Review Microservice Handbook

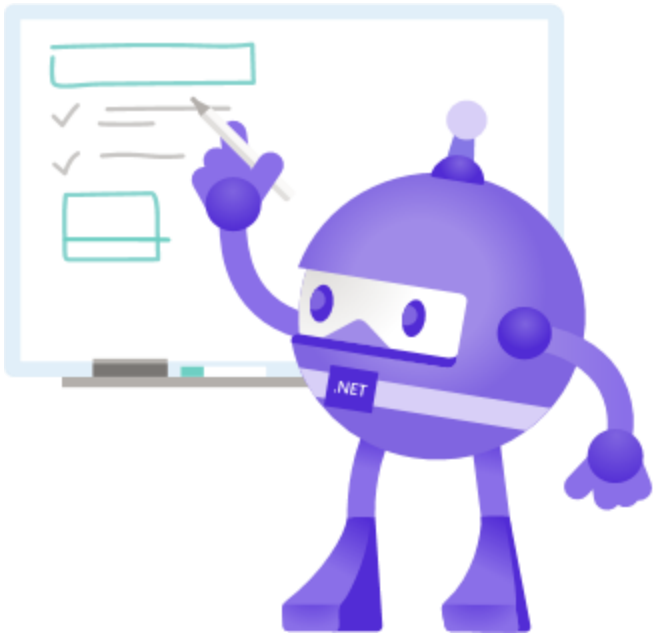
This handbook is a visual companion to the Review Microservice module of the course.

