

LAB 4.1. WEB API (Tiếp theo)

Thời lượng: Làm ở nhà

A. Mục tiêu

Bài thực hành này được thực hiện sau khi sinh viên hoàn tất các yêu cầu trong bài Lab 4 và chuẩn bị cho bài Lab 5. Thông qua bài thực hành bổ sung này, sinh viên cần nắm được:

- Cách đóng gói (wrap) phản hồi cùng với ngoại lệ.
- Thêm siêu dữ liệu (metadata) vào phản hồi, chẳng hạn phân trang, mã trạng thái HTTP, chi tiết lỗi,...
- Sử dụng Swagger để xem thông tin API.

Yêu cầu: Sinh viên tự làm phần “B. Hướng dẫn thực hành” ở nhà và nộp lên hệ thống LMS.

B. Hướng dẫn thực hành

Hầu hết các API phổ biến hiện nay đều trả về một phản hồi nhất quán với cấu trúc tương tự cho tất cả các yêu cầu dù thành công hay thất bại. Điều đó có nghĩa là việc sử dụng tất cả các API của cùng một máy chủ API trở nên dễ dàng và trực quan hơn. Bên cạnh dữ liệu kết quả, chúng ta có thể bổ sung siêu dữ liệu kèm theo.

Lưu ý: Dự án WebAPI phải chọn Target Framework .NET 7.0. Nếu không thấy cần nâng cấp phiên bản Visual Studio mới hơn hoặc cài đặt .NET 7.0 SDK tại <https://dotnet.microsoft.com/en-us/download/dotnet/7.0>

Application

General

Output type

Specifies the type of application to build.

Console Application

Target framework

Specifies the version of .NET that the application targets. This option can have different values depending on which versions of .NET are installed on your computer.

.NET 7.0

Hình 1

1. Tạo ApiResponse

Trong thư mục Models, tạo tập tin ApiResponse.cs với nội dung như sau:

```
using FluentValidation.Results;
using System.Net;

namespace TatBlog.WebApi.Models;

public class ApiResponse
{
    public bool IsSuccess => Errors.Count == 0;
    public HttpStatusCode StatusCode { get; init; }
    public IList<string> Errors { get; init; }

    protected ApiResponse()
    {
        StatusCode = HttpStatusCode.OK;
        Errors = new List<string>();
    }

    public static ApiResponse<T> Success<T>(
        T result,
        HttpStatusCode statusCode = HttpStatusCode.OK)
    {
        return new ApiResponse<T>
        {
            Result = result,
            StatusCode = statusCode
        };
    }

    public static ApiResponse<T> FailWithResult<T>(
        HttpStatusCode statusCode,
        T result,
        params string[] errorMessages)
    {
        return new ApiResponse<T>()
        {
            Result = result,
            StatusCode = statusCode,
            Errors = new List<string>(errorMessages)
        };
    }

    public static ApiResponse Fail(
        HttpStatusCode statusCode,
        params string[] errorMessages)
    {

```

```
        if (errorMessages is null or { Length: 0 })
        {
            throw new ArgumentNullException(nameof(errorMessages));
        }

        return new ApiResponse()
        {
            StatusCode = statusCode,
            Errors = new List<string>(errorMessages)
        };
    }

    public static ApiResponse Fail(
        HttpStatusCode statusCode,
        ValidationResult validationResult)
    {
        return Fail(statusCode, validationResult.Errors
            .Select(x => x.ErrorMessage)
            .Where(e => !string.IsNullOrEmpty(e))
            .ToArray());
    }
}

public class ApiResponse<T> : ApiResponse
{
    public T Result { get; set; }
}
```

