferences()

{

var preferences = new UserPreferences

{

Theme = "Dark",

Language = "en-US",

FontSize = 14

};

// Chuyển đổi đối tượng thành JSON string

string jsonPreferences = System.Text.Json.JsonSerializer.Serialize(preferences);

// Lưu vào Session

HttpContext.Session.SetString("UserPreferences", jsonPreferences);

return Content("User preferences saved to session.");

}

**Đọc từ Session**:

C#

// Trong Controller Action hoặc nơi có HttpContext

public IActionResult GetPreferences()

{

// Đọc JSON string từ Session

string jsonPreferences = HttpContext.Session.GetString("UserPreferences");

if (string.IsNullOrEmpty(jsonPreferences))

{

return Content("User preferences not found in session.");

}

// Chuyển đổi JSON string trở lại đối tượng

UserPreferences preferences = System.Text.Json.JsonSerializer.Deserialize<UserPreferences>(jsonPreferences);

return Content($"Theme: {preferences.Theme}, Language: {preferences.Language}, Font Size: {preferences.FontSize}");

}

**Giải thích**: Session State trong ASP.NET Core chỉ hỗ trợ lưu trữ byte[] hoặc string. Do đó, để lưu trữ các đối tượng phức tạp, bạn cần serialize chúng thành JSON (hoặc XML) trước khi lưu và deserialize chúng khi đọc ra.

Câu hỏi: Phân tích sự khác biệt về vòng đời và mục đích sử dụng giữa HttpContext.Items và TempData.

Đáp án:

HttpContext.Items:

* **Vòng đời**: Dữ liệu được lưu trữ trong HttpContext.Items chỉ tồn tại trong một yêu cầu HTTP duy nhất. Nó được tạo khi yêu cầu bắt đầu và bị hủy khi yêu cầu kết thúc.
* **Mục đích**: Dùng để truyền dữ liệu giữa các thành phần khác nhau trong cùng một yêu cầu HTTP (ví dụ: giữa Middleware và Controller, hoặc giữa các Filter và Action Method). Nó là một cách để chia sẻ dữ liệu "per-request".
* **Cơ chế lưu trữ**: Dữ liệu được lưu trữ trong bộ nhớ của server cho yêu cầu đó.
* **Ví dụ**: Lưu trữ thông tin người dùng đã xác thực để các thành phần khác trong pipeline có thể truy cập mà không cần xác thực lại.

**TempData**:

* **Vòng đời**: Dữ liệu trong TempData được thiết kế để tồn tại trong yêu cầu hiện tại và yêu cầu HTTP tiếp theo (thường là sau một thao tác chuyển hướng - redirect). Sau khi được đọc, dữ liệu sẽ bị xóa (trừ khi được đánh dấu Keep()).
* **Mục đích**: Dùng để truyền dữ liệu giữa các Action Method khi có một thao tác chuyển hướng (redirect). Rất hữu ích cho các thông báo "flash" (ví dụ: "Dữ liệu đã được lưu thành công!") hoặc các thông báo lỗi sau khi người dùng thực hiện một hành động.
* **Cơ chế lưu trữ**: TempData được triển khai bởi các nhà cung cấp TempData (mặc định là sử dụng Session State hoặc Cookies).
* **Ví dụ**: Sau khi người dùng gửi một form và được chuyển hướng đến trang xác nhận, TempData có thể được sử dụng để hiển thị thông báo thành công.

**Giải thích**: HttpContext.Items là cho dữ liệu trong một yêu cầu, trong khi TempData là cho dữ liệu cần tồn tại qua một lần chuyển hướng.

**Phần 6: Entity Framework Core**

Mức độ: Dễ

Câu hỏi: EF Core là viết tắt của từ gì?

A. Entity Framework Core

B. Essential Framework Core

C. Extended Features Core

D. Entity Function Core

Đáp án đúng: A

Giải thích: EF Core là một Object/Relational Mapping (O/RM) framework.

Câu hỏi: Hai phương pháp phát triển chính được hỗ trợ bởi EF Core là gì?

A. Database-First và API-First

B. Code-First và Database-First

C. Model-First và View-First

D. Schema-First và Data-First

Đáp án đúng: B

Giải thích: EF Core hỗ trợ cả Code-First (chủ yếu) và Database-First.

Câu hỏi: Đối tượng nào trong EF Core đại diện cho một phiên làm việc với cơ sở dữ liệu và được sử dụng để truy vấn và lưu trữ dữ liệu?

A. DbSet

B. DbContext

C. Entity

D. Migration

Đáp án đúng: B

Giải thích: DbContext là cầu nối giữa các thực thể miền của bạn và cơ sở dữ liệu.

Câu hỏi: LINQ là viết tắt của từ gì?

A. Language Integrated Network Query

B. Logic Independent Query

C. Language Integrated Query

D. Live Interactive Query

Đáp án đúng: C

Giải thích: LINQ cung cấp một cú pháp truy vấn thống nhất trong C# và VB.NET.

Câu hỏi: Để thêm một thực thể mới vào cơ sở dữ liệu bằng EF Core, bạn sử dụng phương thức nào trên DbSet?

A. Update()

B. Remove()

C. Add()

D. Find()

Đáp án đúng: C

Giải thích: Phương thức Add() được sử dụng để đánh dấu một thực thể là "Added" trong Change Tracker.

