Journal Microservice Handbook

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

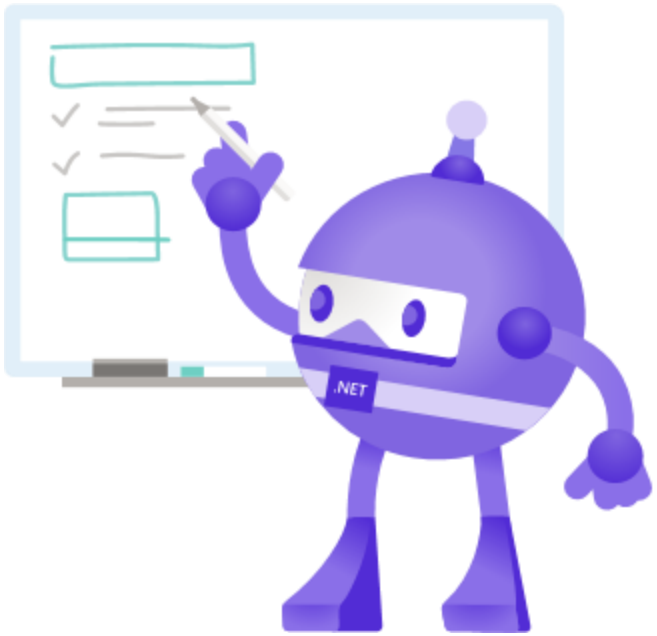
It summarizes the architecture, design diagrams, and code examples covered in the lectures.

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

- Introduction & Overview
  - What This Handbook Covers 1
  - Table of Contents 2
  - Learning Objectives 3
- Architecture & Design
  - High Level Architecture 4
  - Journal Architecture 5
  - Tactical Design Diagram (DDD) 6
  - Sync Journal via Events Diagram.....7
  - Validate Editor Assignment via gRPC Diagram..... 8
- Functional Overview
  - Journal Workflow 9
  - User Stories 10
  - API Endpoints 11
  - Requirements 12
- Implementation
  - Clean Architecture Overview 13
  - Hands-On Projects Structure 14
  - Hands-On Code Snippets 15