2. Sử dụng phản hồi ApiResponse

Trong tập tin AuthorEndpoints.cs, chỉnh sửa phương thức MapAuthorEndpoint như sau:

```
public static WebApplication MapAuthorEndpoints(this WebApplication app)
{
    var routeGroupBuilder = app.MapGroup("/api/authors");

    routeGroupBuilder.MapGet("/", GetAuthors)
        .WithName("GetAuthors")
        .Produces<ApiResponse<PaginationResult<AuthorItem>>>();

    routeGroupBuilder.MapGet("/{id:int}", GetAuthorDetails)
        .WithName("GetAuthorById")
        .Produces<ApiResponse<AuthorItem>>();

    routeGroupBuilder.MapGet("/{slug:regex(^[a-z0-9_-]+$)}/posts",
        GetPostsByAuthorSlug)
```

```

        .WithName("GetPostsByAuthorSlug")
        .Produces<ApiResponse<PaginationResult<PostDto>>>();

routeGroupBuilder.MapPost("/", AddAuthor)
    .AddEndpointFilter<ValidatorFilter<AuthorEditModel>>()
    .WithName("AddNewAuthor")
    .Produces(401)
    .Produces<ApiResponse<AuthorItem>>();

routeGroupBuilder.MapPost("/{id:int}/picture", SetAuthorPicture)
    .WithName("SetAuthorPicture")
    .Accepts<IFormFile>("multipart/form-data")
    .Produces(401)
    .Produces<ApiResponse<string>>();

routeGroupBuilder.MapPut("/{id:int}", UpdateAuthor)
    .WithName("UpdateAnAuthor")
    .Produces(401)
    .Produces<ApiResponse<string>>();

routeGroupBuilder.MapDelete("/{id:int}", DeleteAuthor)
    .WithName("DeleteAnAuthor")
    .Produces(401)
    .Produces<ApiResponse<string>>();

return app;
}

```

Mục đích của việc thay thế các mã trạng thái phản hồi HTTP (HTTP response status code) sang một model mới là ApiResponse sẽ hạn chế các Exception ở phía client nếu mã > 400. Lúc này WebApi luôn trả về mã 200 (OK) và chi tiết kết quả nằm trong model ApiResponse.

Thay đổi dữ liệu trả về của các phương thức khác bên trong AuthorEndpoints.cs:

```

private static async Task<IResult> GetAuthors(
    [AsParameters] AuthorFilterModel model,
    IAuthorRepository authorRepository)
{
    var authorsList = await authorRepository.GetPagedAuthorsAsync(model,
model.Name);
    var paginationResult = new
PaginationResult<AuthorItem>(authorsList);

    return Results.Ok(ApiResponse.Success(paginationResult));
}

private static async Task<IResult> GetAuthorDetails(

