**ĐỀ THI THỬ SỐ 1**

**MÔN: PHÁT TRIỂN ỨNG DỤNG WEB NÂNG CAO**

**LỚP CTK44 – THỜI GIAN: 90 PHÚT**

**CÂU 1: (2 điểm)**

1. Học phần Phát triển ứng dụng Web nâng cao đã trải qua bao nhiêu buổi thực hành?

* Học phần Phát triển ứng dụng Web nâng cao đã trải qua 10 buổi thực hành

1. Có bao nhiêu bài thực hành (Lab) chính thức? Trong số đó, bạn thấy thích nhất là bài thực hành nào? Vì sao?

* Có 6 bài lab chính thức
* Trong số đó, em thích nhất là bài thực hành lab 1 bởi vì nó là nền tảng cho việc xây dựng nên hệ thống ứng dụng web (nhà thì phải có nền móng vững chắc)

1. Những nội dung chính của bài thực hành (Lab) số 3 là gì?

* Những nội dung chính của bài lab số 3 là

Cách tạo MVC Area và các thành phần M/V/C để xây dựng các chức năng cho một phân hệ riêng của ứng dụng. 

Cách xây dựng các chức năng thêm/cập nhật/xóa dữ liệu. 

Cách kiểm tra tính hợp lệ của dữ liệu sử dụng các Attribute có sẵn và sử dụng gói thư viện FluentValidation. 

Cách sử dụng gói thư viện Mapster để đơn giản hóa việc sao chép dữ liệu giữa các đối tượng. 

Cách cấu hình và ghi lại nhật ký hệ thống sử dụng gói thư viện NLog.

***Trả lời:***

**CÂU 2: (2 điểm)**

Cho ba lớp Category (danh mục), Candy (kẹo) và CandyContext như sau:

public class Category

{

// Mã số danh mục

public int Id { get; set; }

// Tên danh mục

public string Name { get; set; }

// Cho biết hiển thị lên menu hay không

public [……] ShowOnMenu { get; set; }

}

public class Candy

{

// Mã số

public int Id { get; set; }

// Tên mặt hàng kẹo

public string Name { get; set; }

// Giá bán (đơn vị là USD, luôn lớn hơn 0 và

// có phần lẻ thập phân. Ví dụ: 5.99).

public [……] Price { get; set; }

// Ngày hết hạn sử dụng

public [……] ExpirationDate { get; set; }

// Mã số danh mục (Mỗi mặt hàng kẹo phải thuộc

// đúng một danh mục)

public [……] CategoryId { get; set; }

public Category Category { get; set; }

}

public class CandyContext : DbContext

{

public DbSet<Category> Categories { get; set; }

public DbSet<Candy> Candies { get; set; }

}

1. Hãy xác định kiểu dữ liệu cho các thuộc tính ShowOnMenu, Price, ExpirationDate, và CategoryId. (Sinh viên sửa trực tiếp mã nguồn ở trên bằng cách thay tên kiểu dữ liệu vào chỗ [……])

public class Category

{

// Mã số danh mục

public int Id { get; set; }

// Tên danh mục

public string Name { get; set; }

// Cho biết hiển thị lên menu hay không

public bool ShowOnMenu { get; set; }

}

public class Candy

{

// Mã số

public int Id { get; set; }

// Tên mặt hàng kẹo

public string Name { get; set; }

// Giá bán (đơn vị là USD, luôn lớn hơn 0 và

// có phần lẻ thập phân. Ví dụ: 5.99).

public double Price { get; set; }

// Ngày hết hạn sử dụng

public DateTime ExpirationDate { get; set; }

// Mã số danh mục (Mỗi mặt hàng kẹo phải thuộc

// đúng một danh mục)

public int CategoryId { get; set; }

public Category Category { get; set; }

}

public class CandyContext : DbContext

{

public DbSet<Category> Categories { get; set; }

public DbSet<Candy> Candies { get; set; }

}

1. Giả sử, ta có lớp CandyService thực thi interface ICandyService chứa phương thức GetCandiesAsync. Hãy định nghĩa phương thức này để tìm các mặt hàng kẹo chưa hết hạn sử dụng theo tên mặt hàng (nếu giá trị truyền vào khác rỗng), đơn giá (miền giá min-max có giá trị hợp lệ), danh mục (nếu có truyền giá trị) hoặc tên danh mục (tìm chính xác theo tên danh mục nếu truyền giá trị khác rỗng) và sắp xếp kết quả tăng dần theo tên mặt hàng kẹo.