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

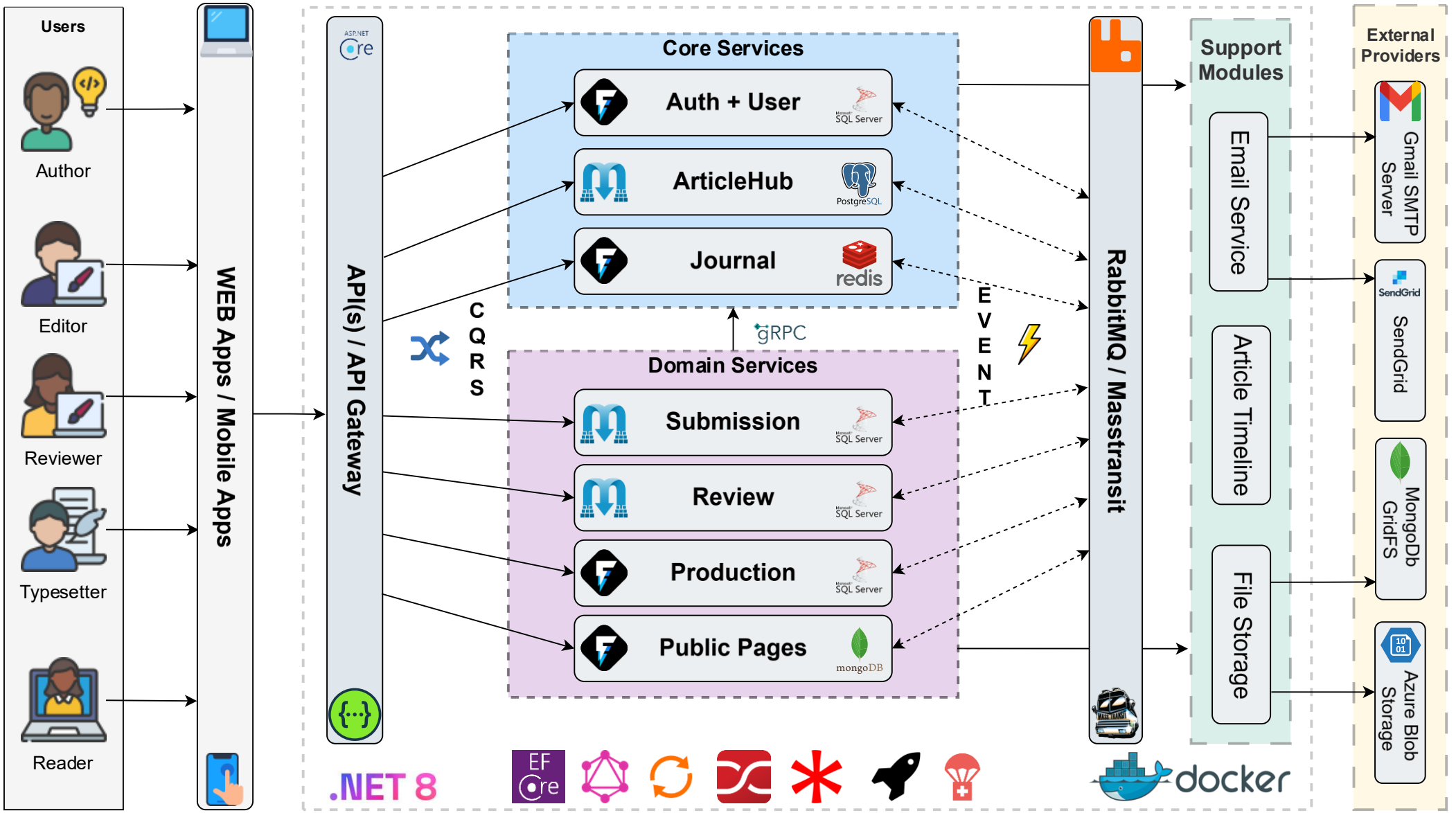


Review Microservice

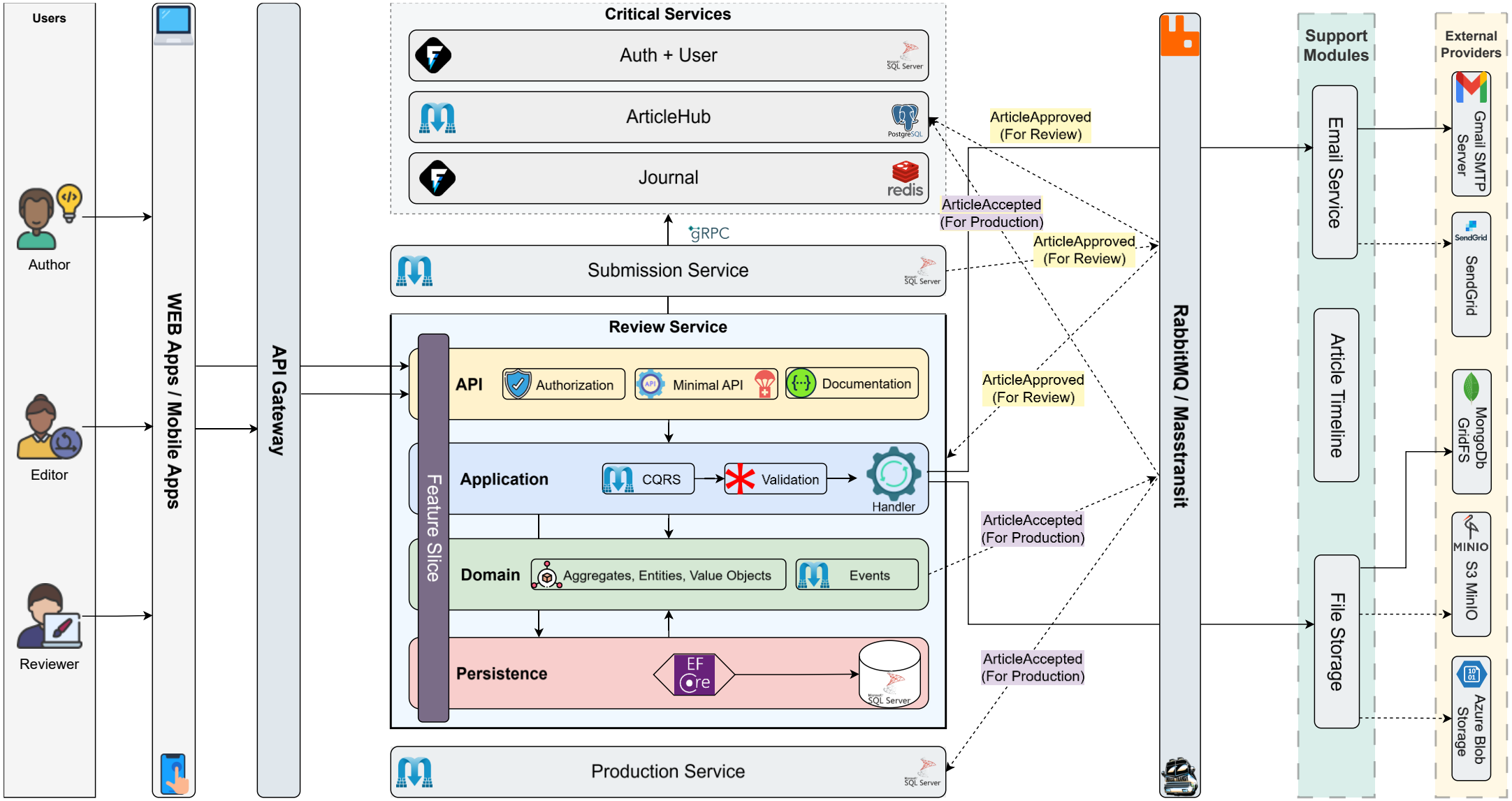
with MediatR, FluentValidation & EF Core

