Submission Microservice Handbook

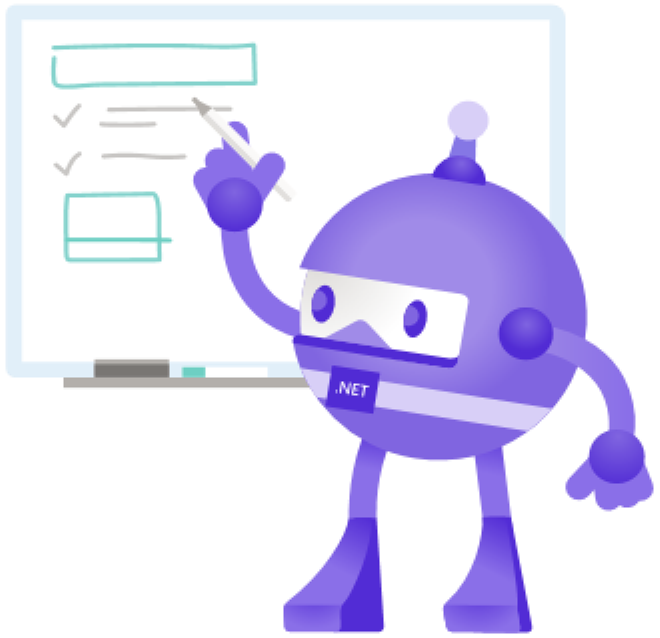
This handbook is a visual companion to the Submission Microservice module of the course.

It summarizes the architecture, design diagrams, and code examples covered in the lectures.

Use this document as a reference guide while following the hands-on videos.

All diagrams and visuals match the slides shown in the course for easier navigation.