Mức độ: Trung bình

Câu hỏi: Khi sử dụng phương pháp Code-First trong EF Core, làm thế nào để EF Core xác định khóa chính (Primary Key) cho một thực thể theo quy ước (convention)?

A. Bất kỳ thuộc tính nào có tên là Id.

B. Thuộc tính có tên là <Tên\_Lớp\_Thực\_thể>Id (ví dụ: StudentId cho lớp Student) hoặc Id (không phân biệt chữ hoa chữ thường).

C. Thuộc tính đầu tiên được khai báo trong lớp.

D. Thuộc tính có kiểu dữ liệu là int.

Đáp án đúng: B

Giải thích: EF Core sẽ tạo khóa chính cho thuộc tính có tên là Id hoặc <Tên\_Lớp\_Thực\_thể>Id (ví dụ: StudentId cho lớp Student), không phân biệt chữ hoa chữ thường.

Câu hỏi: Để lưu các thay đổi (thêm, sửa, xóa) vào cơ sở dữ liệu sau khi thực hiện các thao tác trên DbSet, bạn cần gọi phương thức nào trên DbContext?

A. SaveChanges()

B. Commit()

C. ApplyChanges()

D. Persist()

Đáp án đúng: A

Giải thích: SaveChanges() hoặc SaveChangesAsync() là phương thức chịu trách nhiệm gửi các thay đổi từ Change Tracker đến cơ sở dữ liệu.

Câu hỏi: Cho đoạn mã LINQ sau: context.Books.Where(b => b.Author == "Jane Doe").ToList();. Đây là ví dụ về cú pháp LINQ nào?

A. Query Syntax

B. Method Syntax

C. SQL Syntax

D. Fluent Syntax

Đáp án đúng: B

Giải thích: Đây là Method Syntax (còn gọi là Fluent Syntax) vì nó sử dụng các phương thức mở rộng như Where() và ToList().

Câu hỏi: EF Core có hỗ trợ Lazy Loading (tải lười) dữ liệu liên quan theo mặc định không?

A. Có, luôn luôn.

B. Không, nhưng có thể cấu hình để bật.

C. Chỉ khi sử dụng Database-First.

D. Chỉ khi sử dụng Code-First.

Đáp án đúng: B

Giải thích: Lazy Loading không được hỗ trợ theo mặc định trong EF Core 2.0 (theo tài liệu), nhưng có thể được cấu hình để bật trong các phiên bản sau.

Câu hỏi: Khi bạn có một mối quan hệ One-to-Many giữa hai thực thể (ví dụ: Course và Student), làm thế nào để EF Core theo quy ước xác định khóa ngoại (Foreign Key) trong thực thể phụ thuộc (Student)?

A. Bằng cách thêm một thuộc tính có tên là CourseId trong lớp Student.

B. Bằng cách thêm một thuộc tính có tên là Id trong lớp Student.

C. Bằng cách thêm một thuộc tính có tên là StudentId trong lớp Course.

D. EF Core không thể tự động xác định khóa ngoại.

Đáp án đúng: A

Giải thích: EF Core sẽ tự động tạo khóa ngoại trong thực thể phụ thuộc nếu có một thuộc tính có tên là <Tên\_Thực\_thể\_Chính>Id (ví dụ: CourseId trong Student nếu Course là thực thể chính).

Mức độ: Khó

Câu hỏi: Giải thích khái niệm "Conventions" (quy ước) trong Entity Framework Core và tại sao chúng lại quan trọng trong phương pháp Code-First.

Đáp án:

Khái niệm "Conventions":

Conventions là các quy tắc mặc định mà Entity Framework Core sử dụng để xây dựng mô hình cơ sở dữ liệu dựa trên các lớp thực thể (domain classes) của bạn.

Chúng giúp EF Core tự động suy luận cấu trúc cơ sở dữ liệu (ví dụ: tên bảng, tên cột, khóa chính, khóa ngoại, kiểu dữ liệu) mà không cần bạn phải cấu hình tường minh mọi thứ.

**Tại sao quan trọng trong Code-First**:

* **Giảm thiểu cấu hình**: Conventions giúp giảm đáng kể lượng code cấu hình mà bạn phải viết. Nếu các lớp thực thể của bạn tuân thủ các quy ước, EF Core có thể tự động tạo ra một mô hình cơ sở dữ liệu hợp lý.
* **Tăng tốc phát triển**: Bằng cách tự động hóa quá trình ánh xạ, conventions giúp bạn nhanh chóng bắt đầu phát triển và tập trung vào logic nghiệp vụ thay vì chi tiết cơ sở dữ liệu.
* **Dễ đọc và bảo trì**: Khi các quy ước được tuân thủ, code trở nên dễ đọc và dễ hiểu hơn vì cấu trúc cơ sở dữ liệu có thể được suy luận trực tiếp từ các lớp C#.
* **Khả năng ghi đè**: Mặc dù conventions cung cấp các mặc định, bạn luôn có thể ghi đè chúng bằng cách sử dụng Data Annotations hoặc Fluent API khi bạn cần kiểm soát chi tiết hơn về ánh xạ.