public async Task<IList<Candy>> GetCandiesAsync(

string name = null,

int? categoryId = null,

string categoryName = null,

decimal? minPrice = null,

decimal? maxPrice = null,

CancellationToken cancellationToken = default)

{

// Sinh viên viết mã lệnh ở đây

}

Đáp án

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using static System.Runtime.InteropServices.JavaScript.JSType;

public class CandyService : ICandyService

{

private readonly CandyContext \_context;

public CandyService(CandyContext context)

{

\_context = context;

}

public async Task<IList<Candy>> GetCandiesAsync(

string name = null,

int? categoryId = null,

string categoryName = null,

decimal? minPrice = null,

decimal? maxPrice = null,

CancellationToken cancellationToken = default)

{

IQueryable<Candy> query = \_context.Candies.Include(c => c.Category);

if (!string.IsNullOrEmpty(name))

{

query = query.Where(c => c.Name == name);

}

if (categoryId.HasValue)

{

query = query.Where(c => c.CategoryId == categoryId.Value);

}

if (!string.IsNullOrEmpty(categoryName))

{

query = query.Where(c => c.Category.Name == categoryName);

}

if (minPrice.HasValue)

{

query = query.Where(c => c.Price >= minPrice.Value);

}

if (maxPrice.HasValue)

{

query = query.Where(c => c.Price <= maxPrice.Value);

}

query = query.Where(c => c.ExpirationDate > DateTime.Now);

query = query.OrderBy(c => c.Name);

return await query.ToListAsync(cancellationToken);

}

public async Task<Candy> GetCandyByIdAsync(int candyId, CancellationToken cancellationToken = default)

{

return await \_context.Candies.FindAsync(candyId);

}

public async Task AddCandyAsync(Candy candy, CancellationToken cancellationToken = default)

{

\_context.Candies.Add(candy);

await \_context.SaveChangesAsync(cancellationToken);

}

public async Task UpdateCandyAsync(Candy candy, CancellationToken cancellationToken = default)

{

\_context.Entry(candy).State = EntityState.Modified;

await \_context.SaveChangesAsync(cancellationToken);

}

public async Task DeleteCandyAsync(int candyId, CancellationToken cancellationToken = default)

{

var candy = await \_context.Candies.FindAsync(candyId);

if (candy != null)

{

\_context.Candies.Remove(candy);

await \_context.SaveChangesAsync(cancellationToken);

}

}

public async Task<Candy> AddOrUpdateCandyAsync(Candy candy)

{

var existingCandy = await \_dbContext.Candies.FindAsync(candy.Id);

if (existingCandy == null)

{

\_dbContext.Candies.Add(candy);

}

else

{

existingCandy.Name = candy.Name;

existingCandy.Price = candy.Price;

existingCandy.ExpirationDate = candy.ExpirationDate;

existingCandy.CategoryId = candy.CategoryId;

}

await \_dbContext.SaveChangesAsync();

return candy;

}

}

**CÂU 3: (3 điểm)**

Bạn An sử dụng kết quả ở câu 2 để định nghĩa một API Endpoint như sau:

app.MapGet("/candies", (ICandyService service, CandyFilterModel model) =>

{

var candies = await service.GetCandiesAsync(

model.Name, model.CategoryId, model.CategoryName,

model.MinPrice, model.MaxPrice);

return candies;

})

.WithName("GetCandies")

.Produces<Candy>()

.WithOpenApi();

app.MapPut("/candies/{id}", async (int id, ICandyService service, Candy candy) =>

{

var existingCandy = await service.GetCandyByIdAsync(id);

if (existingCandy == null)

{

return Results.NotFound();

}

existingCandy.Name = candy.Name;

existingCandy.Price = candy.Price;

existingCandy.ExpirationDate = candy.ExpirationDate;

existingCandy.CategoryId = candy.CategoryId;

await service.UpdateCandyAsync(existingCandy);

return Results.Ok(existingCandy);

})

.WithName("UpdateCandy")

.Accepts<int>()