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

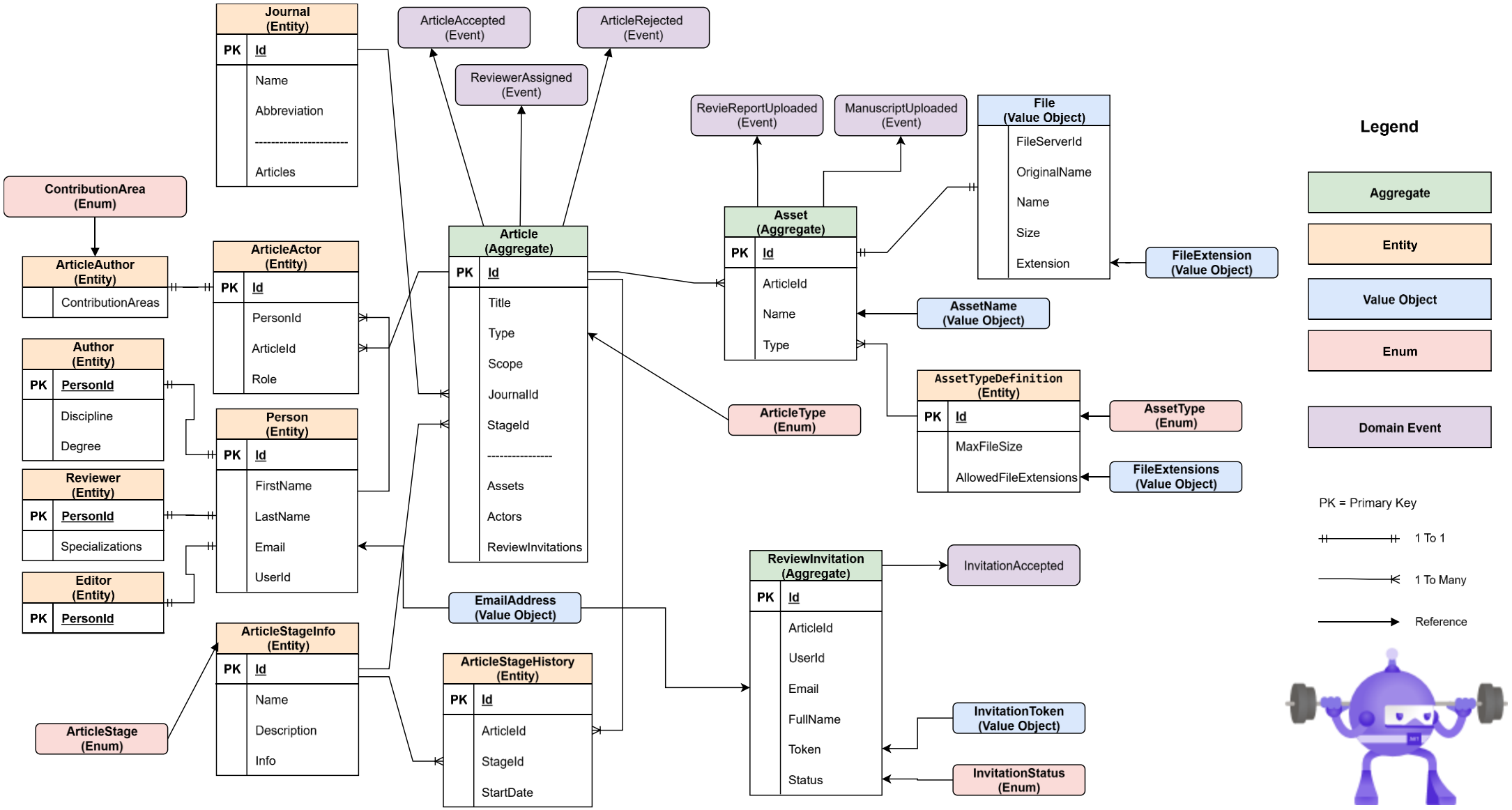
High Level Architecture | C4 Level 2 (Container View)



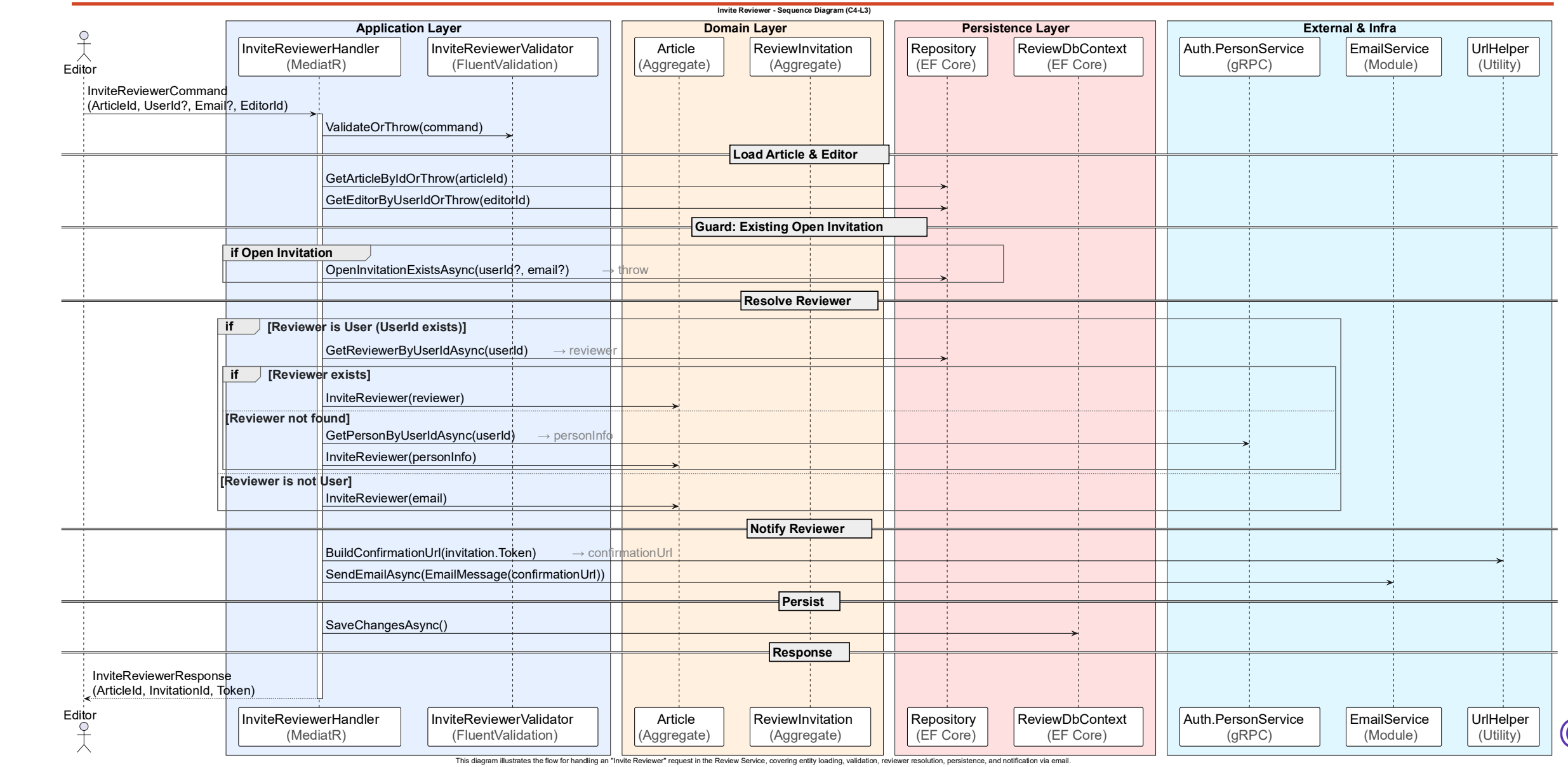
Review Service Architecture – C4 Level 2 (Container View)



