

## LAB 4. TAT-BLOG: WEB API

**Thời lượng: 4 tiết**

### A. Mục tiêu

Bài thực hành này yêu cầu sinh viên xây dựng các API endpoint cho ứng dụng TAT Blog. Trong những bài Lab tiếp theo, sinh viên sẽ xây dựng ứng dụng với React và gọi tới các API endpoint này để lấy dữ liệu hoặc cập nhật dữ liệu cho ứng dụng. Bảng sau liệt kê các API endpoint sẽ được xây dựng trong bài Lab này.

HTTP Method	Endpoint	Giải thích
GET	/api/authors	Lấy danh sách tác giả. Hỗ trợ tìm theo tên và phân trang kết quả.
GET	/api/authors/best/{limit}	Lấy danh sách N (limit) tác giả có nhiều bài viết nhất.
GET	/api/authors/{id}	Lấy thông tin chi tiết của tác giả có mã số (id) cho trước.
GET	/api/authors/{slug}/posts	Lấy danh sách những bài viết được đăng bởi tác giả có tên định danh (slug) cho trước. Hỗ trợ việc phân trang danh sách bài viết.
POST	/api/authors	Thêm một tác giả mới. Các thông tin cần thiết để thêm tác giả mới gồm có: <ul style="list-style-type: none"> <li>Tên tác giả (fullName)</li> <li>Slug (urlSlug)</li> <li>Ngày tham gia (joinedDate)</li> <li>Địa chỉ email (email)</li> <li>Các ghi chú khác (notes)</li> </ul>
POST	/api/authors/{id}/avatar	Upload một hình ảnh và sử dụng nó làm ảnh đại diện cho tác giả.
PUT	/api/authors/{id}	Cập nhật thông tin của tác giả có mã số (id) cho trước. Các thông tin cần cập nhật gồm có: <ul style="list-style-type: none"> <li>Tên tác giả (fullName)</li> <li>Slug (urlSlug)</li> <li>Ngày tham gia (joinedDate)</li> <li>Địa chỉ email (email)</li> <li>Các ghi chú khác (notes)</li> </ul>
DELETE	/api/authors/{id}	Xóa tác giả có mã số (id) cho trước

Sau khi hoàn thành bài thực hành này, sinh viên cần nắm vững:

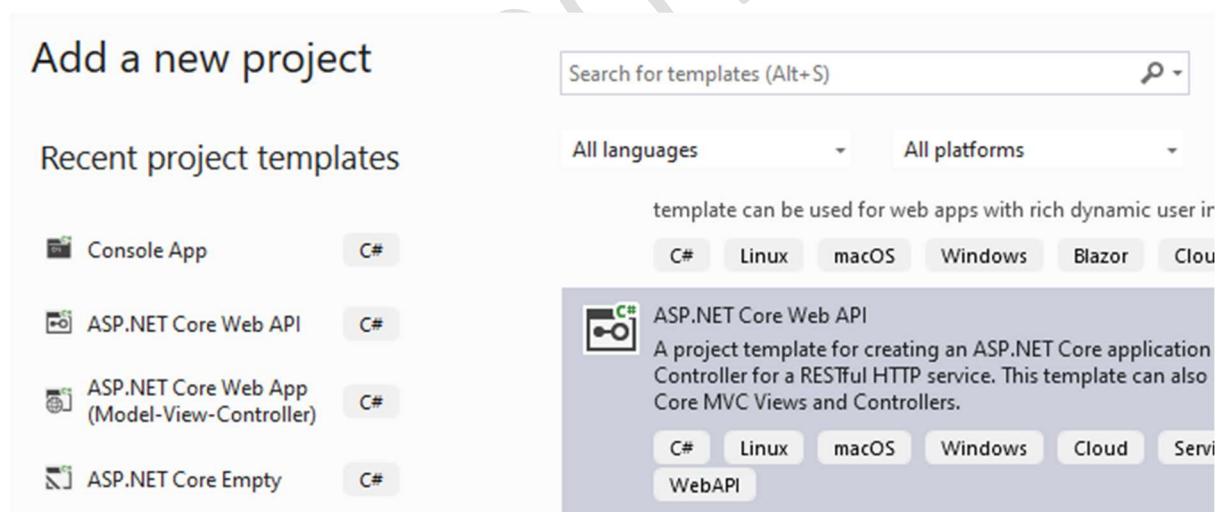
- Cách tạo dự án **Minimal API** và tổ chức cấu trúc của dự án.
- Cách định nghĩa các API endpoint, nhận dữ liệu từ client và trả về kết quả.
- Cách kiểm tra tính hợp lệ của dữ liệu được cung cấp bởi client sử dụng gói thư viện **FluentValidation**.
- Cách sử dụng gói thư viện **Mapster** để đơn giản hóa việc sao chép dữ liệu giữa các đối tượng.
- Cách xử lý và lưu trữ tập tin được tải lên bởi client.
- Cách định nghĩa và sử dụng các lớp **EndpointFilter**

**Yêu cầu: Sinh viên tự làm phần “B. Hướng dẫn thực hành” ở nhà và nộp lên hệ thống LMS. Tại phòng Lab, sinh viên làm phần “C. Bài tập thực hành” dựa trên dự án đã hoàn thành ở phần B.**

## B. Hướng dẫn thực hành

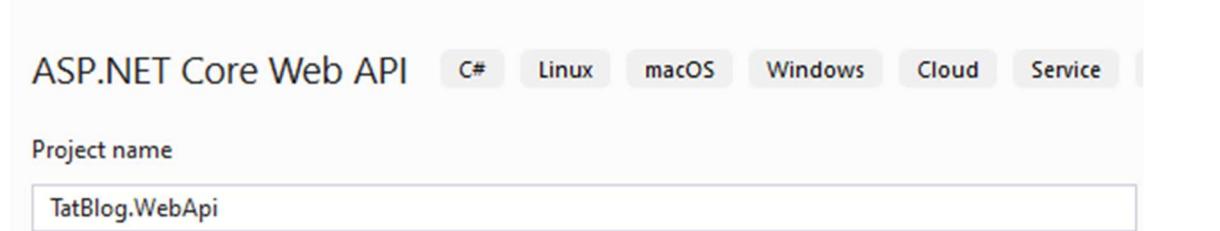
### 1. Tạo dự án Minimal API

Nhấp phải chuột vào solution **TipsAndTricks**, chọn **Add > New Project**. Trong cửa sổ mới, chọn **ASP.NET Core Web API**. Nhấn nút **Next**.



Đặt tên cho dự án mới là **TatBlog.WebApi**. Nhấn nút **Next**.

### Configure your new project



Trong cửa sổ tiếp theo, thiết lập các tùy chọn như hình dưới đây

Additional information

ASP.NET Core Web API    C#    Linux    macOS    Windows    Cloud    Service

Framework [i](#)

.NET 7.0 (Standard Term Support)

Authentication type [i](#)

None

Configure for HTTPS [i](#)

Enable Docker [i](#)

Docker OS [i](#)

Linux

Use controllers (uncheck to use minimal APIs) [i](#)

Enable OpenAPI support [i](#)

Do not use top-level statements [i](#)

Nhấn nút **Create** để hoàn thành việc tạo dự án. Nhấp phải chuột vào tên dự án **TatBlog.WebApi**, chọn **Set as startup project**. Sau đó, thiết lập tùy chọn **Nullable** là **disable**.