**Giải thích**: Conventions là nền tảng của phương pháp Code-First, cho phép EF Core tự động ánh xạ các lớp C# thành cấu trúc cơ sở dữ liệu một cách hiệu quả.

Câu hỏi: Bạn có một lớp Order và một lớp OrderDetail với mối quan hệ One-to-Many. Một Order có thể có nhiều OrderDetail, và mỗi OrderDetail thuộc về một Order duy nhất. Hãy viết các lớp C# để thiết lập mối quan hệ One-to-Many này theo quy ước của EF Core, bao gồm cả thuộc tính điều hướng (navigation property).

Đáp án:

C#

public class Order

{

public int OrderId { get; set; } // Primary Key by convention

public DateTime OrderDate { get; set; }

public decimal TotalAmount { get; set; }

// Navigation property for the 'many' side (collection of OrderDetails)

public ICollection<OrderDetail> OrderDetails { get; set; }

}

public class OrderDetail

{

public int OrderDetailId { get; set; } // Primary Key by convention

public int ProductId { get; set; }

public string ProductName { get; set; }

public int Quantity { get; set; }

public decimal Price { get; set; }

// Foreign Key property by convention (matches primary key of Order)

public int OrderId { get; set; }

// Navigation property for the 'one' side (single Order)

public Order Order { get; set; }

}

**Giải thích**:

* OrderId trong Order là khóa chính theo quy ước.
* OrderDetailId trong OrderDetail là khóa chính theo quy ước.
* OrderId trong OrderDetail là khóa ngoại theo quy ước vì nó khớp với tên khóa chính của thực thể chính (Order) và có kiểu dữ liệu phù hợp.
* ICollection<OrderDetail> OrderDetails trong Order là thuộc tính điều hướng cho phía "nhiều".
* Order Order trong OrderDetail là thuộc tính điều hướng cho phía "một". Với cấu trúc này, EF Core sẽ tự động nhận diện mối quan hệ One-to-Many giữa Order và OrderDetail.

Câu hỏi: Phân biệt giữa Query Syntax và Method Syntax trong LINQ. Khi nào bạn nên ưu tiên sử dụng cú pháp này hơn cú pháp kia?

Đáp án:

Query Syntax (Cú pháp truy vấn):

* **Đặc điểm**: Giống với cú pháp SQL, bắt đầu bằng từ khóa from.
* **Ví dụ**:

C#

var result = from s in context.Students

where s.Age > 20

orderby s.Name

select s;

* **Ưu điểm**: Dễ đọc và quen thuộc hơn đối với những người có kinh nghiệm với SQL, đặc biệt là cho các truy vấn phức tạp có nhiều mệnh đề join, group by.
* **Nhược điểm**: Không phải tất cả các toán tử truy vấn chuẩn đều có thể được biểu diễn bằng Query Syntax (ví dụ: FirstOrDefault(), Count()).

**Method Syntax (Cú pháp phương thức / Fluent Syntax)**:

* **Đặc điểm**: Sử dụng các phương thức mở rộng (extension methods) trên các đối tượng IEnumerable<T> hoặc IQueryable<T>.
* **Ví dụ**:

C#

var result = context.Students

.Where(s => s.Age > 20)

.OrderBy(s => s.Name)

.ToList();

* **Ưu điểm**: Linh hoạt hơn, hầu hết các toán tử LINQ đều có sẵn dưới dạng phương thức. Nó cũng dễ dàng kết hợp với các phương thức không phải LINQ (như ToList(), ToArray()). Thường được ưu tiên trong các kịch bản cần chuỗi các thao tác.
* **Nhược điểm**: Có thể trở nên khó đọc hơn với các truy vấn rất phức tạp do chuỗi các phương thức.

**Khi nào nên ưu tiên sử dụng**:

* **Query Syntax**: Thích hợp cho các truy vấn có cú pháp tương tự SQL, đặc biệt khi có nhiều join, group by hoặc các mệnh đề phức tạp khác mà bạn muốn biểu diễn một cách tường minh.
* **Method Syntax**: Phù hợp hơn trong hầu hết các trường hợp, đặc biệt khi bạn cần sự linh hoạt cao hơn hoặc khi truy vấn không quá phức tạp và có thể được thể hiện một cách rõ ràng bằng chuỗi phương thức. Đây cũng là cú pháp được sử dụng phổ biến hơn trong cộng đồng .NET hiện đại.

**Giải thích**: Cả hai cú pháp đều dịch thành cùng một biểu thức cây và cuối cùng được dịch sang SQL bởi EF Core. Việc lựa chọn cú pháp chủ yếu phụ thuộc vào sở thích cá nhân và độ phức tạp của truy vấn.

**Phần 7: Thực hành Code - ASP.NET Core MVC Basics**

Mức độ: Dễ

**Câu hỏi**: Cho đoạn mã Controller sau:

C#

// Controllers/HomeController.cs

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

}

Và cấu hình định tuyến mặc định trong Program.cs:

C#

// Program.cs

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

Nếu người dùng truy cập URL /, Action Method nào sẽ được thực thi?

A. HomeController.Index()

B. HomeController.View()

C. Không có Action nào được thực thi, sẽ có lỗi.

D. Cần thêm [HttpGet] cho Index().

Đáp án đúng: A

Giải thích: Với định tuyến mặc định, khi URL là /, controller sẽ mặc định là Home và action sẽ mặc định là Index.