.Accepts<Candy>()

.Produces<Candy>()

.WithOpenApi();

app.MapPost("/candies", async (ICandyService service, Candy candy) =>

{

await service.AddCandyAsync(candy);

return Results.Created($"/candies/{candy.Id}", candy);

})

.WithName("AddCandy")

.Accepts<Candy>()

.Produces<Candy>()

.WithOpenApi();

app.MapDelete("/candies/{id}", async (int id, ICandyService service) =>

{

var existingCandy = await service.GetCandyByIdAsync(id);

if (existingCandy == null)

{

return Results.NotFound();

}

await service.DeleteCandyAsync(id);

return Results.NoContent();

})

.WithName("DeleteCandy")

.Accepts<int>()

.WithOpenApi();

1. Hãy giúp bạn An định nghĩa lớp CandyFilterModel.

class CandyFilterModel

{

public string Name { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

public string CategoryName { get; set; }

public double MinPrice { get; set; }

public double MaxPrice { get; set; }

}

1. Hãy giúp bạn An tìm những lỗi sai trong đoạn mã trên và sửa lại cho đúng.

Cần thêm từ khóa async trước lambda expression để sử dụng await.

Phương thức MapGet nhận một tham số là một lambda expression, vì vậy cần sử dụng từ khóa async để khai báo nó là một phương thức bất đồng bộ.

Phương thức Produces nên chấp nhận kiểu dữ liệu là IList<Candy> thay vì Candy.

Cuối cùng,có thể sử dụng phương thức WithOpenApi để cung cấp thông tin về API cho tài liệu OpenAPI.

app.MapGet("/candies", async (ICandyService service, CandyFilterModel model) =>

{

var candies = await service.GetCandiesAsync(

model.Name, model.CategoryId, model.CategoryName,

model.MinPrice, model.MaxPrice);

return candies;

})

.WithName("GetCandies")

.Produces<Candy>()

.WithOpenApi();

1. Hãy giải thích ý nghĩa của toàn bộ đoạn mã trên (sau khi đã sửa).

Đoạn mã trên định nghĩa một endpoint API cho phép lấy danh sách các kẹo dựa trên các tiêu chí lọc được chỉ định. Dưới đây là ý nghĩa của từng phần trong đoạn mã:

app.MapGet("/candies", async (ICandyService service, CandyFilterModel model) => { ... }): Đây là phương thức đăng ký một endpoint GET với đường dẫn "/candies" trong ứng dụng web. Khi có một yêu cầu GET được gửi đến "/candies", đoạn mã bên trong delegate sẽ được thực thi.

ICandyService service: Đây là một tham số của delegate, đại diện cho một thể hiện của interface ICandyService. Interface này chứa phương thức GetCandiesAsync để lấy danh sách kẹo.

CandyFilterModel model: Đây là một tham số của delegate, đại diện cho một đối tượng CandyFilterModel. Đối tượng này chứa các thông tin về tiêu chí lọc kẹo như tên, danh mục, giá, v.v.

var candies = await service.GetCandiesAsync(...): Trong đoạn mã này, chúng ta gọi phương thức GetCandiesAsync trên đối tượng service (thể hiện của ICandyService). Điều này sẽ lấy danh sách các kẹo dựa trên các tiêu chí lọc được truyền vào từ CandyFilterModel.

return candies;: Khi danh sách kẹo đã được lấy, chúng ta trả về danh sách đó dưới dạng kết quả của endpoint API. Điều này cho phép người dùng cuối hoặc các ứng dụng khác có thể nhận được danh sách kẹo thông qua giao diện API.

.WithName("GetCandies"): Đây là một phương thức mở rộng (extension method) cho endpoint API, cho phép đặt tên cho endpoint này là "GetCandies". Điều này có thể hữu ích khi tạo tài liệu API hoặc khi tham chiếu đến endpoint trong các phần khác của ứng dụng.

Produces<Candy>(): Đây là một phương thức mở rộng khác cho endpoint API, chỉ định kiểu dữ liệu mà endpoint này sẽ sản xuất (produce). Trong trường hợp này, endpoint sẽ sản xuất các đối tượng Candy.