```

```
        int id,
        IAuthorRepository authorRepository,
        IMapper mapper)
    {
        var author = await authorRepository.GetCachedAuthorByIdAsync(id);

        return author == null
            ? Results.Ok(ApiResponse.Fail(HttpStatusCode.NotFound, $"Không
tìm thấy tác giả có mã số {id}"))
            :
Results.Ok(ApiResponse.Success(mapper.Map<AuthorItem>(author)));
    }

    private static async Task<IResult> GetPostsByAuthor(
        int id,
        [AsParameters] PagingModel pagingModel,
        IBlogRepository blogRepository)
    {
        var postQuery = new PostQuery()
        {
            AuthorId = id,
            PublishedOnly = true
        };

        var postsList = await blogRepository.GetPagedPostsAsync(
            postQuery, pagingModel,
            posts => posts.ProjectToType<PostDto>());

        var paginationResult = new PaginationResult<PostDto>(postsList);

        return Results.Ok(ApiResponse.Success(paginationResult));
    }

    private static async Task<IResult> GetPostsByAuthorSlug(
        [FromRoute] string slug,
        [AsParameters] PagingModel pagingModel,
        IBlogRepository blogRepository)
    {
        var postQuery = new PostQuery()
        {
            AuthorSlug = slug,
            PublishedOnly = true
        };

        var postsList = await blogRepository.GetPagedPostsAsync(
            postQuery, pagingModel,
            posts => posts.ProjectToType<PostDto>());
    }
```

```
var paginationResult = new PaginationResult<PostDto>(postsList);

return Results.Ok(ApiResponse.Success(paginationResult));
}

private static async Task<IResult> AddAuthor(
    AuthorEditModel model,
    IAuthorRepository authorRepository,
    IMapper mapper)
{
    if (await authorRepository.IsAuthorSlugExistedAsync(0,
model.UrlSlug))
    {
        return Results.Ok(ApiResponse.Fail(
            HttpStatusCode.Conflict, $"Slug '{model.UrlSlug}' đã được sử
dụng"));
    }

    var author = mapper.Map<Author>(model);
    await authorRepository.AddOrUpdateAsync(author);