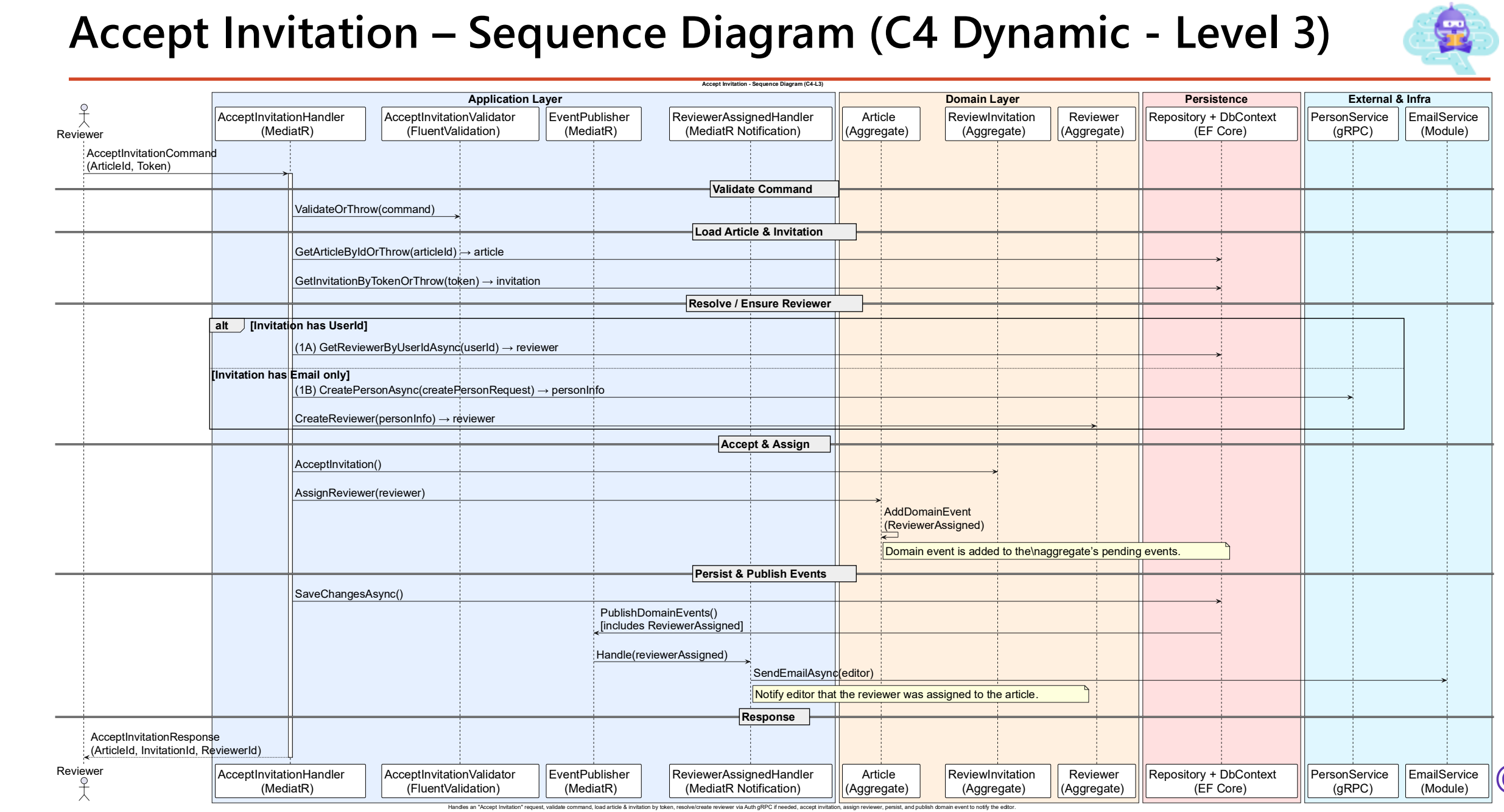
Tactical Design Diagram (DDD) - C4 Level 4