WithOpenApi(): Đây là một phương thức mở rộng khác cho endpoint API, chỉ định rằng endpoint này sẽ có thông tin OpenAPI (Swagger) để mô tả và tài liệu hóa API. Điều này giúp việc tạo

1. Hãy nêu một đề xuất để cải thiện hiệu suất cho API Endpoint trên.

Để cải thiện hiệu suất cho API Endpoint trên, có thể áp dụng các biện pháp sau:

Caching: Sử dụng các cơ chế caching để lưu trữ kết quả truy vấn trước đó và tái sử dụng chúng cho các yêu cầu tiếp theo có cùng các tham số lọc. Việc sử dụng caching giúp giảm tải cho hệ thống cơ sở dữ liệu và cung cấp thời gian phản hồi nhanh hơn cho yêu cầu.

Paging: Nếu danh sách kẹo trả về rất lớn, hãy xem xét triển khai phân trang để chỉ trả về một phần kết quả trong mỗi yêu cầu. Điều này giúp giảm tải và giảm thời gian phản hồi của endpoint, đồng thời cung cấp trải nghiệm tốt hơn cho người dùng khi duyệt qua các trang kết quả.

Tối ưu hóa cơ sở dữ liệu: Đảm bảo rằng câu truy vấn dữ liệu được sử dụng trong phương thức GetCandiesAsync đã được tối ưu hóa. Sử dụng chỉ mục (indexing), tối ưu hóa câu truy vấn SQL, sử dụng các kỹ thuật như eager loading hoặc lazy loading (tùy thuộc vào trường hợp) để giảm thời gian truy vấn và tải dữ liệu từ cơ sở dữ liệu.

Sử dụng kỹ thuật lập chỉ mục (indexing): Đảm bảo rằng các trường dữ liệu quan trọng như tên mặt hàng, danh mục và giá đã được lập chỉ mục trong cơ sở dữ liệu. Điều này sẽ cải thiện hiệu suất khi thực hiện các truy vấn lọc dựa trên các trường này.

Sử dụng kỹ thuật lazy loading: Nếu có liên quan đến các mối quan hệ nhiều-đến-nhiều hoặc nhiều-đến-một, hãy sử dụng kỹ thuật lazy loading để tải dữ liệu chỉ khi nó được yêu cầu. Điều này giúp giảm tải và tối ưu hóa việc truy cập dữ liệu liên quan.

Tối ưu hóa mạng: Kiểm tra và tối ưu hóa việc truyền dữ liệu qua mạng. Có thể sử dụng nén dữ liệu (compression), tối ưu kích thước gói tin (packet size), sử dụng giao thức HTTP/2 hoặc giao thức giao tiếp khác như gRPC để tăng tốc độ truyền dữ liệu và giảm thời gian phản hồi.

Tối ưu hóa mã lệnh: Kiểm tra và tối ưu hóa các phần mã lệnh trong phương thức GetCandiesAsync. Xem xét sử dụng các kỹ thuật như lập chỉ mục trong mã lệnh, sử dụng các thuật toán tìm kiếm hiệu quả và giảm bớt số lượng phép tính không cần thiết.

Sử dụng kỹ thuật load balancing: Nếu hệ thống có nhiều server xử lý yêu cầu, hãy sử dụng kỹ thuật load balancing để phân phối tải đều giữa các server. Điều này giúp cải thiện khả năng mở rộng và khả năng chịu tải của hệ thống.

Đánh giá và tinh chỉnh hiệu suất: Thực hiện các bước đánh giá và tinh chỉnh hiệu suất để xác định các vấn đề tiềm ẩn và điểm yếu trong hệ thống. Sử dụng công cụ giám sát và ghi nhật ký (logging) để theo dõi hoạt động của API Endpoint và tìm hiểu về các vấn đề hiệu suất.

Cân nhắc sử dụng bộ nhớ cache: Xem xét việc lưu trữ kết quả truy vấn hoặc dữ liệu phổ biến trong bộ nhớ cache. Điều này giúp giảm thời gian truy cập cơ sở dữ liệu và cung cấp phản hồi nhanh hơn cho các yêu cầu lần sau.

Tuy nhiên, việc cải thiện hiệu suất của một API Endpoint cụ thể phụ thuộc vào nhiều yếu tố, bao gồm thiết kế hệ thống, quy mô dữ liệu, yêu cầu kinh doanh và tài nguyên có sẵn. Do đó, cần tiến hành đánh giá và tối ưu hóa trên cơ sở từng trường hợp cụ thể để đạt được hiệu suất tốt nhất.