    return Results.Ok(ApiResponse.Success(
        mapper.Map<AuthorItem>(author), HttpStatusCode.Created));
}

private static async Task<IResult> SetAuthorPicture(
    int id,
    IFormFile imageFile,
    IAuthorRepository authorRepository,
    IMediaManager mediaManager)
{
    var imageUrl = await mediaManager.SaveFileAsync(
        imageFile.OpenReadStream(),
        imageFile.FileName,
        imageFile.ContentType);

    if (string.IsNullOrEmpty(imageUrl))
    {
        return Results.Ok(ApiResponse.Fail(
            HttpStatusCode.BadRequest, "Không lưu được tập tin"));
    }

    await authorRepository.SetImageUrlAsync(id, imageUrl);
    return Results.Ok(ApiResponse.Success(imageUrl));
}

private static async Task<IResult> UpdateAuthor(
    int id,
```

```
    AuthorEditModel model,
    IValidator<AuthorEditModel> validator,
    IAuthorRepository authorRepository,
    IMapper mapper)
{
    var validationResult = await validator.ValidateAsync(model);

    if (!validationResult.IsValid)
    {
        return Results.Ok(ApiResponse.Fail(
            HttpStatusCode.BadRequest, validationResult));
    }

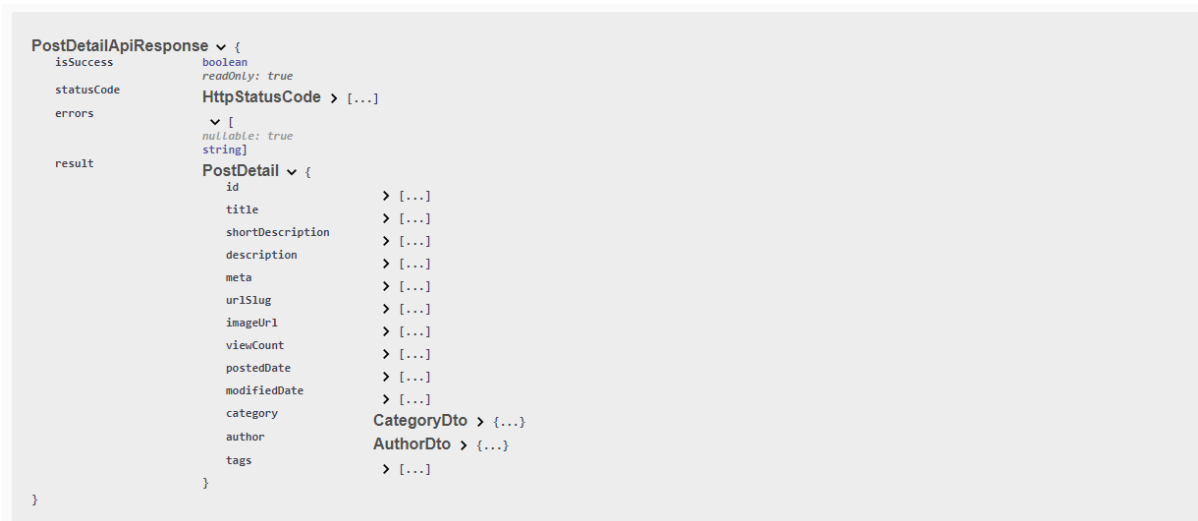
    if (await authorRepository.IsAuthorSlugExistedAsync(id,
model.UrlSlug))
    {
        return Results.Ok(ApiResponse.Fail(
            HttpStatusCode.Conflict,
            $"Slug '{model.UrlSlug}' đã được sử dụng"));
    }

    var author = mapper.Map<Author>(model);
    author.Id = id;

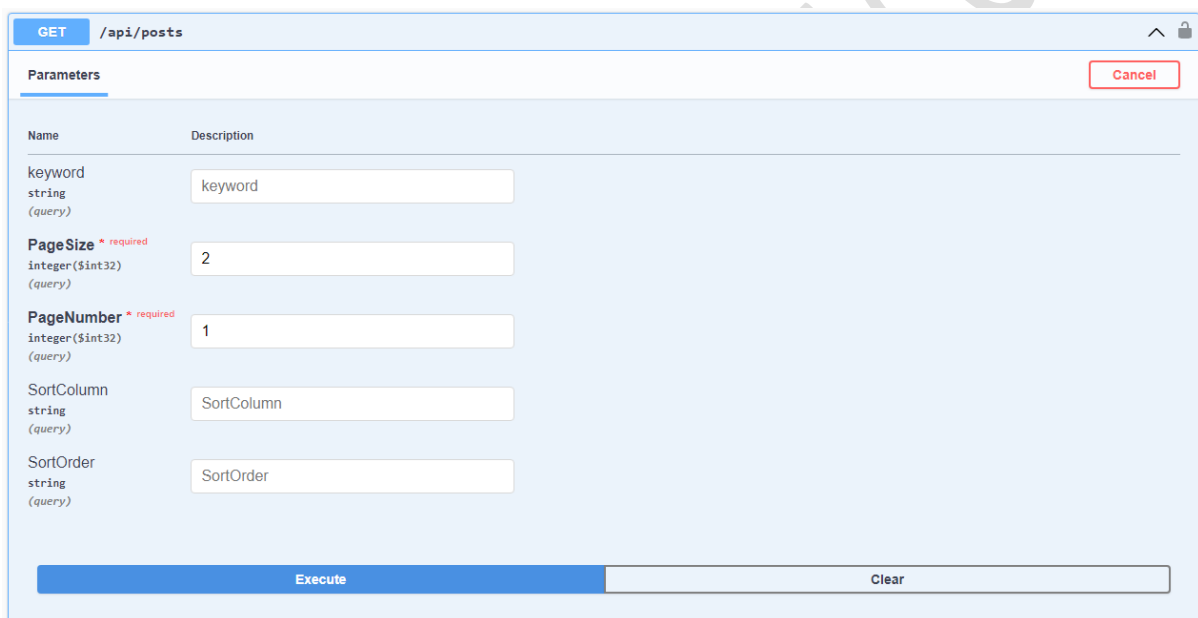
    return await authorRepository.AddOrUpdateAsync(author)
        ? Results.Ok(ApiResponse.Success("Author is updated",
HttpStatusCode.NoContent))
        : Results.Ok(ApiResponse.Fail(HttpStatusCode.NotFound, "Could
not find author"));
}

private static async Task<IResult> DeleteAuthor(int id,
IAuthorRepository authorRepository)
{
    return await authorRepository.DeleteAuthorAsync(id)
        ? Results.Ok(ApiResponse.Success("Author is deleted",
HttpStatusCode.NoContent))
        : Results.Ok(ApiResponse.Fail(HttpStatusCode.NotFound, "Could
not find author"));
}
```