Cài đặt các gói thư viện sau cho dự án **TatBlog.WebApi**.

- FluentValidation.AspNetCore
- Mapster.DependencyInjection
- NLog.Web.AspNetCore

Thêm tham chiếu (**Add > Project Reference**) từ dự án **TatBlog.WebApi** đến các dự án:

- TatBlog.Core
- TatBlog.Data
- TatBlog.Services

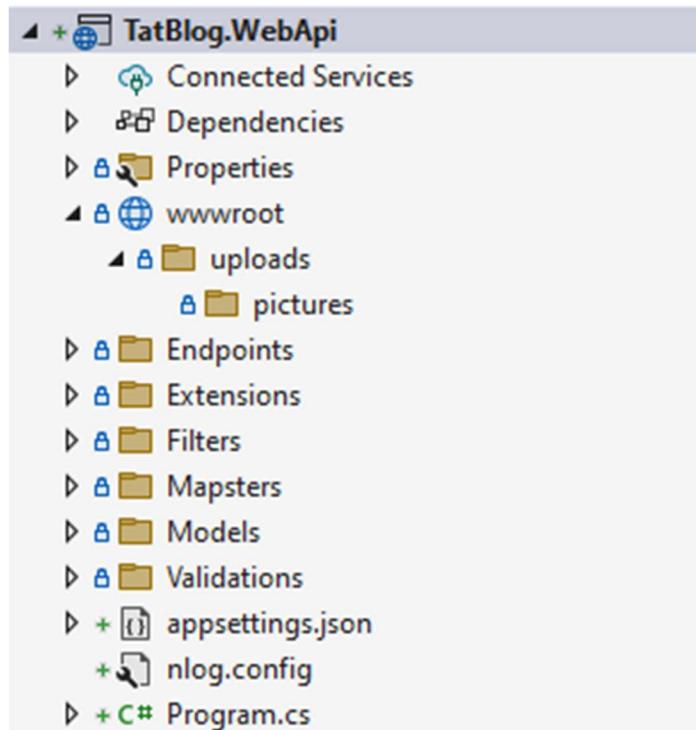
Tiếp theo, xóa bớt mã nguồn trong tập tin **Program.cs** để được kết quả như sau.

```
1 var builder = WebApplication.CreateBuilder(args);
2
3 var app = builder.Build();
4
5 app.Run();
```

Bổ sung chuỗi kết nối cơ sở dữ liệu vào tập tin **appsettings.json** (sinh viên có thể sao chép từ dự án **TatBlog.WebApp**).

Tham khảo phần **B.7. Ghi nhật ký hệ thống sử dụng NLog** trong **Lab 3** để tạo tập tin **nlog.config**.

Cũng trong dự án **TatBlog.WebApi**, tạo thêm các thư mục như trong hình dưới đây.



## 2. Cập nhật mã nguồn trong các dự án đã xây dựng

Sinh viên tải tập tin **Lab04-Resources.zip** từ hệ thống LMS và giải nén vào một thư mục.

Sao chép các tập tin **PagedList.cs**, **PaginationResult.cs**, **PagingMetadata.cs** trong thư mục vừa giải nén vào thư mục **TatBlog.Core > Collections**.

Sao chép các tập tin **IAuthorRepository.cs**, **AuthorRepository.cs** trong thư mục vừa giải nén vào thư mục **TatBlog.Services > Blogs**.

Trong interface **IBlogRepository** và lớp **BlogRepository**, tìm và xóa những phương thức liên quan đến việc đọc và cập nhật thông tin tác giả (vì chúng ta đã có lớp riêng **AuthorRepository** để thực hiện công việc này).

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có). Chẳng hạn, nếu chương trình báo lỗi không thể gọi hàm **GetAuthorsAsync** từ đối tượng **\_blogRepository** trong lớp **PostsController**, hãy thay đổi biến thành **\_authorRepository** vào lớp **PostsController** và gọi hàm **GetAuthorsAsync** từ biến này.

The screenshot shows two code snippets in a .NET IDE:

**Top Snippet (PostFilterModel.cs):**

```

148     1 reference | 0 changes | 0 authors, 0 changes
149     private async Task PopulatePostFilterModelAsync(
150         PostFilterModel model)
151     {
152         var authors = await _blogRepository.GetAuthorsAsync();
153         var categories = await _blogRepository.GetCategoriesAsync();
154
155         model.AuthorList = authors.Select(a => new SelectListItem()
156         {
157             Text = a.FullName,
158             Value = a.Id.ToString()
159         });
160
161         model.CategoryList = categories.Select(c => new SelectListItem()
162         {
163             Text = c.Name,
164             Value = c.Id.ToString()
165         });
166     }

```

**Bottom Snippet (PostsController.cs):**

```

14     public class PostsController : Controller
15     {
16         private readonly ILogger<PostsController> _logger;
17         private readonly IBlogRepository _blogRepository;
18         private readonly IAuthorRepository _authorRepository;
19         private readonly IMediaManager _mediaManager;
20         private readonly IMapper _mapper;
21
22         0 references | 0 changes | 0 authors, 0 changes
23         public PostsController(
24             ILogger<PostsController> logger,
25             IBlogRepository blogRepository,
26             IAuthorRepository authorRepository, ←
27             IMediaManager mediaManager,
28             IMapper mapper)
29         {
30             _logger = logger;
31             _blogRepository = blogRepository;
32             _authorRepository = authorRepository; ←
33             _mediaManager = mediaManager;
34             _mapper = mapper;

```

Annotations:

- A red arrow points to the error message in the IDE status bar: "IBlogRepository' does not contain a definition for 'GetAuthorsAsync' and no accessible extension method reference?"
- Three red arrows point to the four missing dependency注入 points in the constructor of PostsController: `IBlogRepository`, `IAuthorRepository`, `IMediaManager`, and `IMapper`.

Tiếp theo, trong dự án **TatBlog.Services**, tạo thêm thư mục **Timing**. Sao chép các tập tin **ITimeProvider.cs**, **LocalTimeProvider.cs**, **UtcTimeProvider.cs** trong thư mục đã giải nén vào thư mục **Timing** vừa tạo.