**Câu hỏi**: Cho đoạn mã Controller sau:

C#

// Controllers/ProductController.cs

public class ProductController : Controller

{

public string GetName()

{

return "Laptop XYZ";

}

}

Nếu người dùng truy cập URL /Product/GetName, kết quả trả về cho trình duyệt sẽ là gì?

A. Một trang HTML trống.

B. Lỗi 404 Not Found.

C. Chuỗi "Laptop XYZ".

D. Lỗi biên dịch.

Đáp án đúng: C

Giải thích: Action Method trả về kiểu string sẽ được render trực tiếp thành nội dung phản hồi HTTP.

**Câu hỏi**: Để truyền một thông báo đơn giản từ Controller sang View bằng ViewData, bạn sẽ điền vào chỗ trống \_\_\_\_ và \_\_\_\_ trong đoạn mã sau:

C#

// Controller

public IActionResult About()

{

ViewData["Message"] = "Welcome to our About page!";

return View();

}

// View (About.cshtml)

<h1>\_\_\_\_</h1>

<p>\_\_\_\_</p>

A. @ViewData["Title"], @ViewData["Message"]

B. About, @ViewData["Message"]

C. @ViewData["Message"], About

D. ViewData.Title, ViewData.Message

Đáp án đúng: B

Giải thích: <h1>About</h1> là tiêu đề tĩnh, và @ViewData["Message"] được sử dụng để hiển thị giá trị từ ViewData.

**Câu hỏi**: Cho đoạn mã sau trong một Controller:

C#

public IActionResult Privacy()

{

return Redirect("https://www.google.com");

}

Nếu người dùng truy cập Action Privacy, điều gì sẽ xảy ra?

A. Ứng dụng sẽ hiển thị một View có tên "https://www.google.com".

B. Trình duyệt của người dùng sẽ được chuyển hướng đến Google.

C. Sẽ có lỗi runtime vì không tìm thấy View.

D. Phương thức Redirect không hợp lệ.

Đáp án đúng: B

Giải thích: Redirect() tạo ra một phản hồi HTTP redirect, yêu cầu trình duyệt chuyển hướng đến URL được cung cấp.

Câu hỏi: Để thêm dịch vụ Session State vào ứng dụng ASP.NET Core, bạn cần thêm dòng code nào vào phương thức ConfigureServices trong Program.cs?

A. app.UseSession();

B. services.AddSession();

C. services.AddMvc();

D. app.UseMvc();

Đáp án đúng: B

Giải thích: services.AddSession() đăng ký các dịch vụ cần thiết cho Session State.

Mức độ: Trung bình

**Câu hỏi**: Cho đoạn mã Controller và Model sau:

C#

// Models/Product.cs

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

// Controllers/ProductController.cs

public class ProductController : Controller

{

[HttpPost]

public IActionResult Create(Product product)

{

// Giả sử dữ liệu gửi lên là: Name=Laptop&Price=1200.50

// Giá trị của product.Id sẽ là gì?

return Ok();

}

}

Nếu dữ liệu gửi lên là Name=Laptop&Price=1200.50, giá trị của product.Id sẽ là gì khi Action Create được gọi?

A. 0

B. null

C. 1

D. Lỗi Model Binding.

Đáp án đúng: A

Giải thích: Id là một kiểu giá trị (int), nếu không có giá trị nào được cung cấp từ request, nó sẽ nhận giá trị mặc định là 0.

**Câu hỏi**: Cho đoạn mã sau:

C#

// Controllers/OrderController.cs

public class OrderController : Controller

{

public IActionResult Detail(int id)

{

if (id <= 0)

{

return BadRequest("Invalid Order ID.");

}

// Logic để lấy chi tiết đơn hàng

return View();

}

}

Nếu người dùng truy cập URL /Order/Detail?id=0, HTTP status code trả về sẽ là gì?

A. 200 OK

B. 400 Bad Request

C. 404 Not Found

D. 500 Internal Server Error

Đáp án đúng: B

Giải thích: Phương thức BadRequest() trả về HTTP status code 400.

Câu hỏi: Bạn muốn tạo một liên kết đến Action Privacy trong HomeController bằng Anchor Tag Helper. Hãy điền vào chỗ trống \_\_\_\_ và \_\_\_\_ để hoàn thành đoạn mã sau:

<a asp-\_\_\_\_="Home" asp-\_\_\_\_="Privacy">Privacy Policy</a>

A. action, controller

B. controller, action

C. route, name

D. href, link

Đáp án đúng: B

Giải thích: asp-controller chỉ định tên Controller và asp-action chỉ định tên Action.

**Câu hỏi**: Cho đoạn mã sau trong một Controller:

C#

public IActionResult ProcessForm(string name, int age)

{

if (string.IsNullOrEmpty(name))

{

ModelState.AddModelError("name", "Tên không được để trống.");

}

if (age < 18)

{

ModelState.AddModelError("age", "Tuổi phải từ 18 trở lên.");

}

if (!ModelState.IsValid)

{

return View(); // Trả về View với lỗi validation

}

return Content($"Hello {name}, you are {age} years old.");

}

Nếu người dùng gửi request POST đến /ControllerName/ProcessForm với dữ liệu form là name=&age=15, kết quả trả về sẽ là gì?