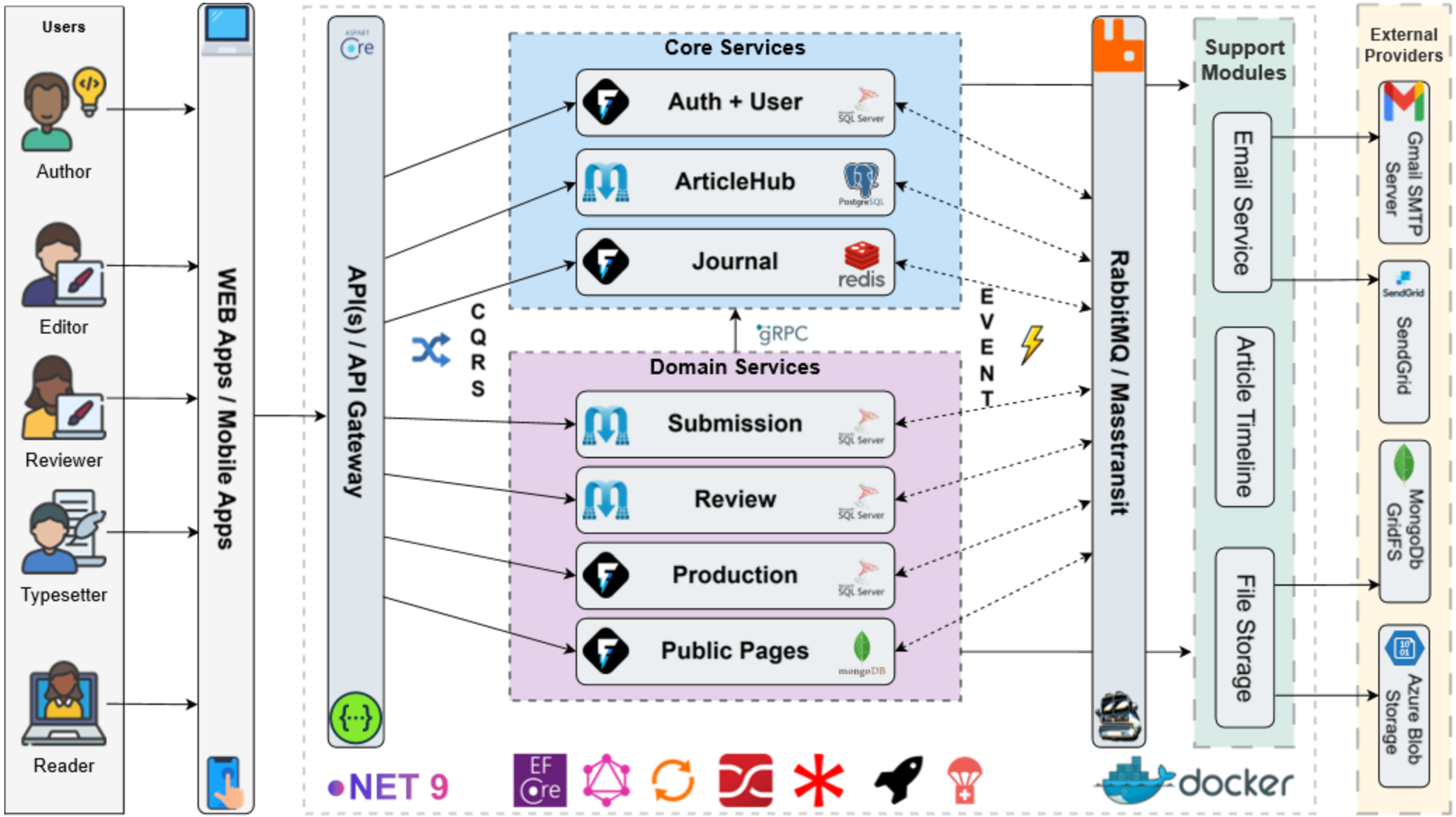


Journal Microservice

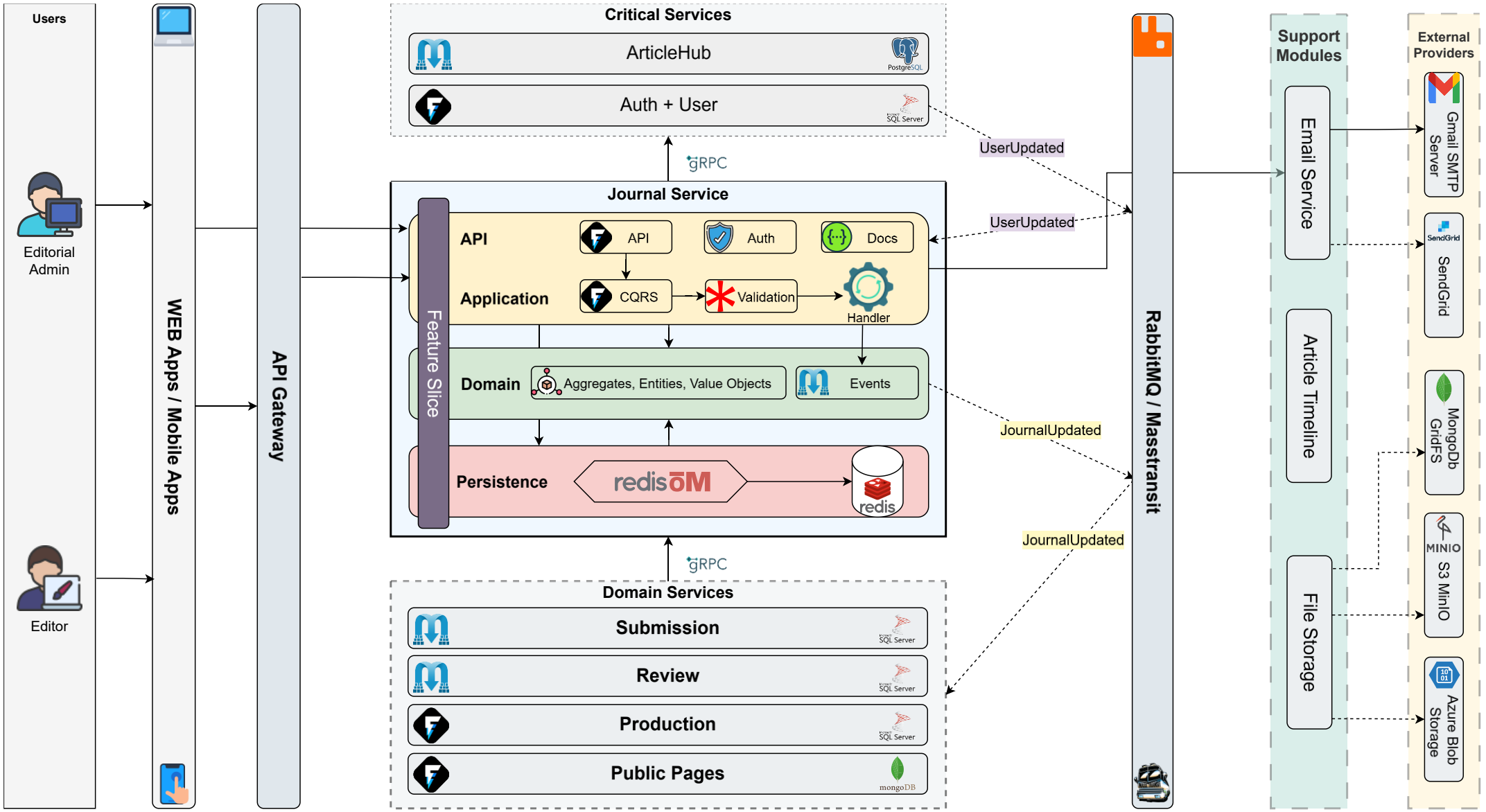
with FastEndpoints, Redis & gRPC

- Build API endpoints and implement CQRS with **FastEndpoints**
- Validate requests using **FluentValidation** with FastEndpoints
- Model the domain with **Redis.OM**
- Store data in **Redis** as a database & search with **RediSearch indexes**
- Send confirmation emails via **domain event handlers**
- Publish integration events with **RabbitMQ** and **MassTransit**
- Expose journal data via **gRPC (protobuf-net)**

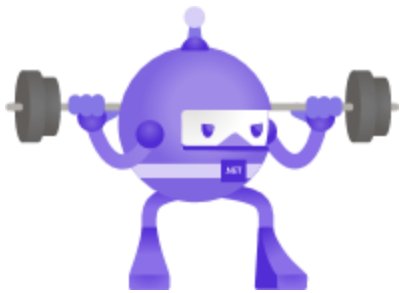
High Level Architecture | C4 Level 2 (Container View)