Table of Contents

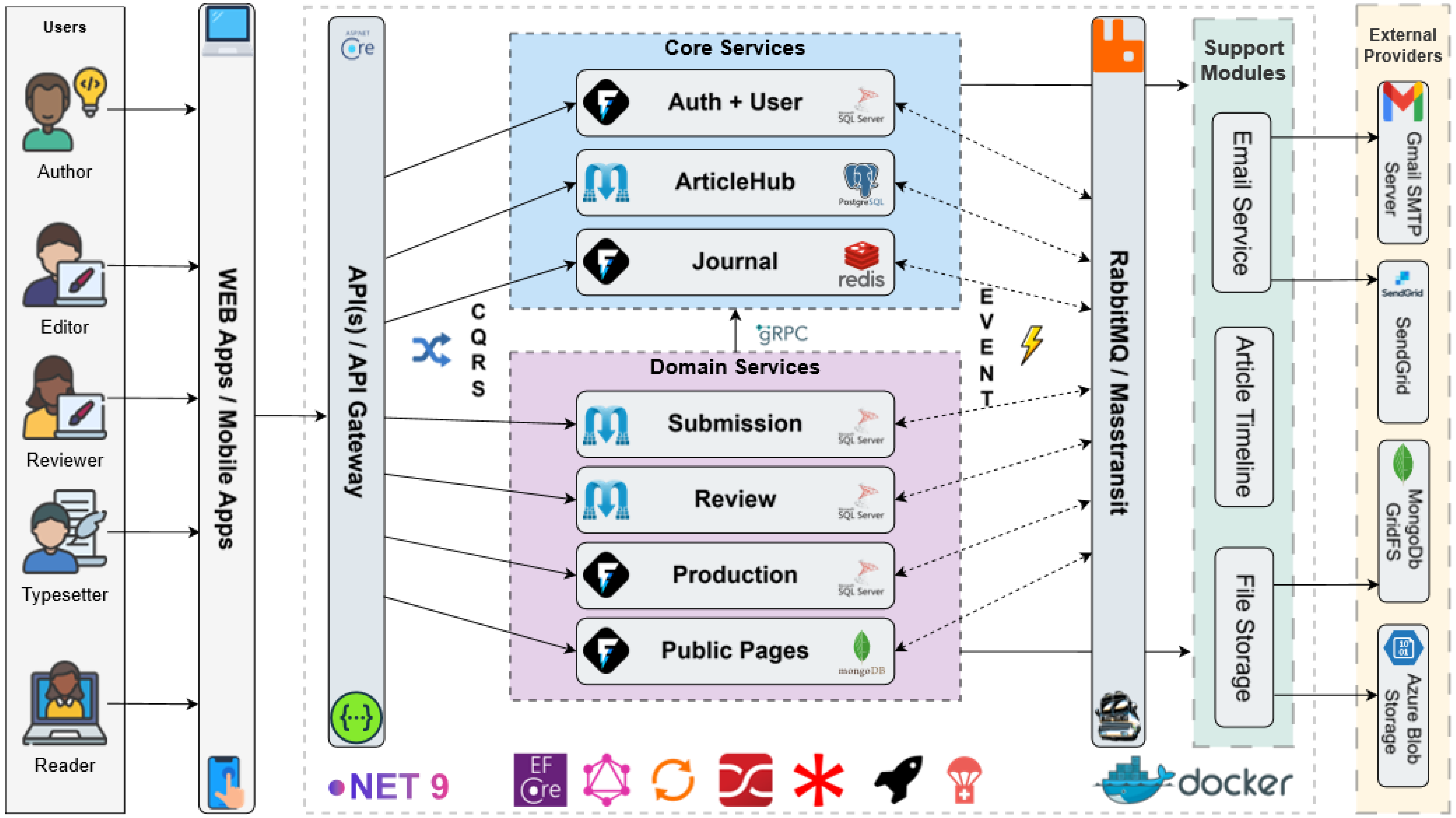


Submission Microservice

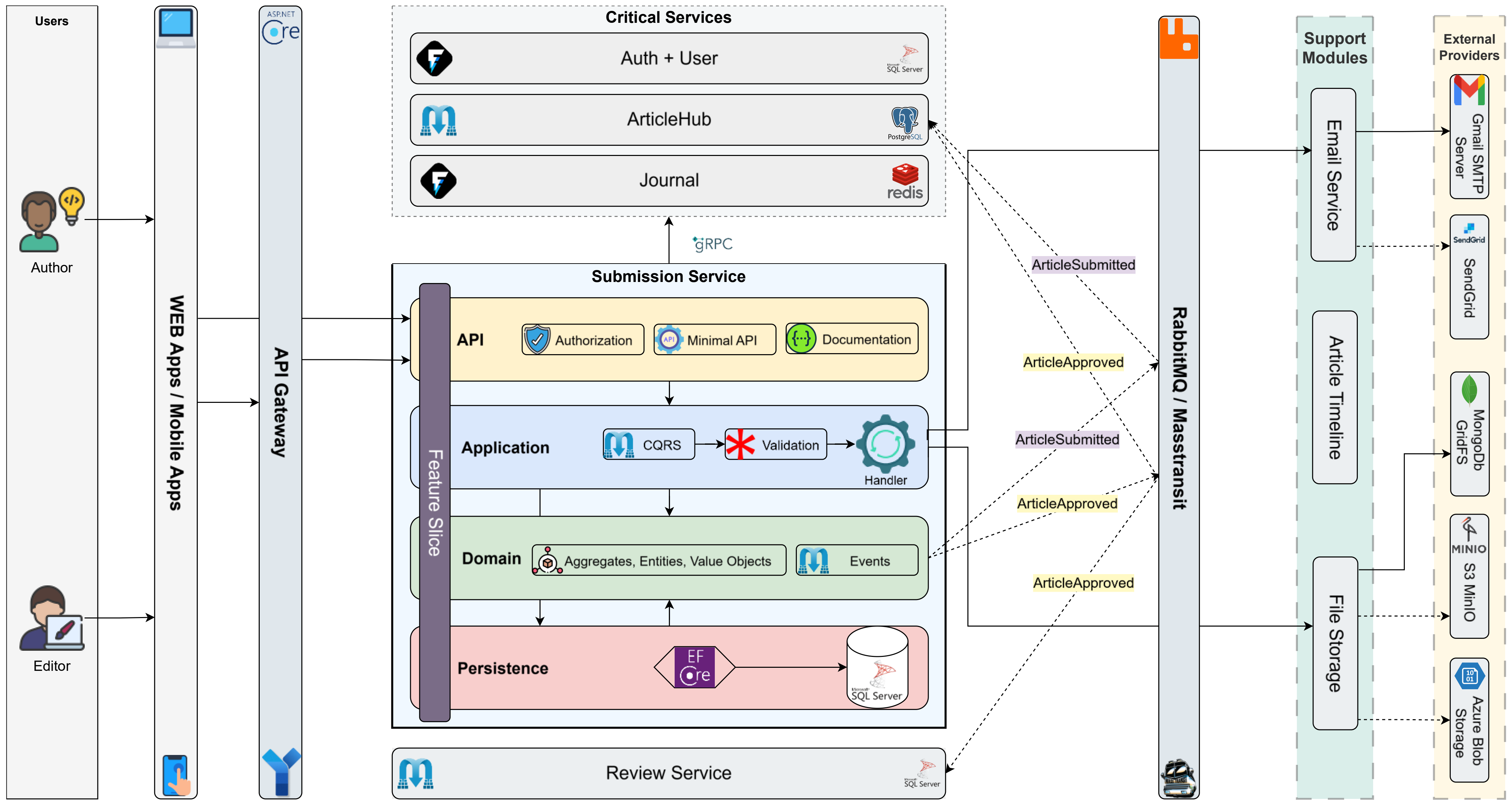
with MediatR, FluentValidation & EF Core

- Build **Minimal API Endpoints**
- Implement **CQRS with MediatR**
- Validate requests using **FluentValidation**
- Configure domain persistence with **EF Core**
- Upload & Download files via the **FileStorage Module**
- Transform **domain events** into integration events
- Publish integration events with **RabbitMQ** and **MassTransit**

High Level Architecture | C4 Level 2 (Container View)