A. Chuỗi "Hello , you are 15 years old."

B. Một View trống.

C. Một View với các thông báo lỗi "Tên không được để trống." và "Tuổi phải từ 18 trở lên.".

D. Lỗi 500 Internal Server Error.

Đáp án đúng: C

Giải thích: ModelState.IsValid sẽ là false do các lỗi được thêm vào, và View() sẽ được trả về, hiển thị các lỗi này nếu View có Validation Summary hoặc Validation Message Tag Helpers.

**Câu hỏi**: Bạn muốn lưu trữ một giá trị vào Session State và sau đó đọc nó. Hãy điền vào chỗ trống \_\_\_\_ và \_\_\_\_ trong đoạn mã sau:

C#

// Lưu vào Session

HttpContext.Session.SetString("UserName", "Alice");

// Đọc từ Session

string userName = HttpContext.Session.\_\_\_\_("UserName");

A. GetString

B. Get

C. ReadString

D. LoadString

Đáp án đúng: A

Giải thích: Phương thức GetString() được sử dụng để đọc một chuỗi từ Session State.

Mức độ: Khó

**Câu hỏi**: Cho đoạn mã sau:

C#

// Models/User.cs

public class User

{

public int Id { get; set; }

public string Username { get; set; }

public string Password { get; set; }

public string Email { get; set; }

}

// Controllers/UserController.cs

public class UserController : Controller

{

[HttpPost]

public IActionResult Update([Bind("Id,Username,Email")] User user)

{

// Giả sử dữ liệu gửi lên là: Id=1&Username=newuser&Password=newpass&Email=new@example.com

// Giá trị của user.Password sẽ là gì sau khi Model Binding?

return Ok();

}

}

Nếu dữ liệu gửi lên là Id=1&Username=newuser&Password=newpass&Email=new@example.com, giá trị của user.Password sẽ là gì sau khi Model Binding?

A. "newpass"

B. null

C. Một chuỗi rỗng ""

D. Lỗi Model Binding.

Đáp án đúng: B

Giải thích: Thuộc tính [Bind("Id,Username,Email")] chỉ định rõ ràng rằng chỉ các thuộc tính Id, Username, và Email được phép bind. Thuộc tính Password sẽ bị bỏ qua và giữ nguyên giá trị mặc định của nó (là null cho kiểu string).

Câu hỏi: Bạn đang phát triển một API và muốn một Action Method chỉ chấp nhận dữ liệu JSON từ request body. Hãy viết đoạn mã Action Method và Model cần thiết để nhận một đối tượng Product từ request body.

Đáp án:

C#

// Models/Product.cs

public class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

}

// Controllers/ApiController.cs

[ApiController] // Recommended for API controllers

[Route("api/[controller]")]

public class ProductsApiController : ControllerBase

{

[HttpPost]

public IActionResult CreateProduct([FromBody] Product product)

{

if (product == null)

{

return BadRequest("Product data is null.");

}

// Logic để lưu sản phẩm vào cơ sở dữ liệu

// Ví dụ: \_productService.AddProduct(product);

return CreatedAtAction(nameof(GetProduct), new { id = 1 }, product); // Giả sử ID là 1

}

[HttpGet("{id}")]

public IActionResult GetProduct(int id)

{

// Logic để lấy sản phẩm

return Ok(new Product { Name = "Sample Product", Price = 99.99m });

}

}

**Giải thích**:

* [ApiController] và ControllerBase là các khuyến nghị cho API Controller, cung cấp các tính năng như tự động validation và inferring binding sources.
* [FromBody] trên tham số Product product chỉ thị cho Model Binder rằng dữ liệu cho đối tượng product sẽ đến từ phần thân của HTTP request. ASP.NET Core sẽ tự động sử dụng một formatter (ví dụ: JSON formatter) để deserialize dữ liệu từ request body thành đối tượng Product.

Câu hỏi: Bạn muốn tạo một View Component để hiển thị danh sách các sản phẩm nổi bật. Hãy viết đoạn mã tối thiểu cho View Component Class và cách gọi nó trong một View Razor.

Đáp án:

View Component Class:

C#

// ViewComponents/FeaturedProductsViewComponent.cs

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

public class FeaturedProductsViewComponent : ViewComponent

{

// Giả sử có một dịch vụ để lấy sản phẩm

// private readonly IProductService \_productService;

// public FeaturedProductsViewComponent(IProductService productService)

// {

// \_productService = productService;

// }

public async Task<IViewComponentResult> InvokeAsync(int numberOfProducts)

{

// Logic để lấy danh sách sản phẩm nổi bật

// Ví dụ: var products = await \_productService.GetFeaturedProducts(numberOfProducts);

var products = new List<string>

{

"Product A", "Product B", "Product C"

};

if (numberOfProducts < products.Count)

{

products = products.GetRange(0, numberOfProducts);

}

return View(products); // Truyền Model đến View của View Component

}

}

**View của View Component** (thường là Views/Shared/Components/FeaturedProducts/Default.cshtml):

C#

@model List<string>

<h3>Featured Products</h3>

<ul>

@foreach (var product in Model)