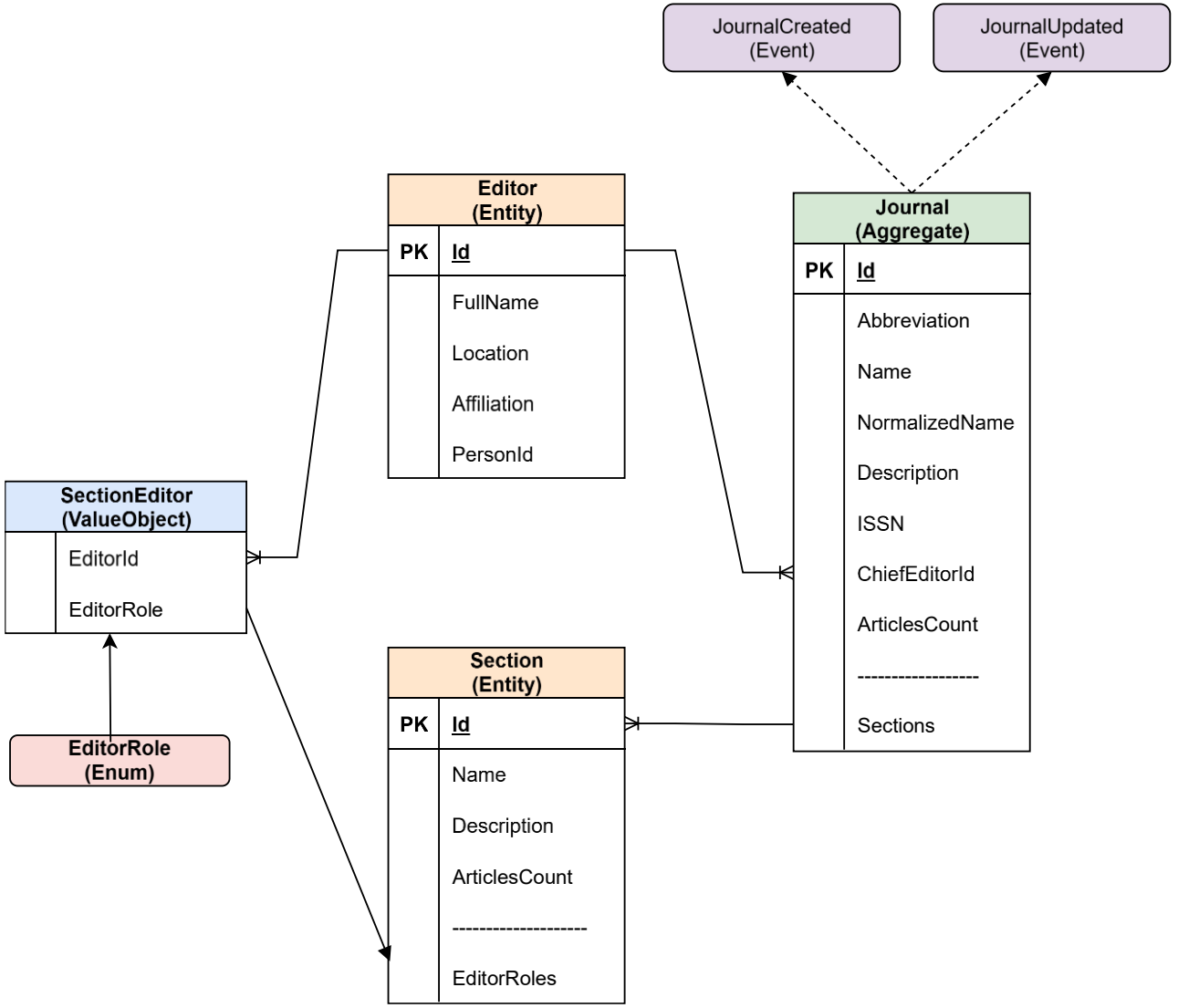


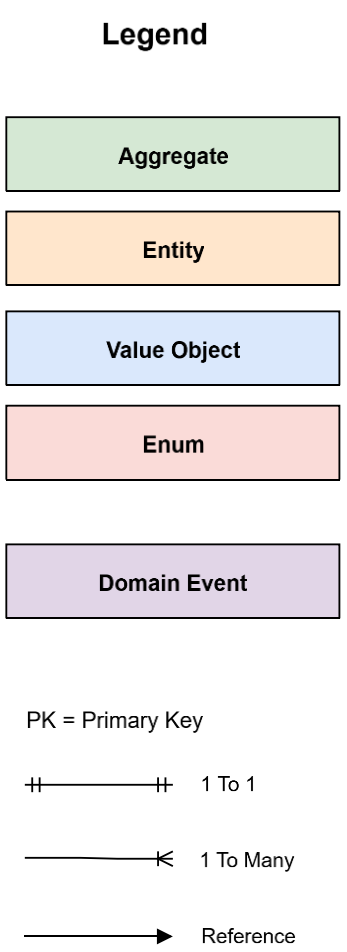
Journal Service Architecture – C4 Level 2 (Container View)



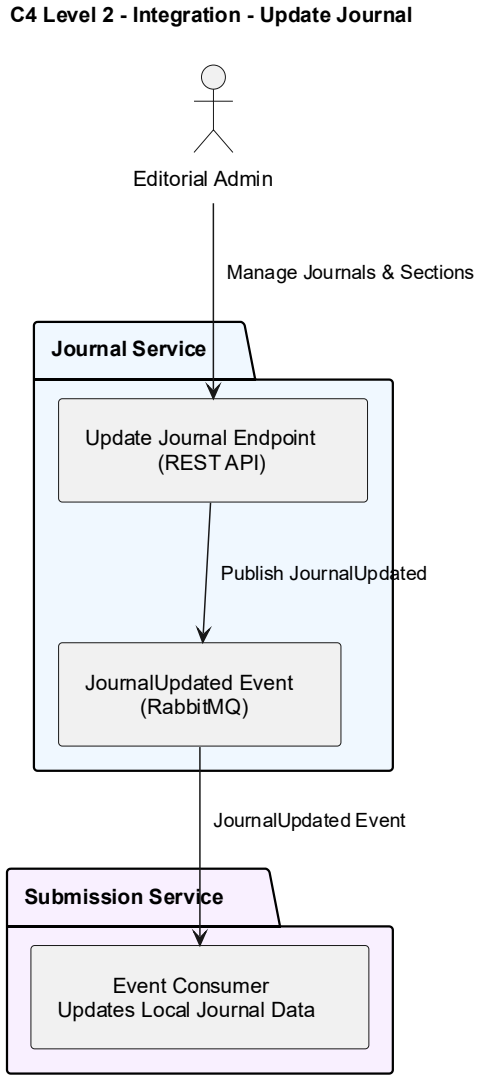
Tactical Design Diagram (DDD) - C4 Level 4