### 3. Cấu hình các dịch vụ được sử dụng trong dự án WebApi

Trong thư mục **TatBlog.WebApi > Extensions**, tạo lớp tĩnh **WebApplication-Extensions** và lần lượt cài đặt các phương thức mở rộng sau đây.

```
1 reference | 0 changes | 0 authors, 0 changes
12  public static WebApplicationBuilder ConfigureServices(
13      this WebApplicationBuilder builder)
14  {
15      builder.Services.AddMemoryCache();
16
17      builder.Services.AddDbContext<BlogDbContext>(options =>
18          options.UseSqlServer(
19              builder.Configuration
20                  .GetConnectionString("DefaultConnection")));
21
22      builder.Services
23          .AddScoped<ITimeProvider, LocalTimeProvider>();
24      builder.Services
25          .AddScoped<IMediaManager, LocalFileSystemMediaManager>();
26      builder.Services
27          .AddScoped<IBlogRepository, BlogRepository>();
28      builder.Services
29          .AddScoped<IAuthorRepository, AuthorRepository>();
30
31      return builder;
32  }
33
34  1 reference | 0 changes | 0 authors, 0 changes
35  public static WebApplicationBuilder ConfigureCors(
36      this WebApplicationBuilder builder)
37  {
38      builder.Services.AddCors(options =>
39      {
40          options.AddPolicy("TatBlogApp", policyBuilder =>
41              policyBuilder
42                  .AllowAnyOrigin()
43                  .AllowAnyHeader()
44                  .AllowAnyMethod());
45      });
46
47      return builder;
48  }
49
// Cấu hình việc sử dụng NLog
50  1 reference | 0 changes | 0 authors, 0 changes
51  public static WebApplicationBuilder ConfigureNLog(
52      this WebApplicationBuilder builder)
53  {
54      builder.Logging.ClearProviders();
55      builder.Host.UseNLog();
56
57      return builder;
58  }
```

```
1 reference | 0 changes | 0 authors, 0 changes
55  public static WebApplicationBuilder ConfigureSwaggerOpenApi(
56      this WebApplicationBuilder builder)
57  {
58      builder.Services.AddEndpointsApiExplorer();
59      builder.Services.AddSwaggerGen();
60
61      return builder;
62  }
63
64  1 reference | 0 changes | 0 authors, 0 changes
65  public static WebApplication SetupRequestPipeline(
66      this WebApplication app)
67  {
68      if (app.Environment.IsDevelopment())
69      {
70          app.UseSwagger();
71          app.UseSwaggerUI();
72
73          app.UseStaticFiles();
74          app.UseHttpsRedirection();
75
76          app.UseCors("TatBlogApp");
77
78      return app;
79  }
```

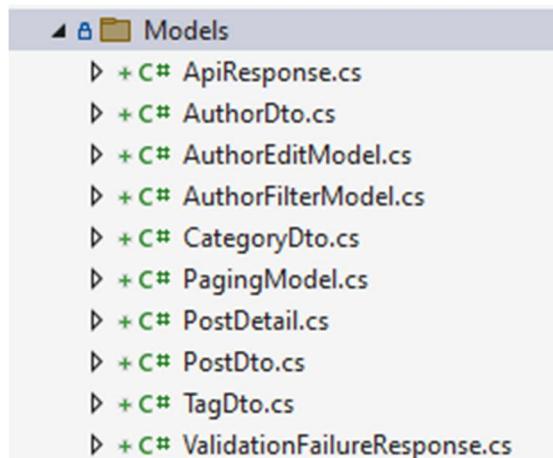
Cập nhật lại mã nguồn trong lớp **Program** để gọi đến các phương thức vừa định nghĩa.

```
1  using TatBlog.WebApi.Extensions;
2
3  var builder = WebApplication.CreateBuilder(args);
4  {
5      // Add services to the container.
6      builder
7          .ConfigureCors()
8          .ConfigureNLog()
9          .ConfigureServices()
10         .ConfigureSwaggerOpenApi();
11 }
12
13 var app = builder.Build();
14 {
15     // Configure the HTTP request pipeline.
16     app.SetupRequestPipeline();
17
18     app.Run();
19 }
```

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có). Chạy chương trình để xem kết quả.

## 4. Tạo các lớp DTO

Trong phần này, ta sẽ tạo các lớp để biểu diễn dữ liệu nhận được từ client và kết quả trả về cho client. Trước tiên, trong thư mục **TatBlog.WebApi > Models**, tạo các lớp như hình sau:



Lần lượt định nghĩa các lớp.

```

1 reference | 0 changes | 0 authors, 0 changes
5  public class PagingModel : IPagingParams
6  {
7      4 references | 0 changes | 0 authors, 0 changes
8          public int PageSize { get; set; } = 10;
9
10     3 references | 0 changes | 0 authors, 0 changes
11         public int PageNumber { get; set; } = 1;
12
13     3 references | 0 changes | 0 authors, 0 changes
14         public string SortColumn { get; set; } = "Id";
15
16     3 references | 0 changes | 0 authors, 0 changes
17         public string SortOrder { get; set; } = "DESC";
18 }

1 reference | 0 changes | 0 authors, 0 changes
3  public class AuthorFilterModel : PagingModel
4  {
5      1 reference | 0 changes | 0 authors, 0 changes
6          public string Name { get; set; }
7
8      2 references | 0 changes | 0 authors, 0 changes
9  public class AuthorDto
10  {
11      0 references | 0 changes | 0 authors, 0 changes
12          public int Id { get; set; }
13
14      0 references | 0 changes | 0 authors, 0 changes
15          public string FullName { get; set; }
16
17      0 references | 0 changes | 0 authors, 0 changes
18          public string UrlSlug { get; set; }
19
20      0 references | 0 changes | 0 authors, 0 changes
21  }
```

```
5 references | 0 changes | 0 authors, 0 changes
3 public class AuthorEditModel
4 {
5     1 reference | 0 changes | 0 authors, 0 changes
6     public string FullName { get; set; }
7
8     5 references | 0 changes | 0 authors, 0 changes
9     public string UrlSlug { get; set; }
10
11    1 reference | 0 changes | 0 authors, 0 changes
12    public DateTime JoinedDate { get; set; }
13
14    1 reference | 0 changes | 0 authors, 0 changes
15    public string Email { get; set; }
16
17    1 reference | 0 changes | 0 authors, 0 changes
18    public string Notes { get; set; }
19
20 }

2 references | 0 changes | 0 authors, 0 changes
3 public class CategoryDto
4 {
5     0 references | 0 changes | 0 authors, 0 changes
6     public int Id { get; set; }
7
8     0 references | 0 changes | 0 authors, 0 changes
9     public string Name { get; set; }
10
11    0 references | 0 changes | 0 authors, 0 changes
12    public string UrlSlug { get; set; }
13
14 }

2 references | 0 changes | 0 authors, 0 changes
3 public class TagDto
4 {
5     0 references | 0 changes | 0 authors, 0 changes
6     public int Id { get; set; }
7
8     0 references | 0 changes | 0 authors, 0 changes
9     public string Name { get; set; }
10
11    0 references | 0 changes | 0 authors, 0 changes
12    public string UrlSlug { get; set; }
13
14 }

4 references | 0 changes | 0 authors, 0 changes
3 public class ValidationFailureResponse
4 {
5     1 reference | 0 changes | 0 authors, 0 changes
6     public IEnumerable<string> Errors { get; set; }
7
8     1 reference | 0 changes | 0 authors, 0 changes
9     public ValidationFailureResponse(
10        IEnumerable<string> errorMessages)
11    {
12        Errors = errorMessages;
13    }
14 }
```

```
14 references | 0 changes | 0 authors, 0 changes
5  public class PostDto
6  {
7      // Mã bài viết
8      0 references | 0 changes | 0 authors, 0 changes
9      public int Id { get; set; }
10
11     // Tiêu đề bài viết
12     0 references | 0 changes | 0 authors, 0 changes
13     public string Title { get; set; }
14
15     // Mô tả hay giới thiệu ngắn về nội dung
16     0 references | 0 changes | 0 authors, 0 changes
17     public string ShortDescription { get; set; }
18
19     // Tên định danh để tạo URL
20     0 references | 0 changes | 0 authors, 0 changes
21     public string UrlSlug { get; set; }
22
23     // Đường dẫn đến tập tin hình ảnh
24     0 references | 0 changes | 0 authors, 0 changes
25     public string ImageUrl { get; set; }
26
27     // Số lượt xem, đọc bài viết
28     0 references | 0 changes | 0 authors, 0 changes
29     public int ViewCount { get; set; }
30
31     // Ngày giờ đăng bài
32     0 references | 0 changes | 0 authors, 0 changes
33     public DateTime PostedDate { get; set; }
34
35     // Ngày giờ cập nhật lần cuối
36     0 references | 0 changes | 0 authors, 0 changes
37     public DateTime? ModifiedDate { get; set; }
38
39 }
40
41     5 references | 0 changes | 0 authors, 0 changes
5  public class PostDetail
6  {
7      // Mã bài viết
8      0 references | 0 changes | 0 authors, 0 changes
9      public int Id { get; set; }
10
11     // Tiêu đề bài viết
12     0 references | 0 changes | 0 authors, 0 changes
13     public string Title { get; set; }
```

```
// Mô tả hay giới thiệu ngắn về nội dung
0 references | 0 changes | 0 authors, 0 changes
public string ShortDescription { get; set; }

// Nội dung chi tiết của bài viết
0 references | 0 changes | 0 authors, 0 changes
public string Description { get; set; }

// Metadata
0 references | 0 changes | 0 authors, 0 changes
public string Meta { get; set; }

// Tên định danh để tạo URL
0 references | 0 changes | 0 authors, 0 changes
public string UrlSlug { get; set; }

// Đường dẫn đến tập tin hình ảnh
0 references | 0 changes | 0 authors, 0 changes
public string ImageUrl { get; set; }

// Số lượt xem, đọc bài viết
0 references | 0 changes | 0 authors, 0 changes
public int ViewCount { get; set; }

// Ngày giờ đăng bài
0 references | 0 changes | 0 authors, 0 changes
public DateTime PostedDate { get; set; }

// Ngày giờ cập nhật lần cuối
0 references | 0 changes | 0 authors, 0 changes
public DateTime? ModifiedDate { get; set; }

// Chuyên mục của bài viết
0 references | 0 changes | 0 authors, 0 changes
public CategoryDto Category { get; set; }

// Tác giả của bài viết
0 references | 0 changes | 0 authors, 0 changes
public AuthorDto Author { get; set; }

// Danh sách các từ khóa của bài viết
0 references | 0 changes | 0 authors, 0 changes
public IList<TagDto> Tags { get; set; }
```

```
21 references | 0 changes | 0 authors, 0 changes
6  public class ApiResponse
7  {
8      0 references | 0 changes | 0 authors, 0 changes
9          public bool IsSuccess => Errors.Count == 0;
10     4 references | 0 changes | 0 authors, 0 changes
11     public HttpStatusCode StatusCode { get; init; }
12     4 references | 0 changes | 0 authors, 0 changes
13     public IList<string> Errors { get; init; }
14     1 reference | 0 changes | 0 authors, 0 changes
15     protected ApiResponse()
16     {
17     }
```

```
1 reference | 0 changes | 0 authors, 0 changes
14  protected ApiResponse()
15  {
16      StatusCode = HttpStatusCode.OK;
17      Errors = new List<string>();
18  }
19
20  7 references | 0 changes | 0 authors, 0 changes
21  public static ApiResponse<T> Success<T>(
22      T result,
23      HttpStatusCode statusCode = HttpStatusCode.OK)
24  {
25      return new ApiResponse<T>
26      {
27          Result = result,
28          StatusCode = statusCode
29      };
30
31  0 references | 0 changes | 0 authors, 0 changes
32  public static ApiResponse<T> FailWithResult<T>(
33      HttpStatusCode statusCode,
34      T result,
35      params string[] errorMessages)
36  {
37      return new ApiResponse<T>()
38      {
39          Result = result,
40          StatusCode = statusCode,
41          Errors = new List<string>(errorMessages)
42      };
43
44
45  0 references | 0 changes | 0 authors, 0 changes
46  public static ApiResponse<T> FailWithResult<T>(
47      HttpStatusCode statusCode,
48      T result,
49      params string[] errorMessages)
50  {
51      return new ApiResponse<T>()
52      {
53          Result = result,
54          StatusCode = statusCode,
55          Errors = new List<string>(errorMessages)
56      };
57
58
59  8 references | 0 changes | 0 authors, 0 changes
60  public static ApiResponse Fail(
61      HttpStatusCode statusCode,
62      params string[] errorMessages)
63  {
64
```

```

48     if (errorMessages == null || errorMessages.Length == 0)
49     {
50         throw new ArgumentNullException(nameof(errorMessages));
51     }
52
53     return new ApiResponse()
54     {
55         StatusCode = statusCode,
56         Errors = new List<string>(errorMessages)
57     };
58
59
60     2 references | 0 changes | 0 authors, 0 changes
61     public static ApiResponse Fail(
62         HttpStatusCode statusCode,
63         ValidationResult validationResult)
64     {
65         return Fail(statusCode, validationResult.Errors
66             .Select(x => x.ErrorMessage)
67             .Where(e => !string.IsNullOrWhiteSpace(e))
68             .ToArray());
69     }
70
71     11 references | 0 changes | 0 authors, 0 changes
72     public class ApiResponse<T> : ApiResponse
73     {
74         2 references | 0 changes | 0 authors, 0 changes
75         public T Result { get; set; }
76     }

```

Trong thư mục **TatBlog.WebApi > Extensions**, tạo thêm lớp tĩnh **ValidationResultExtensions** để định nghĩa các phương thức mở rộng như sau;

```

6     0 references | 0 changes | 0 authors, 0 changes
7     public static class ValidationResultExtensions
8     {
8
9         0 references | 0 changes | 0 authors, 0 changes
10        public static ValidationFailureResponse ToResponse(
11            this ValidationResult validationResult)
12        {
13            return validationResult.Errors.ToResponse();
14        }
15
16
17         1 reference | 0 changes | 0 authors, 0 changes
18        public static ValidationFailureResponse ToResponse(
19            this IEnumerable<ValidationFailure> failures)
20
21        {
22            return new ValidationFailureResponse(
23                failures.Select(e => e.ErrorMessage));
24        }
25
26
27         0 references | 0 changes | 0 authors, 0 changes
28        public static IList<string> GetErrorMessages(
29            this ValidationResult validationResult)
30    }

```

```

23     {
24         return validationResult.Errors.GetErrorMessages();
25     }
26
27     1 reference | 0 changes | 0 authors, 0 changes
28     public static IList<string> GetErrorMessages(
29         this IEnumerable<ValidationFailure> failures)
30     {
31         return failures.Select(e => e.ErrorMessage).ToList();
32     }
33
34     0 references | 0 changes | 0 authors, 0 changes
35     public static IDictionary<string, List<string>> GetErrorsWithPropertyNames(
36         this ValidationResult validationResult)
37     {
38         return validationResult.Errors.GetErrorsWithPropertyNames();
39     }
40
41     1 reference | 0 changes | 0 authors, 0 changes
42     public static IDictionary<string, List<string>> GetErrorsWithPropertyNames(
43         this IEnumerable<ValidationFailure> failures)
44     {
45         return failures
46             .GroupBy(e => e.PropertyName)
47             .ToDictionary(
48                 g => g.Key,
49                 g => g.Select(e => e.ErrorMessage).ToList());
50     }

```

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có).