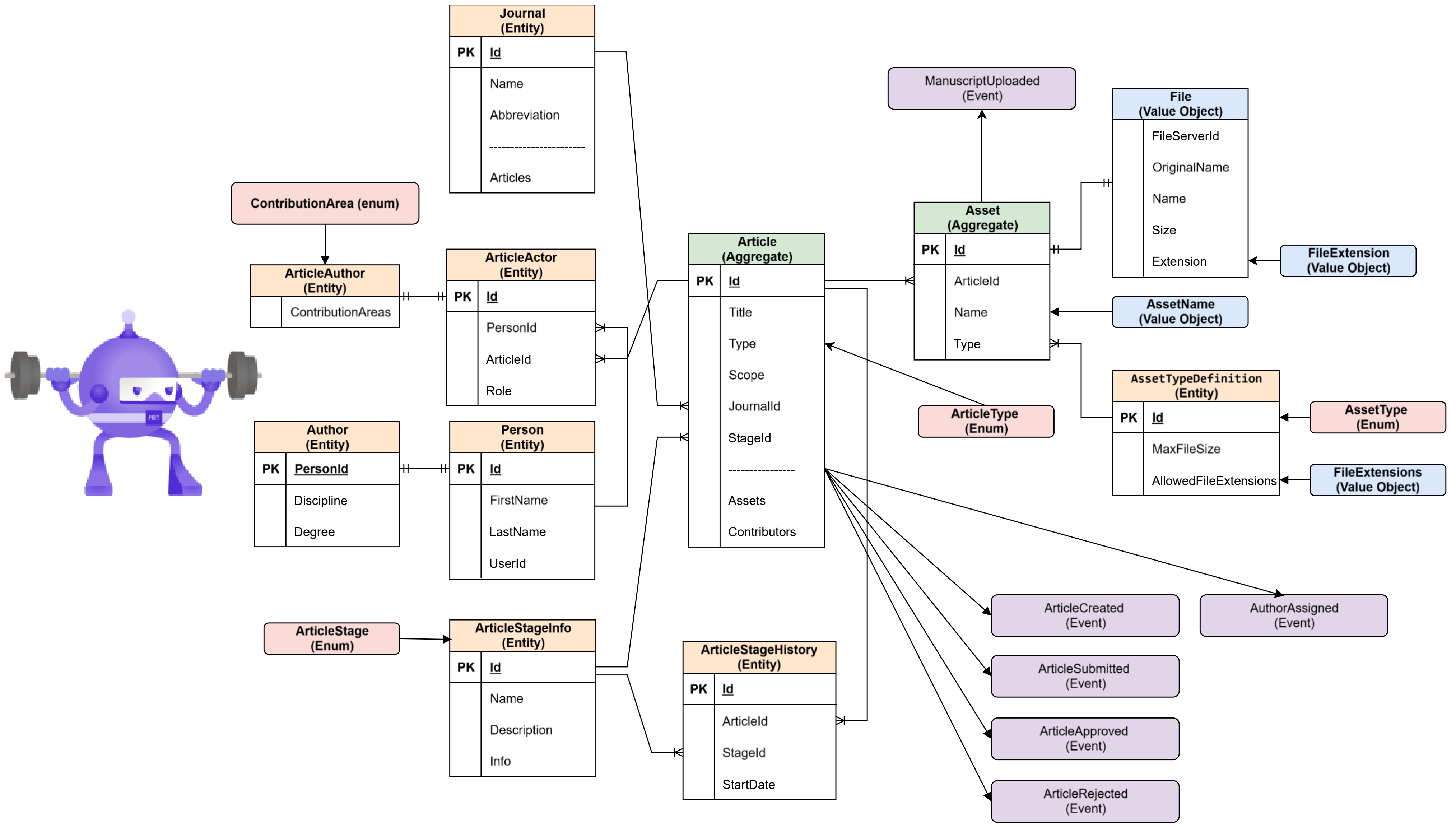


Submission Architecture | C4 Level 2 (Container View)