Ví dụ cấu trúc kết quả trả về từ API:



Hình 2



Hình 3

Minh họa kết quả JSON:

```

{
  "result": {
    "items": [
      {
        "id": 15,
        "title": "JWT creation and validation in Python using Authlib",
        "shortDescription": "Authlib is a Python library that provides various OAuth, OpenID Connect, and JWT functionality. Authlib is my preferred library for JWT functionality, as it is one of the better Python implementations for JWT best practices, designed with OAuth and OpenID Connect in mind.",
        "urlSlug": "authlib-python-jwt",

```



```
"imageUrl": null,
"viewCount": 30,
"postedDate": "2022-08-25T18:44:00",
"modifiedDate": null,
"category": {
  "id": 3,
  "name": "Domain Driven Design",
  "urlSlug": "domain-driven-design"
},
"author": {
  "id": 1,
  "fullName": "Jason Mouth",
  "urlSlug": "jason-mouth"
},
"tags": [
  {
    "id": 7,
    "name": "Bootstrap",
    "urlSlug": "bootstrap"
  },
  {
    "id": 8,
    "name": "Tailwind CSS",
    "urlSlug": "tailwind-css"
  }
]
},
{
```

```
  "id": 14,
  "title": "C# Code Rules",
  "shortDescription": "The C# Compiler's name is Roslyn. Roslyn has a
very large set of analyzers to check the quality of your code, but you must
turn these analyzers on before they start doing anything. This post gives
you some quick information on why it's important to turn these analyzers on
in your C# projects, how to do that, and how to configure them.",
```

```
  "urlSlug": "code-rules",
  "imageUrl": null,
  "viewCount": 16,
  "postedDate": "2022-04-24T09:21:00",
  "modifiedDate": null,
  "category": {
    "id": 9,
    "name": "Practices",
    "urlSlug": "practices"
  },
  "author": {
    "id": 3,
    "fullName": "Kathy Smith",
```

```
        "urlSlug": "kathy-smith"
    },
    "tags": [
        {
            "id": 7,
            "name": "Bootstrap",
            "urlSlug": "bootstrap"
        }
    ]
},
],
"metadata": {
    "pageIndex": 0,
    "pageSize": 2,
    "totalItemCount": 15,
    "pageNumber": 1,
    "pageCount": 8,
    "hasPreviousPage": false,
    "hasNextPage": true,
    "firstItemIndex": 1,
    "lastItemIndex": 2,
    "isFirstPage": true,
    "isLastPage": false
}
},
"isSuccess": true,
"statusCode": 200,
"errors": []
}
```

Lưu ý: AuthorEndpoints.cs đã trả về ApiResponse nên tiếp tục bổ sung ở các lớp dưới còn thiếu.

3. Cập nhật API lấy danh sách chủ đề

Trong WebApi.Extensions.CategoryEndpoints.cs sửa lại phương thức GetCategories như sau:

```
private static async Task<IResult> GetCategories(
    IBlogRepository blogRepository)
{
    var categories = await blogRepository.GetCategoriesAsync();
    return Results.Ok(ApiResponse.Success(categories));
}
```

Đồng thời trong phương thức MapCategoryEndpoints sửa lại như sau:

```
public static WebApplication MapCategoryEndpoints(this WebApplication app)
{
    var routeGroupBuilder = app.MapGroup("/api/categories");
```

```
routeGroupBuilder.MapGet("/", GetCategories)
    .WithName("GetCategories")
    .Produces<ApiResponse<PaginationResult<CategoryItem>>>();
...
```

4. Tạo API lọc bài viết

Trong WebApi.Models.PostDetail bổ sung thêm dòng sau:

```
public bool Published { get; set; }
```

Tạo tập tin WebAPI.Models.PostFilterModel có nội dung như sau:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.ComponentModel;
using System.Globalization;

namespace TatBlog.WebApi.Models;

public class PostFilterModel
{
    [DisplayName("Từ khóa")]
    public string Keyword { get; set; }