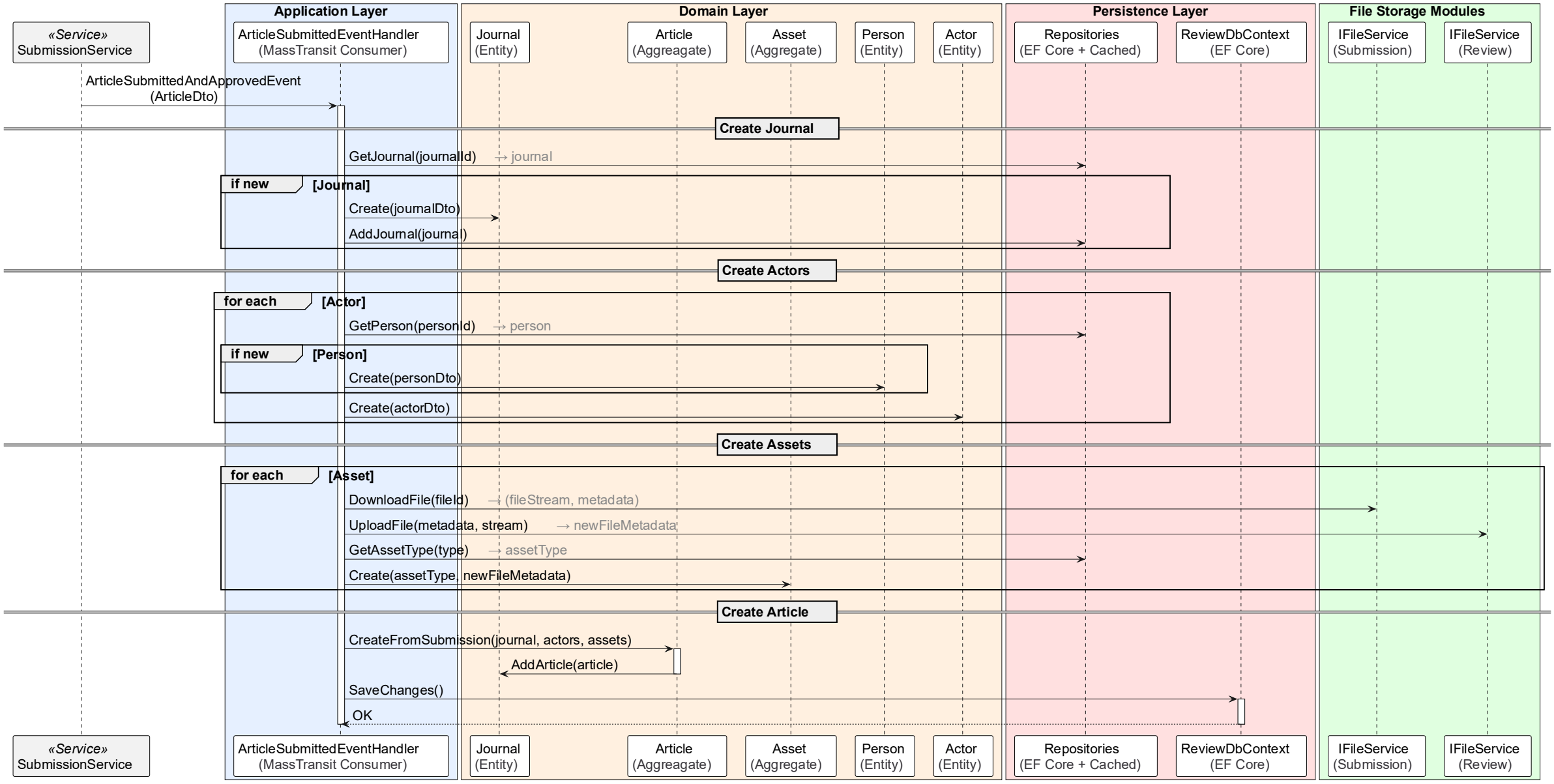
Invite Reviewer – Sequence Diagram (C4 Dynamic - Level 3)