DotNet LabX

Tactical Design Diagram (DDD) | Level 4 (Class) C4 model



Legend

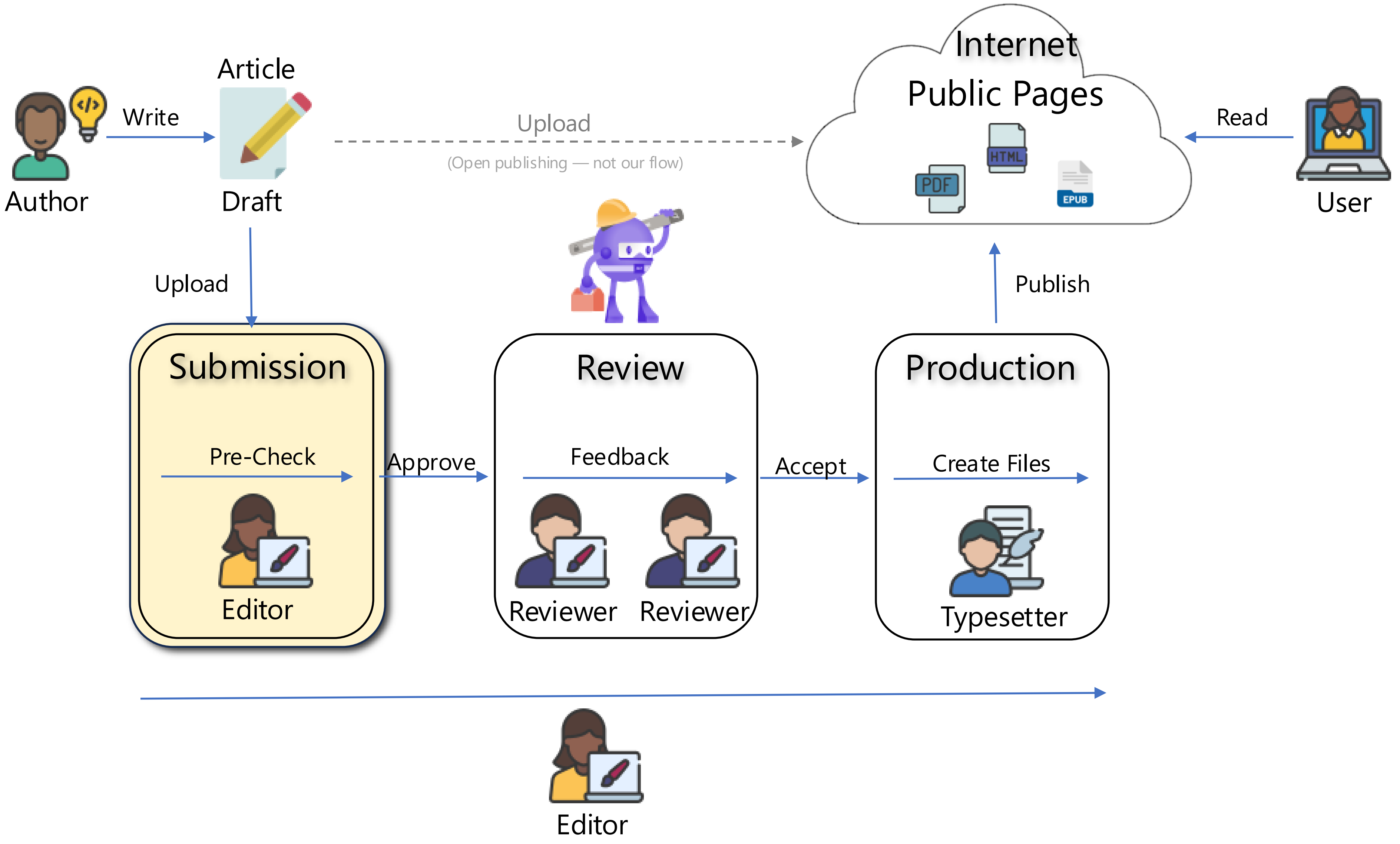
🟩	Aggregate
🟧	Entity
🟦	Value Object
🟥	Enum
🟪	Domain Event

PK = Primary Key

‖———‖	1 To 1
———<	1 To Many
———▶	Reference

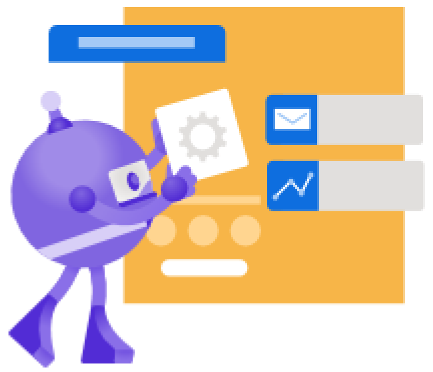
DotNet LabX

Article Workflow



User Stories

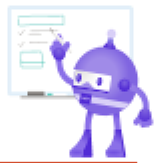
- **Create Article**
    - *As an* author, *I want* to create a new article *so that* I can start preparing a submission
- **Assign Author**
    - *As an* author, *I want* to assign co-authors to my article *so that* their contributions are properly recognized.
- **Create and Assign Author**
    - *As an* author, *I want* to create and assign a new co-author *so that* I can add collaborators who are not yet registered.
- **Upload Manuscript File**
    - *As an* author, *I want* to upload the manuscript file *so that* the core content of my article is available for review.
- **Upload Supplementary Materials**
    - *As an* author, *I want* to upload supplementary material *so that* additional resources can support my article.
- **Submit Article**
    - *As an* author, *I want to* submit my article *so that* it can enter the review process.
- **Approve Article**
    - *As an* editor, *I want* to approve a submitted article *so that* it can move forward in the workflow.
- **Reject Article**
    - *As a* submission editor, *I want* to reject a submitted article *so that* unqualified submissions can be filtered out.
- **Get Article**
    - *As an* author or editor, *I want* to view the details of an article *so that* I can review or take action depending on its stage.
- **Download File**
    - *As an* author or editor, *I want* to download uploaded files *so that* I can review the article content or attachments.



Endpoints

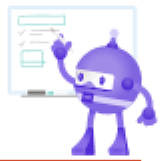
Name	Method	Roles	Endpoint
Create Article	POST	AUT	/api/articles
Assign Author	PUT	AUT	/api/articles/{articleId}/authors/{authorId}
Create and Assign Author	POST	AUT	/api/articles/{articleId}/authors
Upload Manuscript File	POST	AUT	/api/articles/{articleId}/files/manuscript
Upload Suppl. Material File	POST	AUT	/api/articles/{articleId}/files/supplementary-materials
Upload File	POST	AUT	/api/articles/{articleId}/files
Submit Article	PUT	AUT	/api/articles/{articleId}:submit
Approve Article	PUT	EDIT	/api/articles/{articleId}:approve
Reject Article	PUT	EDIT	/api/articles/{articleId}:reject
Get Article	GET	AUT, EDIT	/api/articles/{articleId}
Download File	GET	AUT, EDIT	/api/articles/{articleId}/files/{fileId}/content