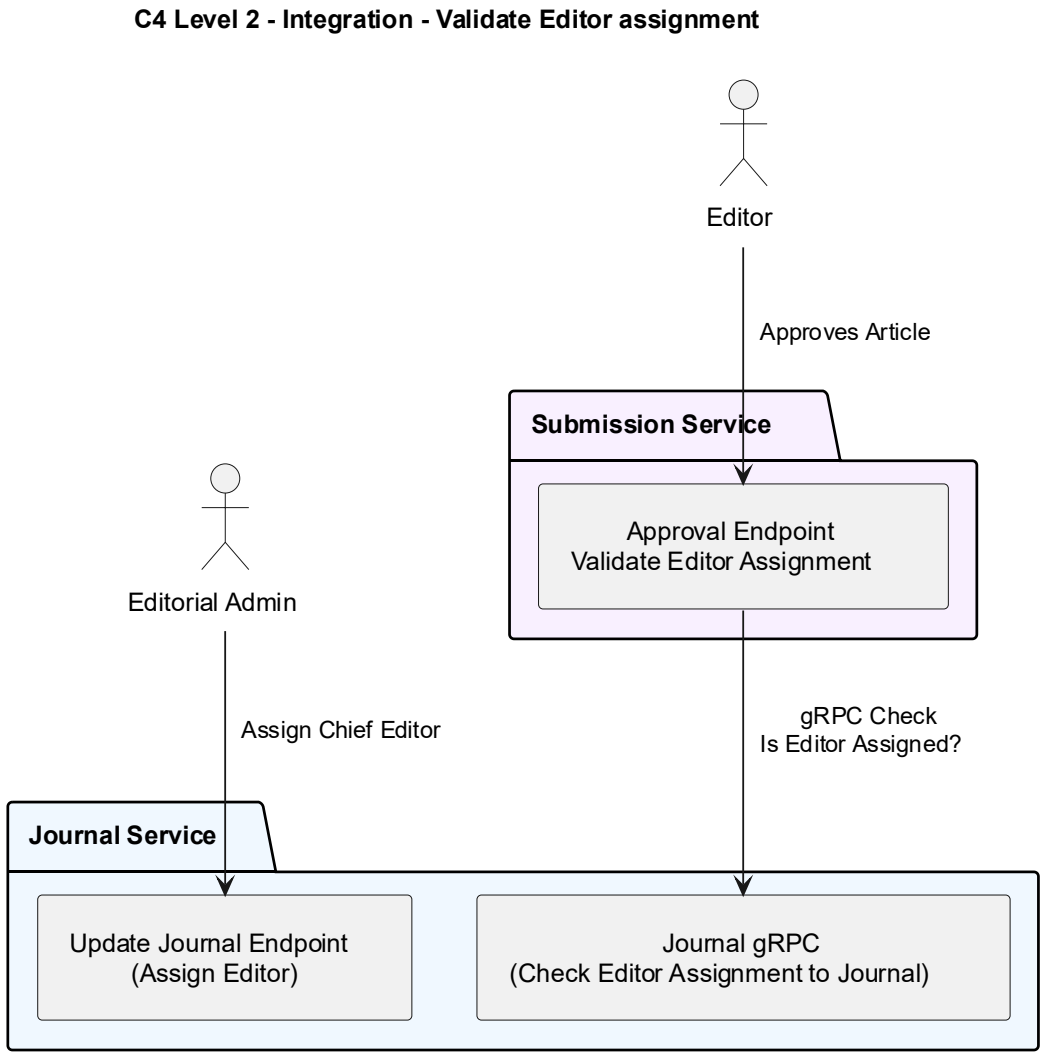


Integration Scenario - Sync Journal Data via Events



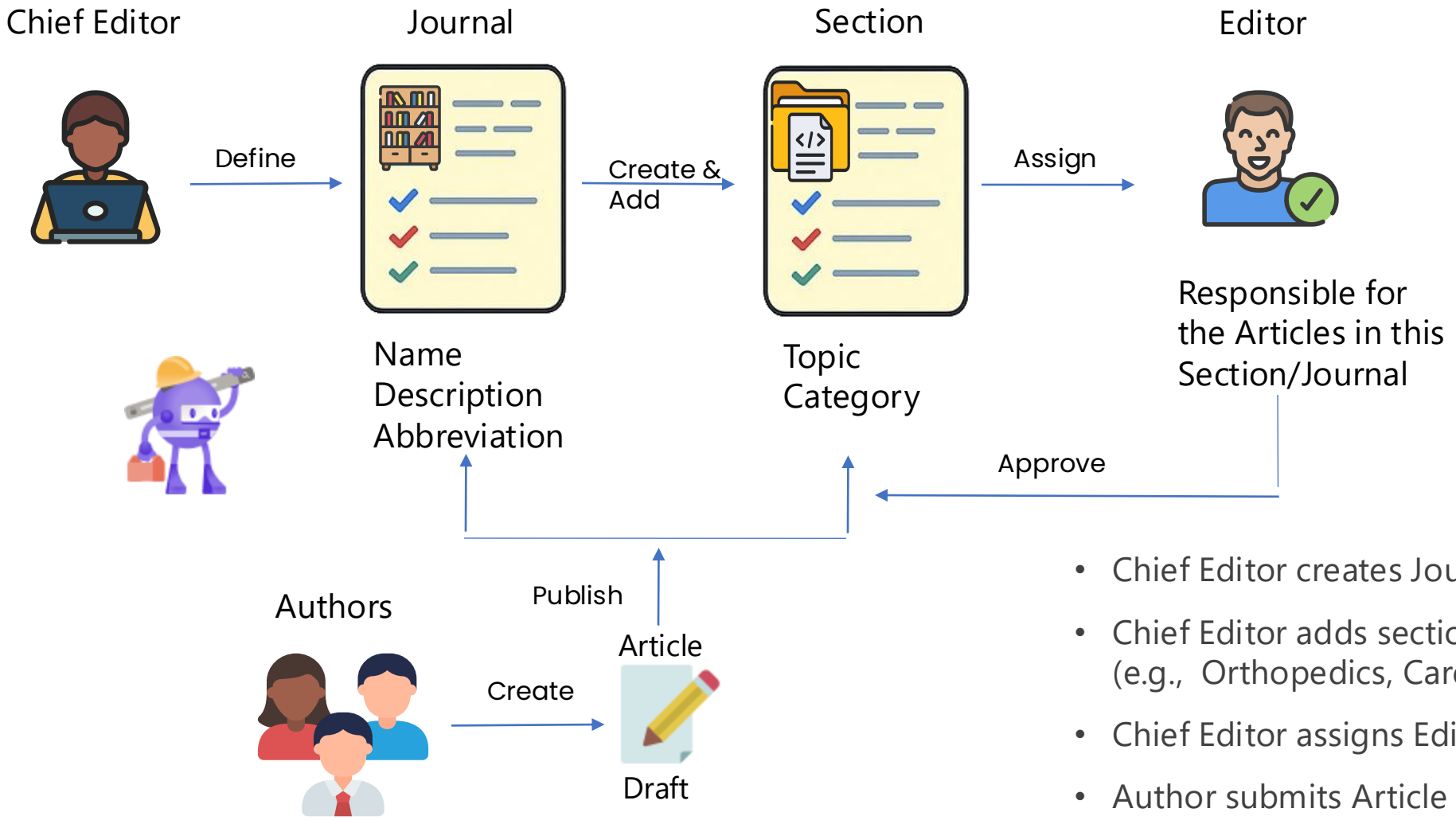
Editorial Admin updates Journals/Sections.
JournalUpdated event keeps Submission Service data in sync.

Integration Scenario - Validate Editor Assignment



Editorial Admin assigns Chief Editor.
Editor approval triggers gRPC validation of assignment in Journal Service.

Journal Workflow



- Chief Editor creates Journal (e.g., Medicine)
- Chief Editor adds sections to Journal (e.g., Orthopedics, Cardiology)
- Chief Editor assigns Editors to Journal/Section
- Author submits Article to Journal/Section
- Editor approves Article

User Stories

- **Create Journal**

  As an **Editorial Admin**, I want **to create a new Journal**, so that it can be managed and assigned a Chief Editor.

- **Update Journal**

  As an **Editorial Admin**, I want **to update the details of an existing Journal**, so that I can correct information or change the Chief Editor.

- **Get Journal**

  As an **Editorial Admin**, I want **to view the details of a specific Journal**, so that I can verify or edit its configuration.
  As a **Service**, I want to **fetch journal information by ID**, so that I can enforce business rules or show journal details.

- **List & Search Journals**

  As an **Editorial Admin**, I want **to list and search Journals by name or abbreviation**, so that I can quickly find and manage specific Journals in the system.

- **Create Section**

  As an **Editorial Admin**, I want **to add a new Section to a Journal**, so that articles can be organized into thematic areas.

- **Update Section**

  As an **Editorial Admin**, I want **to update the details of a Section**, so that its description or assigned Editors can be changed.

- **Get Section**

  As an **Editorial Admin**, I want **to view details of a specific Section**, so that I can review its configuration and assigned Editors.

- **Get Editors By Section**

  As an **Editorial Admin**, I want **to list all Editors assigned to a Section**, so that I can manage their roles and assignments.