    [DisplayName("Tác giả")]
    public int? AuthorId { get; set; }

    [DisplayName("Chủ đề")]
    public int? CategoryId { get; set; }

    [DisplayName("Năm")]
    public int? Year { get; set; }

    [DisplayName("Tháng")]
    public int? Month { get; set; }

    public IEnumerable<SelectListItem> AuthorList { get; set; }
    public IEnumerable<SelectListItem> CategoryList { get; set; }
    public IEnumerable<SelectListItem> MonthList { get; set; }

    public PostFilterModel()
    {
        MonthList = Enumerable.Range(1, 12)
            .Select(m => new SelectListItem()
            {
                Value = m.ToString(),
                Text =
                    CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(m)
            })
            .ToList();
    }
}
```

```

    }
}

```

Trong WebApi.Extensions.PostEndpoints bổ sung các phương thức sau:

```

...
public static WebApplication MapPostEndpoints(this WebApplication app)
{
    ...
    routeGroupBuilder.MapGet("/get-posts-filter", GetFilteredPosts)
        .WithName("GetFilteredPost")
        .Produces<ApiResponse<PostDto>>();

    routeGroupBuilder.MapGet("/get-filter", GetFilter)
        .WithName("GetFilter")
        .Produces<ApiResponse<PostFilterModel>>();
    return app;
}
...
private static async Task<IResult> GetFilter(
    IAuthorRepository authorRepository,
    IBlogRepository blogRepository)
{
    var model = new PostFilterModel()
    {
        AuthorList = (await authorRepository.GetAuthorsAsync())
            .Select(a => new SelectListItem()
            {
                Text = a.FullName,
                Value = a.Id.ToString()
            }),
        CategoryList = (await blogRepository.GetCategoriesAsync())
            .Select(c => new SelectListItem()
            {
                Text = c.Name,
                Value = c.Id.ToString()
            })
    };

    return Results.Ok(ApiResponse.Success(model));
}

private static async Task<IResult> GetFilteredPosts(
    [AsParameters] PostFilterModel model,
    [AsParameters] PagingModel pagingModel,
    IBlogRepository blogRepository)
{
    var postQuery = new PostQuery()
    {
        Keyword = model.Keyword,

```

```

        CategoryId = model.CategoryId,
        AuthorId = model.AuthorId,
        Year = model.Year,
        Month = model.Month,
    };
    var postsList = await blogRepository.GetPagedPostsAsync(
        postQuery, pagingModel, posts =>
posts.ProjectToType<PostDto>());

    var paginationResult = new PaginationResult<PostDto>(postsList);

    return Results.Ok(ApiResponse.Success(paginationResult));
}

```

...

Trong đó:

- Phương thức GetFilter(): Lấy toàn bộ danh sách tác giả và chủ đề mà không phân trang.
- Phương thức GetFilteredPosts(): Lấy các bài viết theo các yêu cầu khác nhau và có phân trang.

5. Tạo API chỉnh sửa bài viết

Tạo tập tin WebApi.Models.PostEditModel.cs:

Lưu ý: Lấy từ WebApp.Areas.Admin.Models.PostEditModel và thay đổi một vài phương thức trong đó.

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace TatBlog.WebApi.Models
{
    public class PostEditModel
    {
        public int Id { get; set; }

        [DisplayName("Tiêu đề")]
        [Required(ErrorMessage = "Tiêu đề không được để trống")]
        [MaxLength(500, ErrorMessage = "Tiêu đề tối đa 500 ký tự")]
        public string Title { get; set; }

        [DisplayName("Giới thiệu")]
        [Required]
        public string ShortDescription { get; set; }
    }
}