Functional Requirements



- **Create Article**
  - What fields are required at creation time? → Title, Scope, Journal, Type, Stage, Audit fields
- **Assign Existing Author / Create and Assign New Author**
  - What fields are required for an Author? → FirstName, LastName, Title, Affiliation
  - Can multiple authors be assigned? → Yes
- **Upload Manuscript & Supplementary Files**
  - What file metadata is required? → FileName, Extension, Size
  - What other metadata is needed → Name, Type(Manuscript & SupplMaterial), State, Category, IsMandatory
  - What is the size limit and what extensions are allowed for each type?
    - Manuscript(pdf, doc) → 10MB
    - SupplMaterial(pdf, doc, jpg, png, tiff, mp3) → 5MB
  - Do we need to preview the files (show/play) → No
  - Can files be replaced or versioned? → No versioning, no replacement but they can be deleted.
  - What happens with the files after the article moves to the next Microservice? → they are kept for 2 years, then archived
- **Submit Article**
  - Can the article be edited after submission? → No. The article is locked.
- **Approve / Reject Article**
  - Do we need to include comments or reason for the decision? → Yes. Optional comments for all actions. Rejection requires a mandatory reason.
  - What happens to the article after acceptance? → It moves to the Review service and becomes locked in Submission. After 2 years, it is archived.
  - What happens to the article after rejection? → It is locked in Submission. After 2 years, it is archived.
- **Get Article**
  - Are authors limited to viewing only their own articles?→ Yes!
  - What level of detail is shown? → Everything the article and its children contain, except file content.
  - Can the article be viewed at any time, regardless of its stage? → Yes.
- **Download File**
  - Who can download files and when? → Same rules as "Get Article": authors (only their own articles) and editors can download at any stage.

Non-Functional Requirements

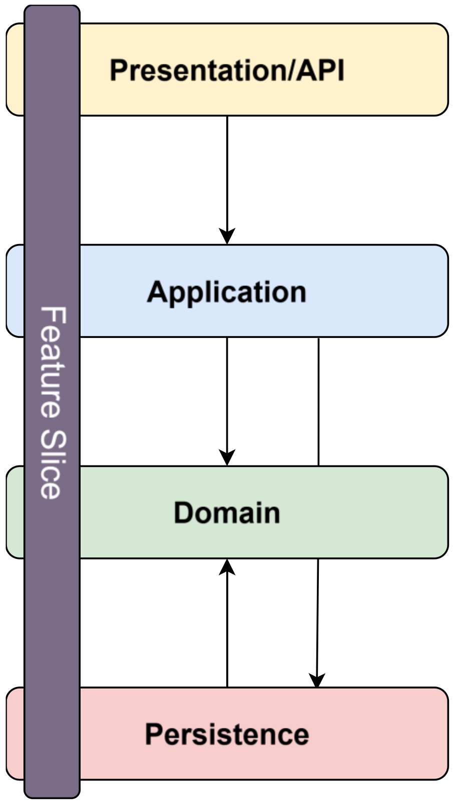


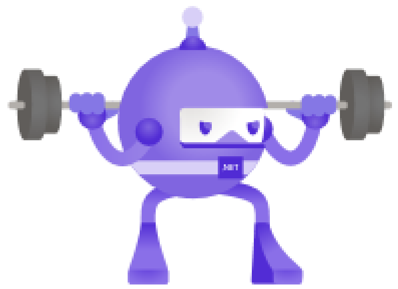
- The system supports 2 roles: Author & Editor
- Authors can only access their own articles and files
- Audit required for each action (Create, Submit, Approve, Reject)
- The system must support **~100,000 articles/year**
- The article lifetime in submission is ~ 1 week, which means, **~2,000 articles active at any time**
- Each article involves 2 users → ~4,000 potential users, ~400 concurrent users
- Must handle **spikes** of up to **10 concurrent uploads/downloads/sec**
- Each article has **~13MB** in total:
  - 1 manuscript (~5MB)
  - 4 supplementary files (~2MB each)
- File retention: 2 years
- Expected storage: ~2.5TB over 2 years
- Target availability: 99.9% uptime
- 95% of API requests should respond in <1s

- 🛡️ **Security**
  - **Role-based** access control mechanism: Author, Editor. A new Auth Service.
  - **Extend the security policy** inside the Submission service.
- 🚀 **Scalability**
  - Based on the numbers, **the system has predictable usage** and won't grow overnight or experience sudden spikes.
  - Submission service is not heavily used and is not critical for the workflow, so we **don't need to invest in a high-end scaling** setup
- 🟢 **Performance & Availability** (Not a Critical Service)
  - According to the numbers, **performance is not a major concern**. Most actions are simple and fast, and we don't expect high load or large data.
  - Target uptime: **99.9%** (~43 minutes downtime/month)
- 📄 **Data Validation & Integrity**
  - All fields must be **validated** (e.g., file size, type, required metadata)
  - Articles are **locked** after submission or final decision — no further edits allowed, we need a **state machine** to define **allowed actions and transitions** per stage.
- 🗄️ **Audit & Tracking**
  - **Action-level audit tracking** key transitions (Create, Submit, Approve, Reject)
  - Audit must be queryable and global → New Audit Service
- 💾 **Data Storage & Retention**
  - Each article stores a small number of files, for 2 years, with a total size of around 13MB. That means the overall storage needs are **modest.**