## 5. Cấu hình Mapster và định nghĩa cách sao chép dữ liệu

Trong thư mục **TatBlog.WebApi > Mapsters**, tạo một lớp mới, đặt tên là **MapsterConfiguration**. Cài đặt mã nguồn để quy định cách sao chép dữ liệu giữa các đối tượng như sau.

```

1 reference | 0 changes | 0 authors, 0 changes
8  public class MapsterConfiguration : IRegister
9  {
10     0 references | 0 changes | 0 authors, 0 changes
11     public void Register(TypeAdapterConfig config)
12     {
13         config.NewConfig<Author, AuthorDto>();
14         config.NewConfig<Author, AuthorItem>()
15             .Map(dest => dest.PostCount,
16                  src => src.Posts == null ? 0 : src.Posts.Count);
17
18         config.NewConfig<AuthorEditModel, Author>();

```

```

18
19     config.NewConfig<Category, CategoryDto>();
20     config.NewConfig<Category, CategoryItem>()
21         .Map(dest => dest.PostCount,
22             src => src.Posts == null ? 0 : src.Posts.Count);
23
24     config.NewConfig<Post, PostDto>();
25     config.NewConfig<Post, PostDetail>();
26 }
27 }
```

Tiếp theo, tạo lớp **MapsterDependencyInjection** và định nghĩa hàm để đăng ký Mapster với DI Container.

```

0 references | 0 changes | 0 authors, 0 changes
6 public static class MapsterDependencyInjection
7 {
8     1 reference | 0 changes | 0 authors, 0 changes
9     public static WebApplicationBuilder ConfigureMapster(
10         this WebApplicationBuilder builder)
11     {
12         var config = TypeAdapterConfig.GlobalSettings;
13         config.Scan(typeof(MapsterConfiguration).Assembly);
14
15         builder.Services.AddSingleton(config);
16         builder.Services.AddScoped<IMapper, ServiceMapper>();
17
18         return builder;
19     }
}
```

Cập nhật mã nguồn trong tập tin **Program.cs** để gọi hàm vừa tạo.

```

4 var builder = WebApplication.CreateBuilder(args);
5 {
6     // Add services to the container.
7     builder
8         .ConfigureCors()
9         .ConfigureNLog()
10        .ConfigureServices()
11        .ConfigureSwaggerOpenApi()
12        .ConfigureMapster(); ←
13 }
```

## 6. Sử dụng FluentValidation để kiểm tra tính hợp lệ của dữ liệu

Trong phần này, ta sẽ sử dụng gói thư viện **FluentValidation** để kiểm tra dữ liệu được gửi lên từ phía client. Sinh viên tìm hiểu thêm về cách sử dụng thư viện này tại: <https://docs.fluentvalidation.net/en/latest/>.

Trong thư mục **TatBlog.WebApi > Validations**, tạo hai lớp **AuthorValidator** và **FluentValidationDependencyInjection**.

Trong lớp **AuthorValidator**, cài đặt các quy tắc kiểm tra dữ liệu nhập về thông tin tác giả.

```

1 reference | 0 changes | 0 authors, 0 changes
6 public class AuthorValidator : AbstractValidator<AuthorEditModel>
7 {
8     0 references | 0 changes | 0 authors, 0 changes
9     public AuthorValidator()
10    {
11        RuleFor(a => a.FullName)
12            .NotEmpty()
13            .WithMessage("Tên tác giả không được để trống")
14            .MaximumLength(100)
15            .WithMessage("Tên tác giả tối đa 100 ký tự");
16
17        RuleFor(a => a.UrlSlug)
18            .NotEmpty()
19            .WithMessage("UrlSlug không được để trống")
20            .MaximumLength(100)
21            .WithMessage("UrlSlug tối đa 100 ký tự");
22
23        RuleFor(a => a.JoinedDate)
24            .GreaterThan(DateTime.MinValue)
25            .WithMessage("Ngày tham gia không hợp lệ");
26
27        RuleFor(a => a.Email)
28            .NotEmpty()
29            .WithMessage("Email không được để trống")
30            .MaximumLength(100)
31            .WithMessage("Email chứa tối đa 100 ký tự");
32
33        RuleFor(a => a.Notes)
34            .MaximumLength(500)
35            .WithMessage("Ghi chú tối đa 500 ký tự");
36    }
}

```

Sau đó, cài đặt lớp **FluentValidation.DependencyInjection** để đăng ký các dịch vụ của **FluentValidation** với DI Container.

```

0 references | 0 changes | 0 authors, 0 changes
7 public static class FluentValidationDependencyInjection
8 {
9     0 references | 0 changes | 0 authors, 0 changes
10    public static WebApplicationBuilder ConfigureFluentValidation(
11        this WebApplicationBuilder builder)
12    {
13        // Scan and register all validators in given assembly
14        builder.Services.AddValidatorsFromAssembly(
15            Assembly.GetExecutingAssembly());
16
17        return builder;
18    }
}

```

Gọi phương thức vừa tạo trong lớp **Program**.

```

5  var builder = WebApplication.CreateBuilder(args);
6  {
7      // Add services to the container.
8      builder
9          .ConfigureCors()
10         .ConfigureNLog()
11         .ConfigureServices()
12         .ConfigureSwaggerOpenApi()
13         .ConfigureMapster()
14         .ConfigureFluentValidation(); ←
15 }
```

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có).

## 7. Định nghĩa các endpoints

Phần này hướng dẫn cách định nghĩa chi tiết các endpoint đã được liệt kê ở đầu bài Lab.