Endpoints

Name	Method	Roles	Endpoint
Create Journal	POST	ADMIN	/journals
Update Journal	PUT	ADMIN	/journals/{journalId}
Get Journal	GET	ADMIN, EDIT	/journals/{journalId}
List & Search Journals	GET	ADMIN, EDIT	/journals?search={search}&page={page}&pageSize={size}
Create Section	POST	ADMIN	/journals/{journalId}/sections
Update Section	PUT	ADMIN	/journals/{journalId}/sections/{sectionId}
Get Section	GET	ADMIN, EDIT	/journals/{journalId}/sections/{sectionId}
Get Editors by Section	GET	ADMIN, EDIT	/journals/{journalId}/sections/{sectionId}/editors

Requirements

Functional 

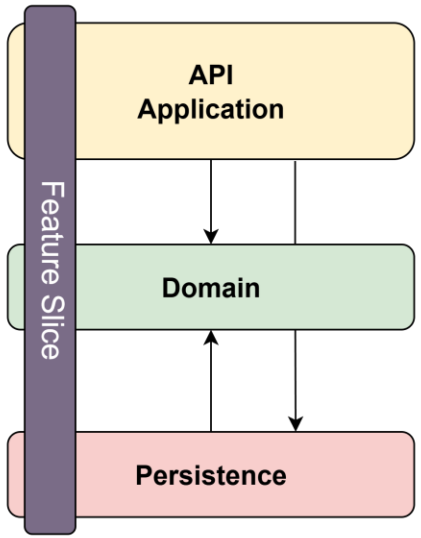
- **Create/Update Journal**
  - Required fields → Name, Abbreviation, ChiefEditorId
  - Validation → Unique name/abbreviation
- **List & Search Journals**
  - Search with pagination→ By Name, Abbreviation
- **Create/Update Section**
  - Required fields → Name, Description, JournalId
  - Assign Editors → Create Editors if they don't exist, using user info from the Auth service
- **Get Editors By Section**
  - List of Editors with basic Info (Name, Affiliation)
- **Integration with Services**
  - Broadcast notifications when Journals/Sections change
  - Expose gRPC endpoint(s) to retrieve Journal/Section.
  - Update Editor info when it changes in Auth service

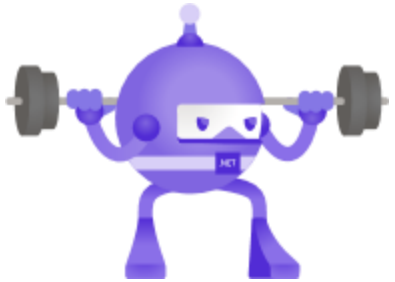
 Non-Functional

- **Security**
  - Role-based access enforced via Auth Service
  - Editorial Admin → Full access to all endpoints
  - Editor → Read-only access to assigned Journals/Sections
- **Performance**
  - Low write activity but high read traffic
  - Use caching to improve read performance and reduce database load
- **Scalability**
  - Support many Journals & Sections without performance loss
  - Efficient queries for Search & Listing
- **Consistency**
  - Changes should be immediately available to consuming services

Clean Architecture

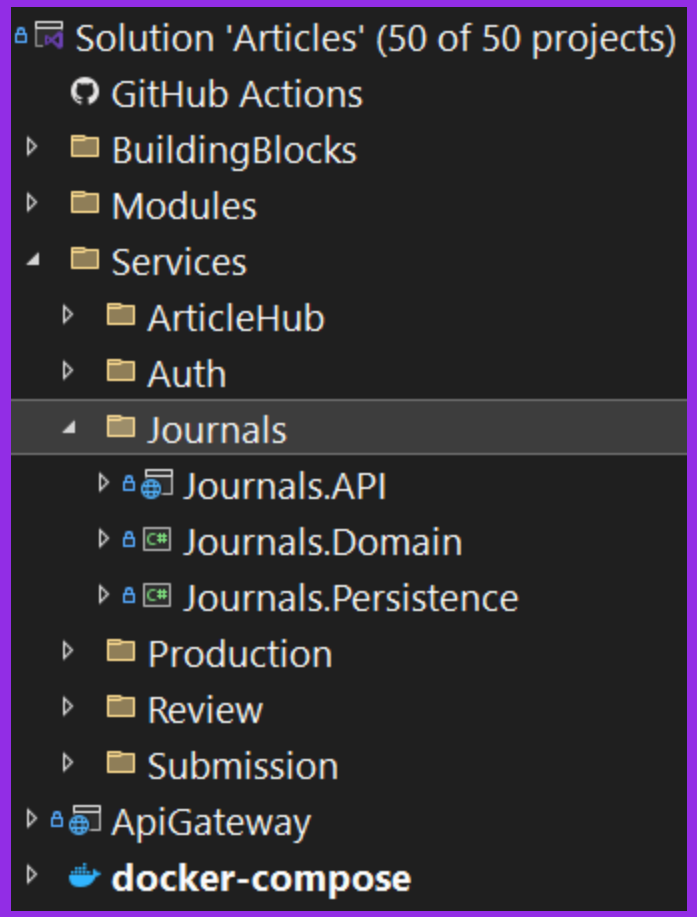
- **API / Application**
  - Endpoints with FastEndpoints
  - Integrates Authorization & other middleware(s)
  - Coordinates the use case logic of the system.
  - Each feature slice includes:
    - A **Command/Query &** A **Validator** (FluentValidation)
    - A **Handler** (FastEndpoints) - coordinates 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** (Journal)
    - **Entities(**Section, Editor)
    - **Domain Events**(JournalCreated, JournalUpdated)
  - Domain Functions – business rules and behavior per feature
  - **Completely isolated** — does not depend on any other layer.
- **Infrastructure / Persistence**
  - Handles all technical concerns and integration points.
  - Contains:
    - Redis.OM (DbContext, Repositories)
    - References to shared modules (e.g., EmailService)
  - Implements contracts or patterns defined in Application or Domain.
  - **Depends on**: `Domain`