Clean Architecture

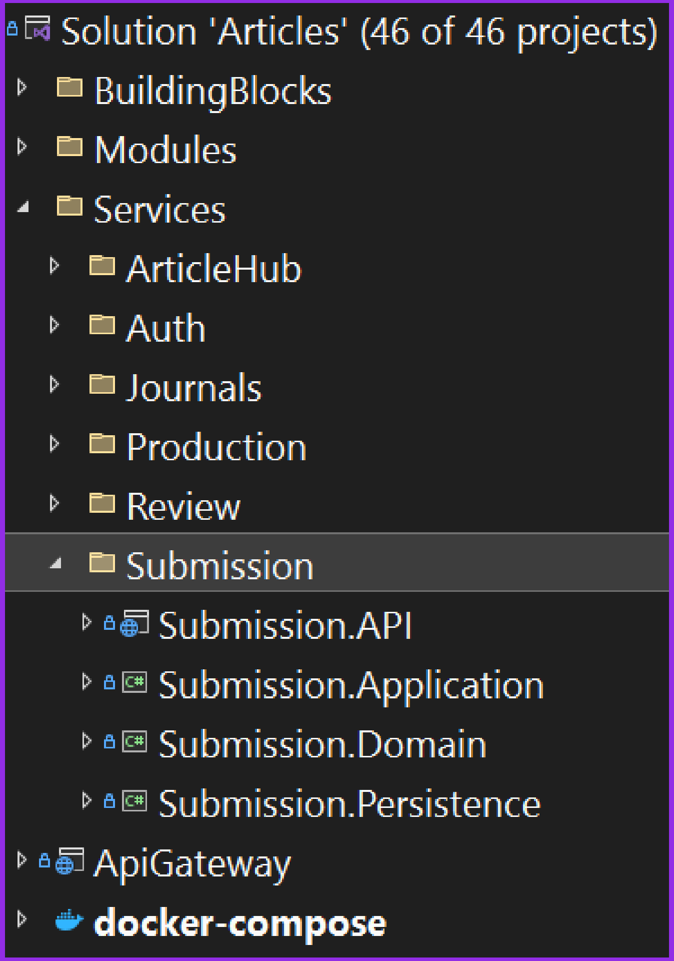
- **API / Presentation**
  - Endpoints with Minimal APIs (or Controllers)
  - Integrates Authorization & other middleware(s)
  - Acts as the composition root — wires all external infrastructure and module dependencies (e.g., FileStorage, EmailService, etc.)
  - Passes commands/queries to the Application layer using MediatR.
  - **Depends on**: `Application`
- **Application**
  - Coordinates the use case logic of the system.
  - Each feature slice includes:
    - A **Command/Query &** A **Validator** (FluentValidation)
    - A **Handler** (MediatR) - coordinates the feature logic
    - A **Mapping configuration** (Mapster)
  - **Depends on**:
    - `Domain` (for domain models)
    - `Persistence`(for DbContext & Repositories) & other `Infrastructure` integrations





- **Domain**
  - Core business logic and rules.
  - Contains:
    - **Aggregates** (Article, Asset)
    - **Entities**(Journal, ArticleActor, Person, ArticleStageInfo etc.)
    - **Value Objects**(AssetName, File, FileExtension etc.)
    - **Domain Events**(ArticleSubmitted, ArticleAccepted etc.)
  - Domain Functions – business rules and behavior per feature
  - **Completely isolated** — does not depend on any other layer.
- **Persistence / Infrastructure**
  - Handles all technical concerns and integration points.
  - Contains:
    - EF Core (DbContext, Repositories)
    - Entity Configurations
    - SaveChangesInterceptor (for dispatching Domain Events)
  - **Depends on**: `Domain`