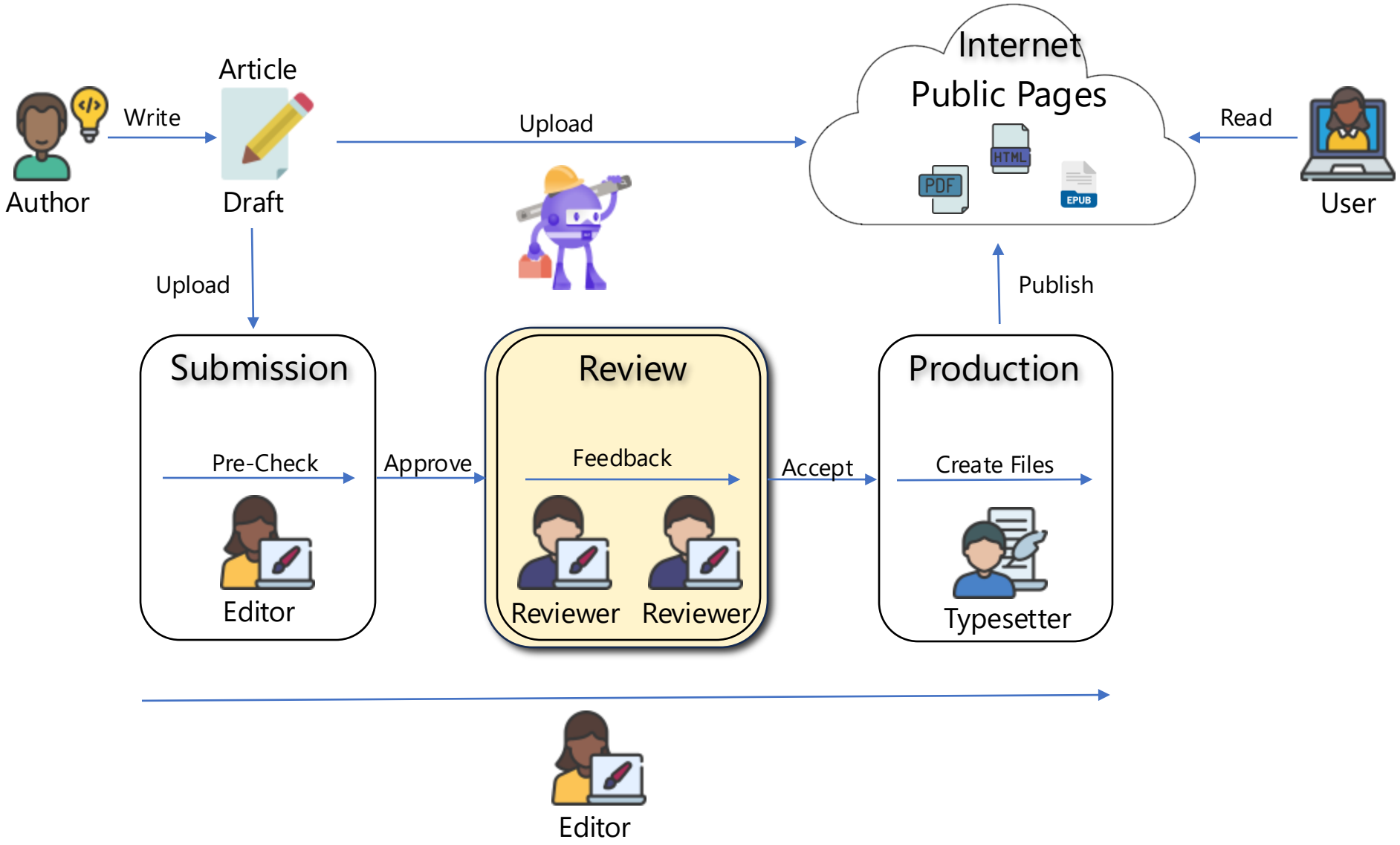
Accept Invitation – Sequence Diagram (C4 Dynamic - Level 3)



From Submission to Review: Article Transfer - Sequence Diagram (C4 Dynamic - L3)



Article Workflow



User Stories

- **Invite Reviewer**
  - As an **Editor**, I want to **invite a reviewer to assess the article**, so I can gather expert feedback.
- **Accept Invitation**
  - As a **Reviewer**, I want **to accept an invitation**, so I can start reviewing the article.
- **Reject Invitation**
  - As a **Reviewer**, I want **to reject an invitation**, if I'm unavailable or not the right fit.
- **Upload Review Report File**
  - As a **Reviewer**, I want **to upload my review report**, so the editor can see my feedback.
- **Upload Manuscript**
  - As an **Author**, I want **to upload the final manuscript after applying the reviewers' feedback**, so it's ready for production.
- **Accept Article**
  - As an **Editor**, I want **to accept the article for production**, so it can move to the next stage.
- **Reject Article**
  - As an **Editor**, I want **to reject the article if the reviews are negative or insufficient,** so that low-quality or unfit submissions don't move forward to production.
- **Get Article**
  - *As a* **reviewer, editor or author**, *I want* **to view the details of an article** *so that* I can review or take action depending on its stage.
- **Download File**
  - *As a* **reviewer, editor or author**, *I want* **to download uploaded files** *so that* I can review the article content or attachments.

Endpoints

Name	Method	Roles	Endpoint
Invite Reviewer	POST	EDIT	/api/articles/{articleId}/invitations
Accept Invitation	POST	-	/api/articles/{articleId}/invitations/{token}:accept
Decline Invitation	POST	-	/api/articles/{articleId}/invitations/{token}:decline
Upload Review Report	POST	REV	/api/articles/{articleId}/assets/review-reports:upload
Upload Manuscript	POST	AUT	/api/articles/{articleId}/assets/manuscript:upload
Accept Article	POST	EDIT	/api/articles/{articleId}:accept
Reject Article	POST	EDIT	/api/articles/{articleId}:reject
Get Article	GET	EDIT, REV, AUT	/api/articles/{articleId}
Download File	GET	EDIT, REV, AUT	/api/articles/{articleId}/assets/{assetId}/content /api/articles/{articleId}/assets/{assetId}:download