Trong thư mục **TatBlog.WebApi > Endpoints**, tạo lớp tĩnh **AuthorEndpoints**. Sau đó, định nghĩa một phương thức mở rộng **MapAuthorEndpoints**.

```

0 references | 0 changes | 0 authors, 0 changes
16 public static class AuthorEndpoints
17 {
18     public static WebApplication MapAuthorEndpoints(
19         this WebApplication app)
20     {
21         return app;
22     }
23 }
```

Cập nhật mã nguồn trong lớp **Program** để gọi tới phương thức vừa tạo

```

18 var app = builder.Build();
19 {
20     // Configure the HTTP request pipeline.
21     app.SetupRequestPipeline();
22
23     // Configure API endpoints
24     app.MapAuthorEndpoints(); ←
25
26     app.Run();
27 }
```

Quay trở lại lớp **AuthorEndpoints**, định nghĩa phương thức **GetAuthors** để xử lý yêu cầu tìm và lấy danh sách sách tác giả.

```

1 reference | 0 changes | 0 authors, 0 changes
81  private static async Task<IResult> GetAuthors(
82      [AsParameters] AuthorFilterModel model,
83      IAuthorRepository authorRepository)
84  {
85      var authorsList = await authorRepository
86          .GetPagedAuthorsAsync(model, model.Name);
87
88      var paginationResult =
89          new PaginationResult<AuthorItem>(authorsList);
90
91      return Results.Ok(paginationResult);
92 }

```

Sau đó, cập nhật phương thức **MapAuthorEndpoints** để định nghĩa API endpoint đầu tiên.

```

1 reference | 0 changes | 0 authors, 0 changes
18  public static WebApplication MapAuthorEndpoints(
19      this WebApplication app)
20  {
21      var routeGroupBuilder = app.MapGroup("/api/authors");
22
23      routeGroupBuilder.MapGet("/", GetAuthors)
24          .WithName("GetAuthors")
25          .Produces<PaginationResult<AuthorItem>>();
26
27      return app;
28  }

```

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có). Chạy chương trình để xem kết quả. Nếu không có lỗi gì khác, trình duyệt sẽ hiển thị kết quả như sau.

Nhấn chuột vào nút mũi tên xuống ở bên phải dòng **GET /api/authors** để xem chi tiết các tham số và định dạng kết quả trả về khi gọi tới API Endpoint này.

The screenshot shows the **GET /api/authors** endpoint in a Swagger UI interface. The **Parameters** section displays five query parameters:

- Name**: string (query)
- PageSize** \* required: integer(\$int32) (query)
- PageNumber** \* required: integer(\$int32) (query)
- SortColumn**: string (query)
- SortOrder**: string (query)

The **Responses** section shows the **200** Success response with **application/json** media type. The response body schema is as follows:

```
{  "items": [    {      "id": 0,      "fullName": "string",      "urlSlug": "string",      "imageUrl": "string",      "joinedDate": "2023-03-21T11:27:12.248Z",      "email": "string",      "notes": "string",      "postCount": 0    }  ],  "metadata": {    "pageIndex": 0,    "pageSize": 0,    "totalItemCount": 0,    "pageNumber": 0,    "pageCount": 0,    "hasPreviousPage": true,    "hasNextPage": true,    "firstItemIndex": 0,    "lastItemIndex": 0,    "isFirstPage": true,    "isLastPage": true  }}
```

Để kiểm tra hoạt động của API endpoint, nhấn chuột vào nút **Try it out**, nhập vào giá trị 10 cho PageSize, 1 cho PageNumber (có thể nhập thêm từ khóa vào ô Name). Sau đó nhấn nút **Execute** để gửi HTTP Request tới API endpoint **/api/authors**.

GET /api/authors ^

Parameters Cancel

Name	Description
Name string (query)	<input type="text" value="Name"/>
PageSize * required integer(\$int32) (query)	<input type="text" value="10"/>
PageNumber * required integer(\$int32) (query)	<input type="text" value="1"/>
SortColumn string (query)	<input type="text" value="SortColumn"/>
SortOrder string (query)	<input type="text" value="SortOrder"/>

Execute

Kết quả trả về tương tự như sau.

Server response

Code	Details
200	Response body

```
{  
  "items": [  
    {  
      "id": 3,  
      "fullName": "Kathy Smith",  
      "urlSlug": "kathy-smith",  
      "imageUrl": null,  
      "joinedDate": "2010-06-09T00:00:00",  
      "email": "kathy.smith@iworld.com",  
      "notes": null,  
      "postCount": 4  
    },  
    {  
      "id": 2,  
      "fullName": "Jessica Wonder",  
      "urlSlug": "jessica-wonder",  
      "imageUrl": null,  
      "joinedDate": "2020-04-19T00:00:00",  
      "email": "jessica665@motip.com",  
      "notes": null,  
      "postCount": 1  
    }  
  ]  
}
```

```

    "notes": null,
    "postCount": 6
  },
],
"metadata": {
  "pageIndex": 0,
  "pageSize": 10,
  "totalItemCount": 3,
  "pageNumber": 1,
  "pageCount": 1,
  "hasPreviousPage": false,
  "hasNextPage": false,
  "firstItemIndex": 1,
  "lastItemIndex": 3,
  "isFirstPage": true,
  "isLastPage": true
}
}

```

Response headers

```

content-type: application/json; charset=utf-8
date: Tue, 21 Mar 2023 11:32:57 GMT
server: Kestrel

```

Tương tự, trong lớp **AuthorEndpoints**, tiếp tục định nghĩa các phương thức sau

```

1 reference | 0 changes | 0 authors, 0 changes
78  private static async Task<IResult> GetAuthorDetails(
79    int id,
80    IAuthorRepository authorRepository,
81    IMapper mapper)
82  {
83    var author = await authorRepository.GetCachedAuthorByIdAsync(id);
84    return author == null
85      ? Results.NotFound($"Không tìm thấy tác giả có mã số {id}")
86      : Results.Ok(mapper.Map<AuthorItem>(author));
87  }
88

0 references | 0 changes | 0 authors, 0 changes
89  private static async Task<IResult> GetPostsByAuthorId(
90    int id,
91    [AsParameters] PagingModel pagingModel,
92    IBlogRepository blogRepository)
93  {
94    var postQuery = new PostQuery()
95    {
96      AuthorId = id,
97      PublishedOnly = true
98    };
99

100   var postsList = await blogRepository.GetPagedPostsAsync(
101     postQuery, pagingModel,
102     posts => posts.ProjectToType<PostDto>());
103

104   var paginationResult = new PaginationResult<PostDto>(postsList);
105

106   return Results.Ok(paginationResult);
107 }