**CÂU 4: (3 điểm)**

Sau khi xây dựng API Endpoint (ở câu 3) hoàn chỉnh, bạn An tạo một React component – tên là CandyList – để hiển thị danh sách mặt hàng kẹo lên ứng dụng Web. Bạn An định nghĩa mã nguồn cho component này như sau:

import React, { useState } from 'react'

const CandyList = () => {

    const [candies, setCandies] = useState([]);

    fetch(`https://localhost:7076/candies`)

        .then(response => response.json())

        .then(data => setCandies(data));

    return (

        <div class="product-list">

            {candies.map((item) => (

               <div className="product-item">

                    <h1>item.name</h1>

                    <p>$ { item.price }</p>

                </div>

            ))}

        </div>

    )

}

export default CandyList;

1. Hãy cho biết đoạn mã trên chứa những lỗi sai hoặc cảnh báo nào? Giải thích nguyên nhân gây ra lỗi.

Lỗi "missing dependency": Trong hàm CandyList, cần import useEffect từ thư viện React và sử dụng nó để gọi API và cập nhật state. Do đó, cần thêm dòng import và sử dụng useEffect để gọi API chỉ khi component được render.

Cảnh báo "missing key prop": Trong vòng lặp .map() để hiển thị danh sách các kẹo, cần cung cấp thuộc tính key cho mỗi phần tử trong mảng. Thuộc tính key giúp React xác định các phần tử duy nhất trong danh sách và tăng hiệu suất khi cập nhật UI. Do đó, cần thêm thuộc tính key cho mỗi phần tử trong vòng lặp .map(), ví dụ: key={item.id} hoặc key={item.name}.

Các lỗi và cảnh báo trên xảy ra do việc sử dụng fetch để gọi API bên trong hàm CandyList mà không sử dụng useEffect. Điều này dẫn đến việc gọi API mỗi khi component render, gây ra hiện tượng vòng lặp vô hạn và không cập nhật state candies đúng cách. Bằng cách sử dụng useEffect và chỉ chạy một lần trong quá trình render, chúng ta đảm bảo rằng API chỉ được gọi một lần khi component được hiển thị và kết quả được cập nhật vào state candies để hiển thị lên giao diện.

1. Hãy giúp bạn An cách sửa lỗi và viết lại mã nguồn đúng.

import React, { useState, useEffect } from 'react';

const CandyList = () => {

    const [candies, setCandies] = useState([]);

    useEffect(() => {

        const fetchData = async () => {

            try {

                const response = await fetch(`https://localhost:7076/candies`);

                const data = await response.json();

                setCandies(data);

            } catch (error) {

                console.error('Error fetching candies:', error);

            }

        };

        fetchData();

    }, []);

    return (

        <div className="product-list">

            {candies.map((item) => (

                <div className="product-item" key={item.id}>

                    <h1>{item.name}</h1>

                    <p>$ {item.price}</p>

                </div>

            ))}

        </div>

    );

};

export default CandyList;

1. Hãy bổ sung mã lệnh để hiển thị dòng thông báo “***Đang tải dữ liệu …***” trong khi chờ kết quả trả về từ API hoặc thông báo lỗi “***Không thể tải dữ liệu***” nếu gặp lỗi khi gọi API.

import React, { useState, useEffect } from 'react';

const CandyList = () => {

const [candies, setCandies] = useState([]);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(false);

useEffect(() => {

const fetchData = async () => {

try {

const response = await fetch(`https://localhost:7076/candies`);

const data = await response.json();

setCandies(data);

setLoading(false);

} catch (error) {

console.error('Error fetching candies:', error);

setError(true);

setLoading(false);

}

};

fetchData();

}, []);

if (loading) {

return <p>Đang tải dữ liệu...</p>;

}

if (error) {

return <p>Không thể tải dữ liệu</p>;

}

return (

<div className="product-list">

{candies.map((item) => (

<div className="product-item" key={item.id}>

<h1>{item.name}</h1>

<p>$ {item.price}</p>

</div>

))}

</div>

);

};

export default CandyList;

**--- HẾT ---**