REV - Reviewer
EDIT - Editor
AUT - Author

Requirements

Functional

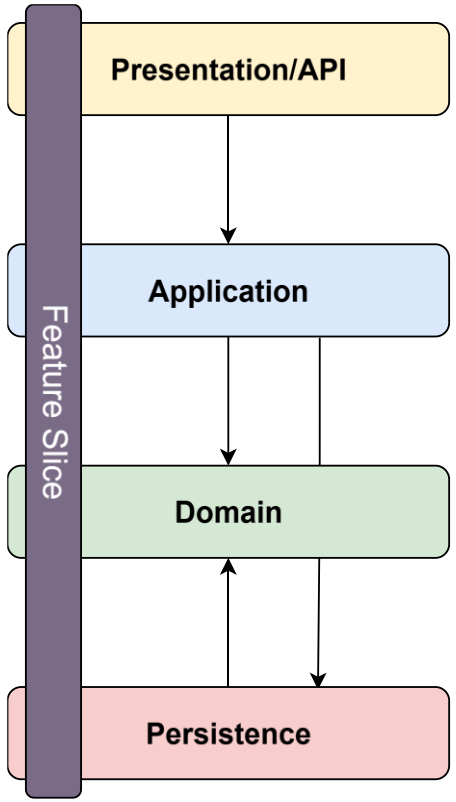
- **Invite Reviewer (Assign)**
  - Invite reviewer by email (existing or new user)
  - Generate a token-based invitation
  - Use gRPC to fetch user info if already exists
  - If user doesn't exist, trigger CreateUser via gRPC (Auth Service)
- **Respond to Invitation**
  - Accept or Decline invitation via token
  - Token must be single-use and time-limited
  - On acceptance: reviewer is added as Actor to the article
- **Upload Review Report**
  - Reviewer uploads 1 report (PDF/DOC, max 10MB) per Article
  - Metadata: Recommendation, FileName, Size, Extension
- **Submit Revised Manuscript**
  - Author uploads new manuscript after revision request
  - Must be of type Manuscript (PDF/DOC, max 10MB)
- **Editorial Decisions**
  - Request Revision (moves article to AwaitingRevision)
  - Accept Article (moves article to Accepted)
  - Reject Article (moves article to Rejected)

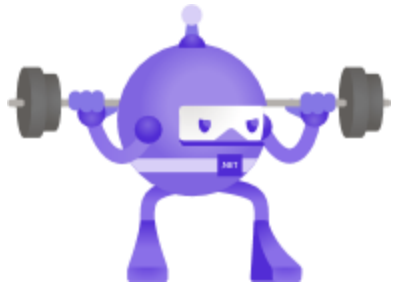
Non-Functional

- **Security (Role-Based)**
  - Only Editors can manage invitations and decisions
  - Only assigned Reviewers can upload reports
  - Only assigned Authors can upload revised manuscript
  - Reviewers and Authors have access only to their articles
- **Performance**
  - Low write but moderate read frequency
- **Scalability**
  - Supports **50,000 articles/year** with ~500 concurrent users
  - Each article has **2-3 reviewers and 1 editor**
- **Consistency**
  - Ensure it processes the article when it is approved in Submission
  - Ensure ArticleHub/Production is updated when article is accepted/rejected
- **Audit & Tracking**
  - Log actions: Invite, Accept/Decline, Upload, Decision
  - Each stage transition must record user, timestamp, and context
- **File Storage**
  - File retention: 2 years
  - Reports and revised manuscripts archived post-acceptance

Clean Architecture

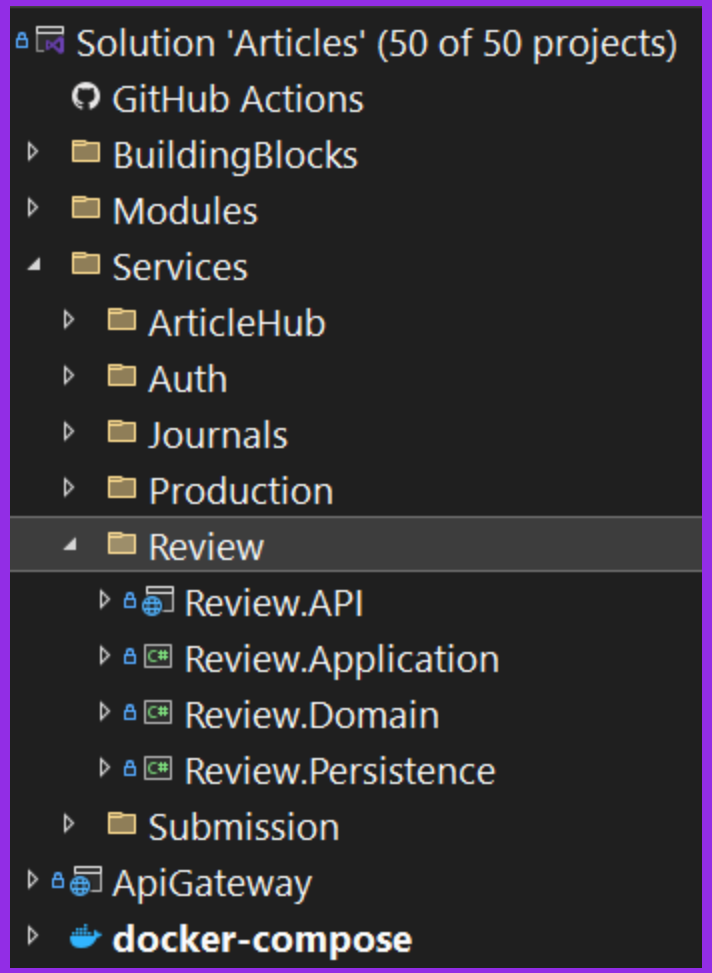
- **API / Presentation**
  - Endpoints with Minimal APIs (or Controllers)
  - Integrates Authorization & other middleware(s)
  - Passes commands/queries to the Application layer using MediatR.
  - **Depends on**: `Application`
- **Application**
  - Coordinates the use case logic of the system.
  - Each 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, ReviewInvitation, Asset)
    - **Entities(**Journal, ArticleActor, Person, Reviewer etc.)
    - **Value Objects**(InvitationToken, AssetName, File etc.)
    - **Domain Events**(ArticleAccepted, InvitationAccepted etc.)
  - 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:
    - EF Core (DbContext, Repositories)
    - SaveChangesInterceptor (for dispatching Domain Events)
    - gRPC clients for external services
    - References to shared modules (e.g., FileStorage)
  - Implements contracts or patterns defined in Application or Domain.
  - **Depends on**: `Domain`