```

```
1 reference | 0 changes | 0 authors, 0 changes
109  private static async Task<IResult> GetPostsByAuthorSlug(
110    [FromRoute] string slug,
111    [AsParameters] PagingModel pagingModel,
112    IBlogRepository blogRepository)
113  {
114    var postQuery = new PostQuery()
115    {
116      AuthorSlug = slug,
117      PublishedOnly = true
118    };
119
120    var postsList = await blogRepository.GetPagedPostsAsync(
121      postQuery, pagingModel,
122      posts => posts.ProjectToType<PostDto>());
123    var paginationResult = new PaginationResult<PostDto>(postsList);
124
125    return Results.Ok(paginationResult);
126  }
127
1 reference | 0 changes | 0 authors, 0 changes
128  private static async Task<IResult> AddAuthor(
129    AuthorEditModel model,
130    IValidator<AuthorEditModel> validator,
131    IAuthorRepository authorRepository,
132    IMapper mapper)
133  {
134    var validationResult = await validator.ValidateAsync(model);
135
136    if (!validationResult.IsValid)
137    {
138      return Results.BadRequest(
139        validationResult.Errors.ToResponse());
140    }
141
142    if (await authorRepository
143      .IsAuthorSlugExistedAsync(0, model.UrlSlug))
144    {
145      return Results.Conflict(
146        $"Slug '{model.UrlSlug}' đã được sử dụng");
147    }
148
149    var author = mapper.Map<Author>(model);
150    await authorRepository.AddOrUpdateAsync(author);
151
152    return Results.CreatedAtRoute(
153      "GetAuthorById", new {author.Id},
154      mapper.Map<AuthorItem>(author));
155  }
156
1 reference | 0 changes | 0 authors, 0 changes
157  private static async Task<IResult> SetAuthorPicture(
158    int id, IFormFile imageFile,
159    IAuthorRepository authorRepository,
160    IMediaManager mediaManager)
161  {
```

```
162     var imageUrl = await mediaManager.SaveFileAsync(
163         imageFile.OpenReadStream(),
164         imageFile.FileName, imageFile.ContentType);
165 
166     if (string.IsNullOrWhiteSpace(imageUrl))
167     {
168         return Results.BadRequest("Không lưu được tập tin");
169     }
170 
171     await authorRepository.SetImageUrlAsync(id, imageUrl);
172     return Results.Ok(imageUrl);
173 }
174 
1 reference | 0 changes | 0 authors, 0 changes
175 private static async Task<IResult> UpdateAuthor(
176     int id, AuthorEditModel model,
177     IValidator<AuthorEditModel> validator,
178     IAuthorRepository authorRepository,
179     IMapper mapper)
180 {
181     var validationResult = await validator.ValidateAsync(model);
182 
183     if (!validationResult.IsValid)
184     {
185         return Results.BadRequest(
186             validationResult.Errors.ToResponse());
187     }
188 
189     if (await authorRepository
190         .IsAuthorSlugExistedAsync(id, model.UrlSlug))
191     {
192         return Results.Conflict(
193             $"Slug '{model.UrlSlug}' đã được sử dụng");
194     }
195 
196     var author = mapper.Map<Author>(model);
197     author.Id = id;
198 
199     return await authorRepository.AddOrUpdateAsync(author)
200         ? Results.NoContent()
201         : Results.NotFound();
202 }
203 
204 1 reference | 0 changes | 0 authors, 0 changes
205 private static async Task<IResult> DeleteAuthor(
206     int id, IAuthorRepository authorRepository)
207 {
208     return await authorRepository.DeleteAuthorAsync(id)
209         ? Results.NoContent()
210         : Results.NotFound($"Could not find author with id = {id}");
```

Cập nhật lại phương thức **MapAuthorEndpoints** để định nghĩa đầy đủ các API endpoint cho phần quản lý thông tin tác giả.

```
1 reference | 0 changes | 0 authors, 0 changes
19   public static WebApplication MapAuthorEndpoints(
20     this WebApplication app)
21   {
22     var routeGroupBuilder = app.MapGroup("/api/authors");
23
24     routeGroupBuilder.MapGet("/", GetAuthors)
25       .WithName("GetAuthors")
26       .Produces<PaginationResult<AuthorItem>>();
27
28     routeGroupBuilder.MapGet("/{id:int}", GetAuthorDetails)
29       .WithName("GetAuthorById")
30       .Produces<AuthorItem>()
31       .Produces(404);
32
33     routeGroupBuilder.MapGet(
34       "/{slug:regex(^[a-z0-9_-]+$)}/posts",
35       GetPostsByAuthorSlug)
36       .WithName("GetPostsByAuthorSlug")
37       .Produces<PaginationResult<PostDto>>();
38
39     routeGroupBuilder.MapPost("/", AddAuthor)
40       .WithName("AddNewAuthor")
41       .Produces(201)
42       .Produces(400)
43       .Produces(409);
44
45     routeGroupBuilder.MapPost("/{id:int}/avatar", SetAuthorPicture)
46       .WithName("SetAuthorPicture")
47       .Accepts<IFormFile>("multipart/form-data")
48       .Produces<string>()
49       .Produces(400);
50
51     routeGroupBuilder.MapPut("/{id:int}", UpdateAuthor)
52       .WithName("UpdateAnAuthor")
53       .Produces(204)
54       .Produces(400)
55       .Produces(409);
56
57     routeGroupBuilder.MapDelete("/{id:int}", DeleteAuthor)
58       .WithName("DeleteAnAuthor")
59       .Produces(204)
60       .Produces(404);
61
62     return app;
63 }
```

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có). Chạy chương trình và sử dụng chức năng **Try it out** để kiểm tra hoạt động của các API endpoint.

## 8. Định nghĩa EndpointFilter và áp dụng cho việc kiểm tra dữ liệu

Để ý rằng, trong hai phương thức **AddAuthor** và **UpdateAuthor** ở phần trước sử dụng đối tượng **IValidator<AuthorEditModel>** **validator** được tạo bởi DI Container để kiểm tra dữ liệu đầu vào được cung cấp bởi Client và việc kiểm tra này được thực hiện thủ công, lặp lại nhiều lần.

Ta có thể đơn giản hóa việc này bằng cách tạo đối tượng **EndpointFilter** và sử dụng nó để thực hiện việc kiểm tra dữ liệu nhập một cách tự động. Theo đó, mã nguồn có thể được sử dụng lại, giúp các phương thức **AddAuthor**, **UpdateAuthor** trông gọn gàng hơn.

```
3 references | 0 changes | 0 authors, 0 changes
8 public class ValidatorFilter<T> : IEndpointFilter where T : class
9 {
10     private readonly IValidator<T> _validator;
11
12     public ValidatorFilter(IValidator<T> validator)
13     {
14         _validator = validator;
15     }
16
17     public async ValueTask<object> InvokeAsync(
18         EndpointFilterInvocationContext context,
19         EndpointFilterDelegate next)
20     {
21         var model = context.Arguments
22             .SingleOrDefault(x => x?.GetType() == typeof(T)) as T;
23
24         if (model == null)
25         {
26             return Results.BadRequest(
27                 new ValidationFailureResponse(new []
28                 {
29                     "Could not create model object"
30                 }));
31         }
32
33         var validationResult = await _validator.ValidateAsync(model);
34
35         if (!validationResult.IsValid)
36         {
37             return Results.BadRequest(
38                 validationResult.Errors.ToResponse());
39         }
40
41         return await next(context);
42     }
43 }
```

Trong thư mục **TatBlog.WebApi > Filters**, tạo một lớp mới, đặt tên là **ValidatorFilter**. Cài đặt lớp này như hình trên.