{

<li>@product</li>

}

</ul>

**Cách gọi trong một View Razor** (ví dụ: Index.cshtml):

HTML

<h1>Welcome to our store!</h1>

@\* Gọi View Component bằng Tag Helper \*@

<vc:featured-products number-of-products="2"></vc:featured-products>

@\* Hoặc gọi bằng HTML Helper \*@

@await Component.InvokeAsync("FeaturedProducts", new { numberOfProducts = 3 })

**Giải thích**:

* **View Component Class** phải kế thừa từ ViewComponent hoặc có tên kết thúc bằng ViewComponent.
* Logic chính nằm trong phương thức InvokeAsync (hoặc Invoke cho đồng bộ).
* **View của View Component** được tìm kiếm theo quy ước trong thư mục Views/Shared/Components/{View Component Name}/.
* Bạn có thể truyền tham số cho View Component thông qua các thuộc tính trong Tag Helper (number-of-products) hoặc thông qua một đối tượng ẩn danh trong InvokeAsync.

**Phần 8: Thực hành Code - EF Core Basics**

Mức độ: Dễ

**Câu hỏi**: Cho đoạn mã Model và DbContext sau:

C#

// Models/Book.cs

public class Book

{

public int Id { get; set; }

public string Title { get; set; }

public string Author { get; set; }

}

// Data/ApplicationDbContext.cs

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Book> Books { get; set; }

}

Để thêm một cuốn sách mới vào cơ sở dữ liệu, bạn sẽ điền vào chỗ trống \_\_\_\_ trong đoạn mã sau:

C#

public void AddNewBook(string title, string author)

{

using (var context = new ApplicationDbContext(\_\_\_\_)) // Giả sử options đã được cung cấp

{

var newBook = new Book { Title = title, Author = author };

context.Books.Add(newBook);

context.SaveChanges();

}

}

A. new DbContextOptions<ApplicationDbContext>()

B. options

C. null

D. this.options

Đáp án đúng: B

Giải thích: DbContextOptions là đối tượng cấu hình cho DbContext, thường được inject vào constructor.

**Câu hỏi**: Cho đoạn mã LINQ sau:

C#

using (var context = new ApplicationDbContext(options))

{

var firstBook = context.Books.FirstOrDefault(b => b.Author == "Jane Doe");

// ...

}

Nếu không có cuốn sách nào của tác giả "Jane Doe" trong cơ sở dữ liệu, giá trị của firstBook sẽ là gì?

A. Một đối tượng Book với các thuộc tính mặc định.

B. null

C. Một ngoại lệ (exception).

D. Một danh sách trống.

Đáp án đúng: B

Giải thích: FirstOrDefault() trả về phần tử đầu tiên khớp với điều kiện hoặc giá trị mặc định của kiểu (là null cho các kiểu tham chiếu) nếu không tìm thấy.

**Câu hỏi**: Để lấy tất cả các cuốn sách từ cơ sở dữ liệu, bạn sẽ điền vào chỗ trống \_\_\_\_ trong đoạn mã sau:

C#

public List<Book> GetAllBooks()

{

using (var context = new ApplicationDbContext(options))

{

return context.Books.\_\_\_\_;

}

}

A. ToList()

B. ToArray()

C. AsEnumerable()

D. SelectAll()

Đáp án đúng: A

Giải thích: ToList() thực thi truy vấn và trả về kết quả dưới dạng List<T>.

**Câu hỏi**: Để cập nhật thông tin của một cuốn sách đã tồn tại trong cơ sở dữ liệu, bạn sẽ điền vào chỗ trống \_\_\_\_ trong đoạn mã sau:

C#

public void UpdateBookTitle(int bookId, string newTitle)

{

using (var context = new ApplicationDbContext(options))

{

var bookToUpdate = context.Books.Find(bookId);

if (bookToUpdate != null)

{

bookToUpdate.Title = newTitle;

context.\_\_\_\_; // Đánh dấu thực thể là đã thay đổi

context.SaveChanges();

}

}

}

A. Books.Update(bookToUpdate)

B. Entry(bookToUpdate).State = EntityState.Modified

C. Update(bookToUpdate)

D. Không cần dòng code nào ở đây, EF Core tự động theo dõi.

Đáp án đúng: D

Giải thích: Khi bạn lấy một thực thể từ DbContext và thay đổi các thuộc tính của nó, Change Tracker của EF Core sẽ tự động phát hiện những thay đổi này. Bạn chỉ cần gọi SaveChanges().

**Câu hỏi**: Để xóa một cuốn sách khỏi cơ sở dữ liệu, bạn sẽ điền vào chỗ trống \_\_\_\_ trong đoạn mã sau:

C#

public void DeleteBook(int bookId)

{

using (var context = new ApplicationDbContext(options))

{

var bookToDelete = context.Books.Find(bookId);

if (bookToDelete != null)

{

context.Books.\_\_\_\_(bookToDelete);

context.SaveChanges();

}

}

}

A. Delete

B. Remove

C. Clear

D. Detach

Đáp án đúng: B

Giải thích: Phương thức Remove() được sử dụng để đánh dấu một thực thể là "Deleted" trong Change Tracker.