Review – Structure



- **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 Carter 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**

Review – Invite Reviewer Feature

API

```
namespace Review.API.Endpoints.Invitations;

0 references
public class InviteReviewerEndpoint: ICarterModule
{
    0 references
    public void AddRoutes(IEndpointRouteBuilder app)
    {
        app.MapPost("/articles/{articleId:int}/invitations", async (int articleId, InviteReviewerCommand
        {
            command.ArticleId = articleId;
            var response = await sender.Send(command);
            return Results.Ok(response);
        })
        .RequireRoleAuthorization(Role.Editor, Role.EditorAdmin)
        .WithName("Invite Reviewer")
```

Application

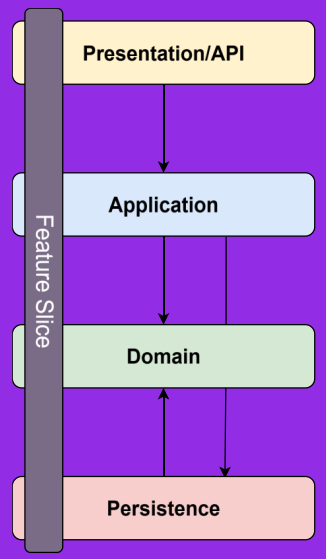
```
namespace Review.Application.Features.Invitations.InviteReviewer;

8 references
public record InviteReviewerCommand(int? UserId, string FirstName, string LastName, string Email)
        : ArticleCommand<InviteReviewerResponse>
{
    10 references
    public override ArticleActionType ActionType => ArticleActionType.InviteReviewer;
}
4 references
public record InviteReviewerResponse(int ArticleId, int InvitationId, string Token);
2 references
public class InviteReviewerCommandValidator : AbstractValidator<InviteReviewerCommand>
{
    0 references
    public InviteReviewerCommandValidator()
    {
        When(c => c.UserId == null, () =>
        {
            RuleFor(x => x.Email)
                .NotEmptyWithMessage(nameof(InviteReviewerCommand.Email))
                .MaximumLengthWithMessage(MaxLength.C64, nameof(InviteReviewerCommand.Email))
                .EmailAddress();
```

```
public async Task<InviteReviewerResponse> Handle(InviteReviewerCommand command, CancellationToken ct)
{
    var article = await _articleRepository.GetByIdOrThrowAsync(command.ArticleId);
    var editor = await _dbContext.Editors.SingleAsync(r => r.UserId == command.CreatedById);

    if (await _reviewInvitationRepository.OpenInvitationExistsAsync(command.ArticleId, command.UserId, command.Email,
        throw new DomainException("An open invitation already exists for this reviewer.");

    ReviewInvitation invitation = default!;
    if (command.UserId != null)
    {
        var reviewer = await _reviewRepository.GetByUserIdAsync(command.UserId.Value);
        if (reviewer is not null)
        {
            invitation = article.InviteReviewer(reviewer, command);
        }
```



Domain

```
namespace Review.Domain.Invitations;

18 references
public partial class ReviewInvitation : AggregateRoot
{
    4 references
    public required int ArticleId { get; init; }

    5 references
    public int? UserId { get; init; }
    6 references
    public required EmailAddress Email { get; init; }
    5 references
    public required InvitationToken Token { get; init; }
```

Persistence

```
namespace Review.Persistence.EntityConfigurations;

0 references
public class ReviewInvitationEntityConfiguration : EntityConfiguration<ReviewInvitation>
{
    26 references
    public override void Configure(EntityTypeBuilder<ReviewInvitation> builder)
    {
        base.Configure(builder);

        builder.Property(e => e.UserId).IsRequired(false);
        builder.Property(e => e.FirstName).IsRequired().HasMaxLength(MaxLength.C64);
        builder.Property(e => e.LastName).IsRequired().HasMaxLength(MaxLength.C64);
        builder.Property(e => e.Status).IsRequired().HasEnumToStringConversion(MaxLength.C8);

        builder.ComplexProperty(
            o => o.Email, builder =>
```