```

```
[DisplayName("Nội dung")]
[Required]
public string Description { get; set; }

[DisplayName("Metadata")]
[Required]
public string Meta { get; set; }

[DisplayName("Chọn hình ảnh")]
public IFormFile ImageFile { get; set; }

[DisplayName("Hình hiện tại")]
public string ImageUrl { get; set; }

[DisplayName("Xuất bản ngay")]
public bool Published { get; set; }

[DisplayName("Chủ đề")]
[Required]
public int CategoryId { get; set; }

[DisplayName("Tác giả")]
[Required]
public int AuthorId { get; set; }

[DisplayName("Từ khóa (mỗi từ 1 dòng)")]
[Required]
public string SelectedTags { get; set; }

public IEnumerable<SelectListItem> AuthorList { get; set; }

public IEnumerable<SelectListItem> CategoryList { get; set; }

public List<string> GetSelectedTags()
{
    return (SelectedTags ?? "")
        .Split(new[] { ',', ';', '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries)
        .ToList();
}

public static async ValueTask<PostEditModel> BindAsync(HttpContext
context)
{
    var form = await context.Request.ReadFormAsync();
    return new PostEditModel()
    {
        ImageFile = form.Files["ImageFile"],
```

```

        Id = int.Parse(form["Id"]),
        Title = form["Title"],
        ShortDescription = form["ShortDescription"],
        Description = form["Description"],
        Meta = form["Meta"],
        Published = form["Published"] != "false",
        CategoryId = int.Parse(form["CategoryId"]),
        AuthorId = int.Parse(form["AuthorId"]),
        SelectedTags = form["SelectedTags"]
    };
}
}
}

```

Lưu ý: Vì FromBody không được hỗ trợ trong .NET Core Minimal API nên chúng ta phải đọc FormData từ HttpContext.

Trong WebApi.Models.PostDto.cs bổ sung thuộc tính sau:

```
public bool Published { get; set; }
```

Trong phương thức MapPostEndpoints của WebApi.Extensions.PostEndpoints.cs, bổ sung:

```

...
routeGroupBuilder.MapPost("/", AddPost)
    .WithName("AddNewPost")
    .Accepts<PostEditModel>("multipart/form-data")
    .Produces(401)
    .Produces<ApiResponse<PostItem>>();
return app;
}
...

```

Cũng trong tập tin đó bổ sung phương thức AddPost như sau:

```

private static async Task<IResult> AddPost(
    HttpContext context,
    IBlogRepository blogRepository,
    IMapper mapper,
    IMediaManager mediaManager)
{
    var model = await PostEditModel.BindAsync(context);
    var slug = model.Title.GenerateSlug();
    if (await blogRepository.IsPostSlugExistedAsync(model.Id, slug))
    {
        return Results.Ok(ApiResponse.Fail(
            HttpStatusCode.Conflict, $"Slug '{slug}' đã được sử dụng cho
            bài viết khác"));
    }
}

```

```
var post = model.Id > 0 ? await
blogRepository.GetPostByIdAsync(model.Id) : null;

if (post == null)
{
    post = new Post()
    {
        PostedDate = DateTime.Now
    };
}

post.Title = model.Title;
post.AuthorId = model.AuthorId;
post.CategoryId = model.CategoryId;
post.ShortDescription = model.ShortDescription;
post.Description = model.Description;
post.Meta = model.Meta;
post.Published = model.Published;
post.ModifiedDate = DateTime.Now;
post.UrlSlug = model.Title.GenerateSlug();

if (model.ImageFile?.Length > 0)
{
    string hostname =
$"{context.Request.Scheme}://{context.Request.Host}{context.Request.PathBase
}"/,
    uploadedPath = await
mediaManager.SaveFileAsync(
        model.ImageFile.OpenReadStream(),
        model.ImageFile.FileName,
        model.ImageFile.ContentType);

    if (!string.IsNullOrEmpty(uploadedPath))
    {
        post.ImageUrl = uploadedPath;
    }
}

await blogRepository.CreateOrUpdatePostAsync(post,
model.GetSelectedTags());

return Results.Ok(ApiResponse.Success(
    mapper.Map<PostItem>(post), HttpStatusCode.Created));
}
```

--- HẾT ---