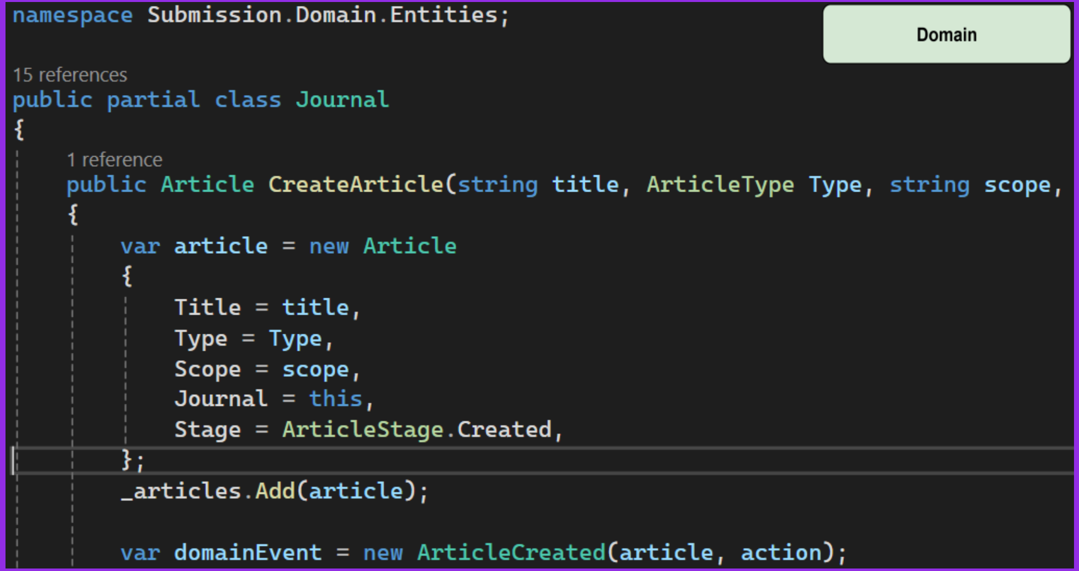
Submission – Structure



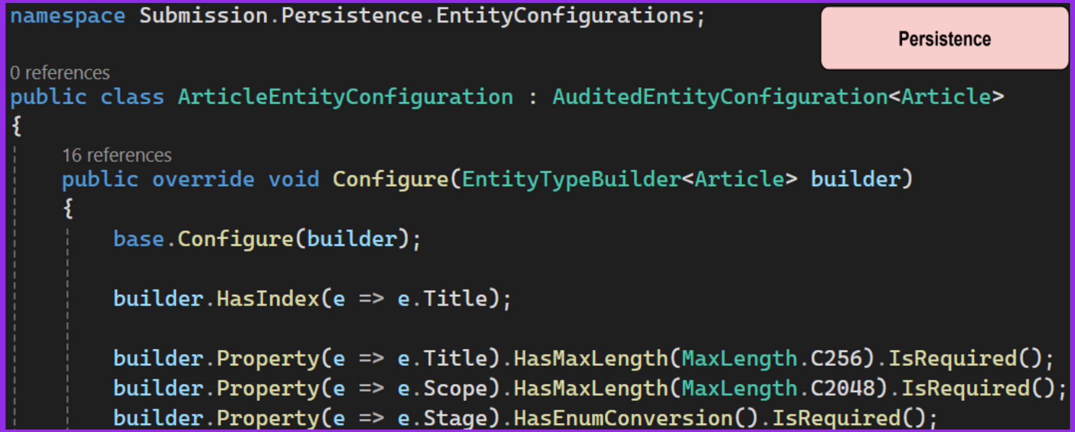
```
Solution 'Articles' (46 of 46 projects)
  BuildingBlocks
  Modules
  Services
    ArticleHub
    Auth
    Journals
    Production
    Review
    Submission
      Submission.API
      Submission.Application
      Submission.Domain
      Submission.Persistence
  ApiGateway
  docker-compose
```

- **Clean Architecture Projects Setup**
  - Create the solution and 4 projects: **API**, **Application**, **Domain**, **Persistence**
  - Add project references and essential **NuGet packages**
- **Designing the Domain Model**
  - Define Aggregates, Entities, Value Objects, Events and domain behavior
- **Configuring Persistence**
  - Set up **DbContext** and EF Core configuration
  - Create the **first migration** and apply it
- **Implementing the Vertical Slice**
  - Create folders in each of the Projects following Vertical Slice
  - Implement Command, Validator, Handler
  - Apply business rules and trigger domain logic
- **Exposing the Endpoint**
  - Add **Minimal API endpoints** and set up routing
  - Wire everything up in the **API startup**
- **Docker & End-to-End Testing**
  - Add **Dockerfile** and **docker-compose** setup
  - Test the flow using **Swagger** or **Postman**
- **Pushing to GitHub** (optional)
  - Initialize Git and push the code to **GitHub**

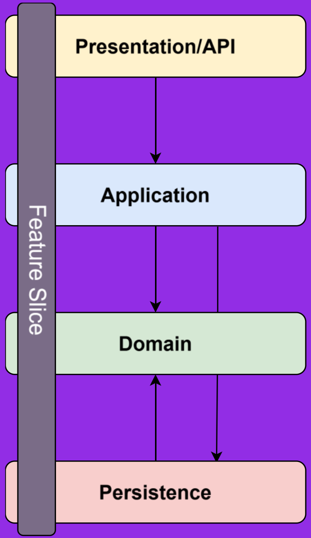
Submission – Create Article Feature

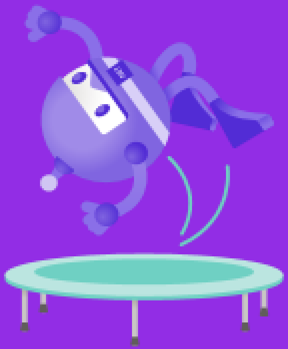


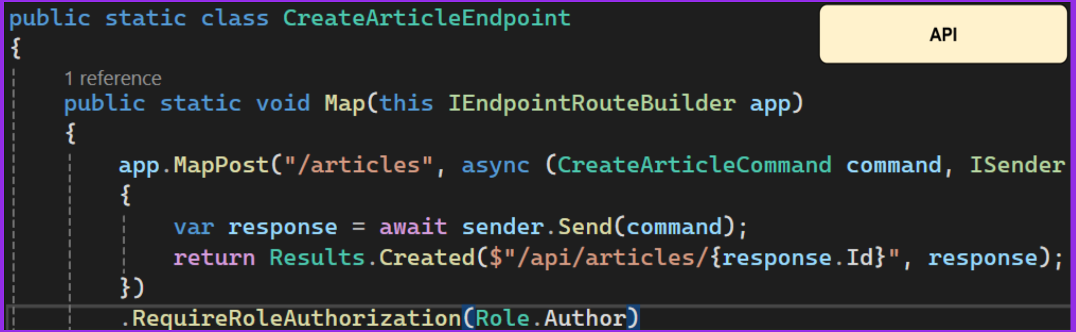
```
namespace Submission.Domain.Entities;

15 references
public partial class Journal
{
    1 reference
    public Article CreateArticle(string title, ArticleType Type, string scope,
    {
        var article = new Article
        {
            Title = title,
            Type = Type,
            Scope = scope,
            Journal = this,
            Stage = ArticleStage.Created,
        };
        _articles.Add(article);

        var domainEvent = new ArticleCreated(article, action);
```