Mức độ: Trung bình

**Câu hỏi**: Cho đoạn mã sau:

C#

public class Author

{

public int AuthorId { get; set; }

public string Name { get; set; }

public ICollection<Book> Books { get; set; } // Navigation property

}

public class Book

{

public int BookId { get; set; }

public string Title { get; set; }

public int AuthorId { get; set; } // Foreign Key

public Author Author { get; set; } // Navigation property

}

// Trong DbContext

public DbSet<Author> Authors { get; set; }

public DbSet<Book> Books { get; set; }

Để lấy tất cả các cuốn sách cùng với thông tin tác giả của chúng (sử dụng Eager Loading), bạn sẽ điền vào chỗ trống \_\_\_\_ trong đoạn mã sau:

C#

public List<Book> GetBooksWithAuthors()

{

using (var context = new ApplicationDbContext(options))

{

return context.Books.\_\_\_\_.ToList();

}

}

A. Include(b => b.Author)

B. Load(b => b.Author)

C. Join(b => b.Author)

D. Select(b => b.Author)

Đáp án đúng: A

Giải thích: Phương thức Include() được sử dụng để tải dữ liệu liên quan (navigation properties) cùng với thực thể chính trong một truy vấn duy nhất (Eager Loading).

**Câu hỏi**: Cho đoạn mã LINQ sau:

C#

using (var context = new ApplicationDbContext(options))

{

var query = from b in context.Books

where b.Author == "Jane Doe"

select b;

// ...

}

Truy vấn này sẽ được thực thi trên cơ sở dữ liệu vào thời điểm nào?

A. Ngay lập tức khi dòng var query = ... được gán.

B. Khi query được lặp qua (ví dụ: trong một vòng lặp foreach) hoặc khi một phương thức thực thi truy vấn (như ToList(), Count()) được gọi.

C. Khi DbContext được khởi tạo.

D. Khi SaveChanges() được gọi.

Đáp án đúng: B

Giải thích: LINQ to Entities sử dụng "Deferred Execution" (thực thi trì hoãn). Truy vấn chỉ được thực thi khi kết quả thực sự cần thiết.

**Câu hỏi**: Bạn muốn thêm một Author mới và một Book mới liên quan đến Author đó trong cùng một SaveChanges() transaction. Hãy điền vào chỗ trống \_\_\_\_ và \_\_\_\_ trong đoạn mã sau:

C#

public void AddAuthorAndBook(string authorName, string bookTitle)

{

using (var context = new ApplicationDbContext(options))

{

var newAuthor = new Author { Name = authorName };

context.Authors.Add(newAuthor);

var newBook = new Book { Title = bookTitle, Author = newAuthor }; // Gán đối tượng Author

context.Books.Add(newBook);

context.\_\_\_\_(); // Lưu cả Author và Book

}

}

A. SaveChanges

B. Commit

C. SubmitChanges

D. Persist

Đáp án đúng: A

Giải thích: EF Core sẽ tự động theo dõi mối quan hệ giữa newAuthor và newBook và lưu cả hai thực thể trong cùng một transaction khi SaveChanges() được gọi.

Câu hỏi: Để thực hiện một truy vấn LINQ mà chỉ trả về một tập hợp con các thuộc tính từ một thực thể (ví dụ: chỉ Title và Author của Book), bạn sẽ sử dụng toán tử LINQ nào?

A. Where

B. OrderBy

C. Select

D. GroupBy

Đáp án đúng: C

Giải thích: Toán tử Select được sử dụng để chiếu (project) các thuộc tính mong muốn vào một kiểu mới (thường là một đối tượng ẩn danh hoặc một DTO).

Câu hỏi: Khi nào bạn nên sử dụng AsNoTracking() trong một truy vấn EF Core?

A. Khi bạn muốn EF Core theo dõi các thay đổi của thực thể.

B. Khi bạn chỉ muốn đọc dữ liệu và không có ý định cập nhật chúng, để cải thiện hiệu suất.

C. Khi bạn muốn tải dữ liệu liên quan một cách lười biếng (Lazy Loading).

D. Khi bạn muốn thực hiện một truy vấn SQL thô.

Đáp án đúng: B

Giải thích: AsNoTracking() ngăn EF Core theo dõi các thực thể được trả về bởi truy vấn. Điều này có thể cải thiện hiệu suất cho các truy vấn chỉ đọc vì EF Core không cần thiết lập Change Tracking.

Mức độ: Khó

**Câu hỏi**: Bạn có một lớp Student và Course với mối quan hệ Many-to-Many thông qua một lớp Enrollment (bảng nối).

C#

public class Student

{

public int StudentId { get; set; }

public string Name { get; set; }

public ICollection<Enrollment> Enrollments { get; set; }

}

public class Course

{

public int CourseId { get; set; }

public string Title { get; set; }

public ICollection<Enrollment> Enrollments { get; set; }

}

public class Enrollment

{

public int EnrollmentId { get; set; } // Primary Key for join table

public int StudentId { get; set; }

public Student Student { get; set; }

public int CourseId { get; set; }

public Course Course { get; set; }

public DateTime EnrollmentDate { get; set; }

}

// Trong DbContext

public DbSet<Student> Students { get; set; }

public DbSet<Course> Courses { get; set; }

public DbSet<Enrollment> Enrollments { get; set; }

Viết đoạn mã LINQ để lấy tất cả các Student cùng với tất cả các Course mà họ đã đăng ký, và EnrollmentDate của mỗi lần đăng ký.