Sau đó, cập nhật lại các phương thức **AddAuthor** và **UpdateAuthor** trong lớp **AuthorEndpoints** để loại bỏ phần kiểm tra dữ liệu đầu vào.

```
1 reference | 0 changes | 0 authors, 0 changes
130  private static async Task<IResult> AddAuthor(
131    AuthorEditModel model,
132    IAuthorRepository authorRepository,
133    IMapper mapper)
134  {
135    if (await authorRepository
136      .IsAuthorSlugExistedAsync(0, model.UrlSlug))
137    {
138      return Results.Conflict(
139        $"Slug '{model.UrlSlug}' đã được sử dụng");
140    }
141
142    var author = mapper.Map<Author>(model);
143    await authorRepository.AddOrUpdateAsync(author);
144
145    return Results.CreatedAtRoute(
146      "GetAuthorById", new {author.Id},
147      mapper.Map<AuthorItem>(author));
148  }

149
1 reference | 0 changes | 0 authors, 0 changes
150  private static async Task<IResult> UpdateAuthor(
151    int id, AuthorEditModel model,
152    IAuthorRepository authorRepository,
153    IMapper mapper)
154  {
155    if (await authorRepository
156      .IsAuthorSlugExistedAsync(id, model.UrlSlug))
157    {
158      return Results.Conflict(
159        $"Slug '{model.UrlSlug}' đã được sử dụng");
160    }
161
162    var author = mapper.Map<Author>(model);
163    author.Id = id;
164
165    return await authorRepository.AddOrUpdateAsync(author)
166      ? Results.NoContent()
167      : Results.NotFound();
168 }
```

Để sử dụng lớp **ValidatorFilter** vừa tạo, ta cập nhật lại phần định nghĩa các API endpoint sau đây trong phương thức **MapAuthorEndpoints**.

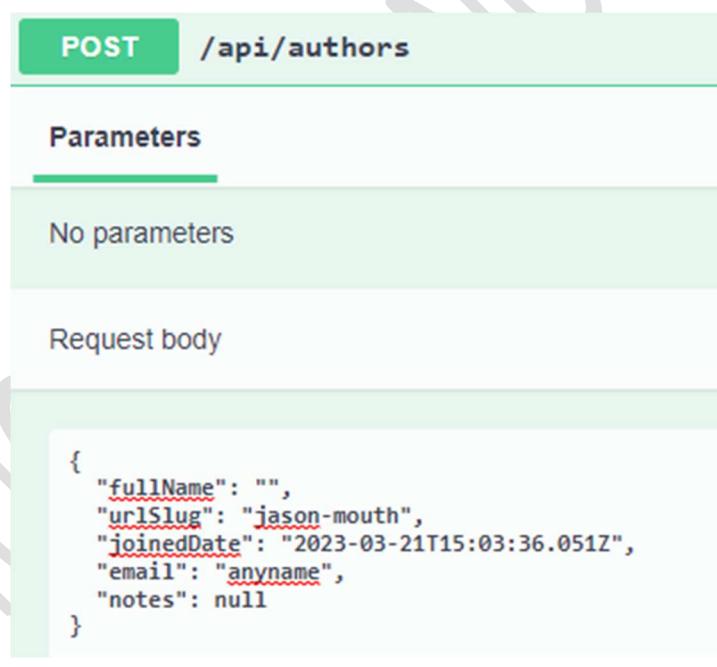
```

39   routeGroupBuilder.MapPost("/", AddAuthor)
40     .WithName("AddNewAuthor")
41     .AddEndpointFilter<ValidatorFilter<AuthorEditModel>>() ←
42     .Produces(201)
43     .Produces(400)
44     .Produces(409);
45
46   routeGroupBuilder.MapPut("/{id:int}", UpdateAuthor)
47     .WithName("UpdateAnAuthor")
48     .AddEndpointFilter<ValidatorFilter<AuthorEditModel>>() ←
49     .Produces(204)
50     .Produces(400)
51     .Produces(409);

```

Biên dịch chương trình, kiểm tra và sửa lỗi (nếu có). Chạy chương trình và sử dụng chức năng **Try it out** để kiểm tra hoạt động của 2 API endpoint **POST /api/authors, PUT /api/authors/{id}**.

Trong phần **Request body**, hãy thử nhập vào dữ liệu không hợp lệ, nhấn nút **Execute** để xem kết quả trả về là gì.



### Vận dụng:

- Hãy định nghĩa API endpoint **GET /api/authors/best/{limit:int}** trả về danh sách N (limit) tác giả có nhiều bài viết đóng góp cho blog nhất.
- Hãy bổ sung mã lệnh sử dụng đối tượng **ILogger** để ghi lại nhật ký hệ thống lên các tập tin (sử dụng gói thư viện **NLog**).

## C. Bài tập thực hành

### 1. Xây dựng các API endpoint để quản lý thông tin các chuyên mục.

HTTP Method	Endpoint	Giải thích
GET	/api/categories	Lấy danh sách chuyên mục. Hỗ trợ tìm theo tên và phân trang kết quả.
GET	/api/categories/{id}	Lấy thông tin chi tiết của chuyên mục có mã số (id) cho trước.
GET	/api/categories/{slug}/posts	Lấy danh sách những bài viết được đăng trong chuyên mục có tên định danh (slug) cho trước. Hỗ trợ việc phân trang danh sách bài viết.
POST	/api/categories	Thêm một chuyên mục mới.
PUT	/api/categories/{id}	Cập nhật thông tin của chuyên mục có mã số (id) cho trước.
DELETE	/api/categories/{id}	Xóa chuyên mục có mã số (id) cho trước

### 2. Xây dựng các API endpoint để quản lý thông tin bài viết.

HTTP Method	Endpoint	Giải thích
GET	/api/posts	Lấy danh sách bài viết. Hỗ trợ tìm theo từ khóa, chuyên mục, tác giả, ngày đăng, ... và phân trang kết quả.
GET	/api/posts/featured/{limit}	Lấy danh sách N (limit) bài viết được nhiều người đọc nhất.
GET	/api/posts/random/{limit}	Lấy ngẫu nhiên một danh sách N (limit) bài viết.
GET	/api/posts/archives/{limit}	Lấy danh sách thống kê số lượng bài viết trong N (limit) tháng gần nhất.
GET	/api/posts/{id}	Lấy thông tin chi tiết của bài viết có mã số (id) cho trước.
GET	/api/posts/byslug/{slug}	Lấy thông tin chi tiết bài viết có tên định danh (slug) cho trước.
POST	/api/posts	Thêm một bài viết mới.
POST	/api/posts/{id}/picture	Tải lên hình ảnh đại diện cho bài viết.
PUT	/api/posts/{id}	Cập nhật thông tin của bài viết có mã số (id) cho trước.
DELETE	/api/posts/{id}	Xóa bài viết có mã số (id) cho trước
GET	/api/posts/{id}/comments	Lấy danh sách bình luận của bài viết

3. Xây dựng các API endpoint để quản lý thẻ/từ khóa (tag).
4. Xây dựng các API endpoint để quản lý thông tin người theo dõi blog, đăng ký và hủy đăng ký theo dõi.
5. Xây dựng các API endpoint để quản lý các bình luận.
6. Xây dựng các API endpoint để xử lý việc nhận các góp ý từ người dùng (nhập vào form liên hệ).
7. Xây dựng các API endpoint trả về các chỉ số thống kê: Tổng số bài viết, số bài viết chưa xuất bản, số lượng chủ đề, số lượng tác giả, số lượng bình luận đang chờ phê duyệt, số lượng người theo dõi, số lượng người mới theo dõi đăng ký (lấy số liệu trong ngày).
8. Tìm hiểu cách sử dụng gói thư viện [Carter](#) để phân chia API endpoints vào các mô-đun và tự động tìm quét các endpoint.

--- HẾT ---