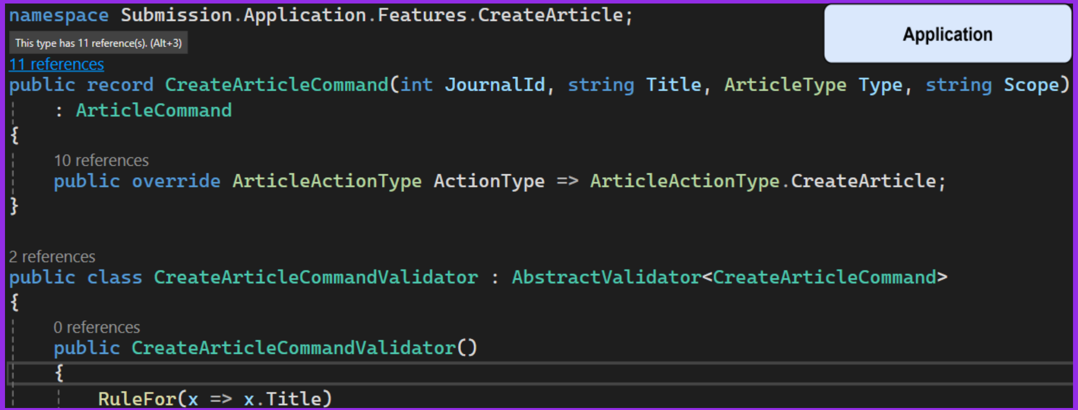
```
namespace Submission.Persistence.EntityConfigurations;

0 references
public class ArticleEntityConfiguration : AuditedEntityConfiguration<Article>
{
    16 references
    public override void Configure(EntityTypeBuilder<Article> builder)
    {
        base.Configure(builder);

        builder.HasIndex(e => e.Title);

        builder.Property(e => e.Title).HasMaxLength(MaxLength.C256).IsRequired();
        builder.Property(e => e.Scope).HasMaxLength(MaxLength.C2048).IsRequired();
        builder.Property(e => e.Stage).HasEnumConversion().IsRequired();
```



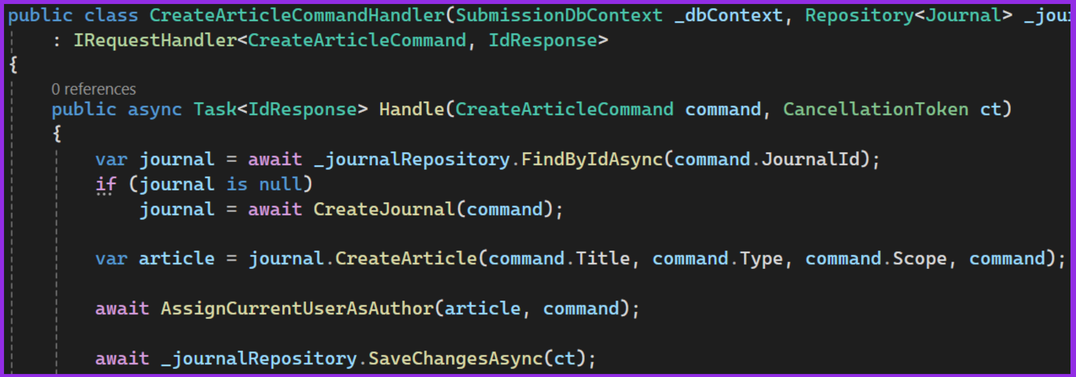




```
public static class CreateArticleEndpoint
{
    1 reference
    public static void Map(this IEndpointRouteBuilder app)
    {
        app.MapPost("/articles", async (CreateArticleCommand command, ISender
        {
            var response = await sender.Send(command);
            return Results.Created($"/api/articles/{response.Id}", response);
        })
        .RequireRoleAuthorization(Role.Author)
```



```
namespace Submission.Application.Features.CreateArticle;
This type has 11 reference(s). (Alt+3)
11 references
public record CreateArticleCommand(int JournalId, string Title, ArticleType Type, string Scope)
    : ArticleCommand
{
    10 references
    public override ArticleActionType ActionType => ArticleActionType.CreateArticle;
}

2 references
public class CreateArticleCommandValidator : AbstractValidator<CreateArticleCommand>
{
    0 references
    public CreateArticleCommandValidator()
    {
        RuleFor(x => x.Title)
```



```
public class CreateArticleCommandHandler(SubmissionDbContext _dbContext, Repository<Journal> _jou
    : IRequestHandler<CreateArticleCommand, IdResponse>
{
    0 references
    public async Task<IdResponse> Handle(CreateArticleCommand command, CancellationToken ct)
    {
        var journal = await _journalRepository.FindByIdAsync(command.JournalId);
        if (journal is null)
            journal = await CreateJournal(command);

        var article = journal.CreateArticle(command.Title, command.Type, command.Scope, command);

        await AssignCurrentUserAsAuthor(article, command);

        await _journalRepository.SaveChangesAsync(ct);
```