Đáp án:

C#

public List<Student> GetStudentsWithCoursesAndEnrollmentDates()

{

using (var context = new ApplicationDbContext(options))

{

var studentsWithCourses = context.Students

.Include(s => s.Enrollments) // Eager load Enrollments

.ThenInclude(e => e.Course) // Then eager load Course for each Enrollment

.ToList();

// Hoặc nếu bạn muốn một cấu trúc dữ liệu phẳng hơn:

/\*

var studentCourseData = context.Enrollments

.Select(e => new

{

StudentName = e.Student.Name,

CourseTitle = e.Course.Title,

EnrollmentDate = e.EnrollmentDate

})

.ToList();

\*/

return studentsWithCourses;

}

}

**Giải thích**:

* Include(s => s.Enrollments): Tải dữ liệu của bảng nối Enrollment cho mỗi Student.
* .ThenInclude(e => e.Course): Sau khi tải Enrollment, tải tiếp dữ liệu Course liên quan đến mỗi Enrollment. Điều này đảm bảo rằng bạn có thể truy cập Student.Enrollments.Course.Title mà không gây ra N+1 query problem.

Câu hỏi: Giải thích sự khác biệt giữa Find() và Where().FirstOrDefault() khi truy vấn một thực thể bằng khóa chính trong EF Core. Khi nào bạn nên ưu tiên sử dụng Find()?

Đáp án:

Find():

* **Cách hoạt động**:
  + Đầu tiên, Find() cố gắng tìm thực thể trong **Change Tracker** (bộ nhớ cục bộ của DbContext).
  + Nếu không tìm thấy trong Change Tracker, nó sẽ truy vấn cơ sở dữ liệu.
* **Kiểu trả về**: Trả về một thực thể hoặc null.
* **Ưu điểm**: Rất hiệu quả khi thực thể có thể đã được tải vào bộ nhớ của DbContext trước đó, tránh truy vấn cơ sở dữ liệu không cần thiết.
* **Nhược điểm**: Chỉ hoạt động với khóa chính (hoặc các khóa thay thế được cấu hình). Không thể sử dụng với các điều kiện truy vấn phức tạp.

**Where().FirstOrDefault()**:

* **Cách hoạt động**: Luôn luôn tạo và thực thi một truy vấn trên cơ sở dữ liệu (trừ khi truy vấn được tối ưu hóa bởi nhà cung cấp LINQ).
* **Kiểu trả về**: Trả về một thực thể hoặc null.
* **Ưu điểm**: Rất linh hoạt, có thể sử dụng với bất kỳ điều kiện truy vấn nào (không chỉ khóa chính) và kết hợp với các toán tử LINQ khác.
* **Nhược điểm**: Có thể kém hiệu quả hơn Find() nếu thực thể đã có trong Change Tracker, vì nó vẫn sẽ truy vấn cơ sở dữ liệu.

**Khi nào ưu tiên sử dụng Find()**:

* Khi bạn cần truy vấn một thực thể **chỉ bằng khóa chính** của nó.
* Khi bạn có khả năng thực thể đó đã được tải vào DbContext trong cùng một ngữ cảnh (ví dụ: trong cùng một request HTTP hoặc transaction).
* Khi bạn muốn tận dụng hiệu suất của việc tìm kiếm trong bộ nhớ cục bộ trước khi truy vấn cơ sở dữ liệu.

**Giải thích**: Find() là một phương thức tối ưu hóa cho việc tìm kiếm bằng khóa chính, tận dụng Change Tracker, trong khi Where().FirstOrDefault() là một phương thức truy vấn tổng quát hơn, luôn truy vấn cơ sở dữ liệu.

Câu hỏi: Bạn muốn thực hiện một truy vấn SQL thô (raw SQL query) để lấy dữ liệu từ cơ sở dữ liệu bằng EF Core. Viết đoạn mã để lấy tất cả các cuốn sách có Author là "Unknown" bằng SQL thô.

Đáp án:

C#

public List<Book> GetUnknownBooksRawSql()

{

using (var context = new ApplicationDbContext(options))

{

// Sử dụng FromSqlRaw cho các truy vấn SQL thô

// Đảm bảo tên bảng và cột khớp với cơ sở dữ liệu

var books = context.Books

.FromSqlRaw("SELECT \* FROM Books WHERE Author = {0}", "Unknown")

.ToList();

// Hoặc FromSqlInterpolated để tránh SQL Injection

/\*

string authorName = "Unknown";

var books = context.Books

.FromSqlInterpolated($"SELECT \* FROM Books WHERE Author = {authorName}")

.ToList();

\*/

return books;

}

}

**Giải thích**:

* FromSqlRaw(): Cho phép bạn thực thi một chuỗi SQL thô. Bạn phải cẩn thận với SQL Injection khi sử dụng phương thức này bằng cách sử dụng tham số hóa ({0}).
* FromSqlInterpolated(): Là một phiên bản an toàn hơn, sử dụng string interpolation để tự động tham số hóa các giá trị, giúp ngăn chặn SQL Injection.
* Kết quả của truy vấn SQL thô phải ánh xạ được vào kiểu thực thể (Book trong trường hợp này).