Journal – Structure



- **Clean Architecture Projects Setup**
  - Create the solution and 3 projects: **API**, **Domain**, **Persistence**
  - Add project references and essential **NuGet packages**
- **Designing the Domain Model**
  - Define Aggregates, Entities and Domain Events
- **Configuring Persistence**
  - Set up **DbContext** with StackExchange.Redis & Redis.OM
  - Configure entities with Redis.OM
- **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 FastEndpoints **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**

Journal – Create Journal Feature

API / Application

```
namespace Journals.API.Features.Journals.Create;

3 references
public record CreateJournalCommand(string Abbreviation, string Name, string Description, string ISSN,
    int ChiefEditorId)
{
    1 reference
    public string NormalizedName => Name.ToLowerInvariant();
}

1 reference
public class CreateJournalCommandValidator : Validator<CreateJournalCommand>
{
    0 references
    public CreateJournalCommandValidator()
    {
        RuleFor(r => r.Abbreviation).NotEmpty();
        RuleFor(r => r.Name).NotEmpty();
        RuleFor(r => r.ISSN).NotEmpty().Matches(@"\d{4}-\d{3}[\dX]").WithMessage("Invalid ISSN format");
```

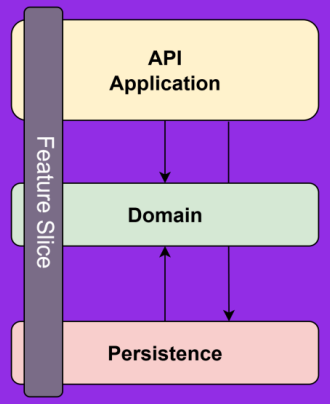
```
[Authorize(Roles = Role.EditorAdmin)]
[HttpPost("journals")]
[Tags("Journals")]
0 references
public class CreateJournalEndpoint(Repository<Journal> _journalRepository, Repository<Editor> _editorRepository,
    : Endpoint<CreateJournalCommand, IdResponse>
{
    0 references
    public override async Task HandleAsync(CreateJournalCommand command, CancellationToken ct)
    {
        if (_journalRepository.Collection.Any(j => j.Abbreviation == command.Abbreviation ||  j.NormalizedName ==
            throw new BadRequestException("Journal with the same name or abbreviation already exists");

        if (!_editorRepository.Collection.Any(e => e.Id == command.ChiefEditorId))
            await CreateEditor(command.ChiefEditorId, ct);

        var journal = command.Adapt<Journal>();

        await _journalRepository.AddAsync(journal);

        await PublishAsync(new JournalCreated(journal));
```



Domain

```
namespace Journals.Domain.Journals;

[Document(StorageType = StorageType.Json, Prefixes = new[] { nameof(Journal) })]
23 references
public partial class Journal : Entity
{
    [Indexed]
    2 references
    public required string Abbreviation { get; set; }

    private string _name = null!;
    [Searchable]
    1 reference
    public required string Name
    {
        get => _name;
        set
        {
            _name = value;
            NormalizedName = _name.ToLowerInvariant(); // Normalize on set
```

Persistence

```
namespace Journals.Persistence;

5 references
public class JournalDbContext
{
    private readonly RedisConnectionProvider _provider;
    private readonly IDatabase _redisDb;

    0 references
    public JournalDbContext(IConnectionMultiplexer redis, RedisConnectionProvider provider) =>
        (_redisDb, _provider)= (redis.GetDatabase(), provider);

    3 references
    public IRedisCollection<Journal> Journals => _provider.RedisCollection<Journal>();
    //public IRedisCollection<Section> Sections => _provider.RedisCollection<Section>();
    4 references
    public IRedisCollection<Editor> Editors => _provider.RedisCollection<Editor>();

    0 references
    public RedisConnectionProvider Provider => _provider;

    0 references
    public async Task<int> GenerateNewId<T>() => (int)await _redisDb.StringIncrementAsync($"{t
```