ArticleHub Microservice Handbook

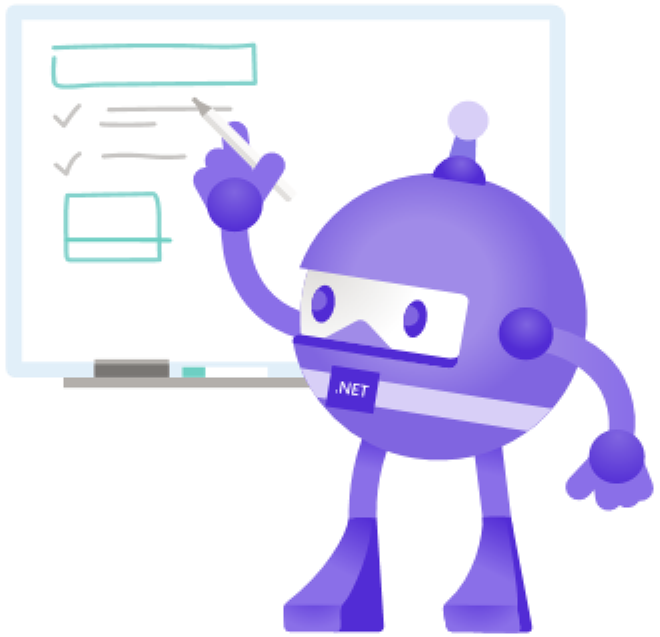
This handbook is a visual companion to the ArticleHub Microservice module of the course.

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



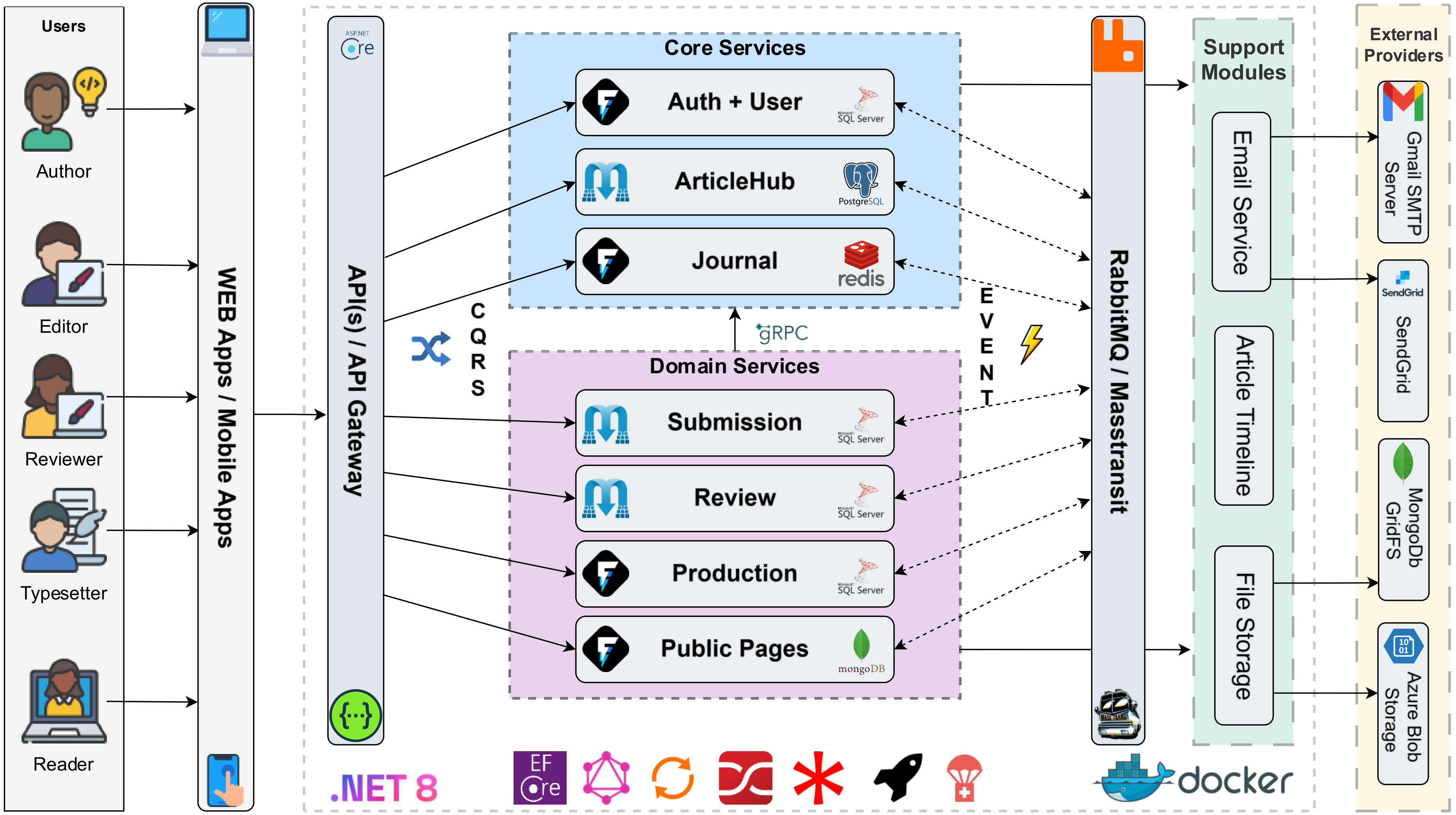
DotNet LabX

ArticleHub Microservice

with GraphQL, Hasura & EF Core

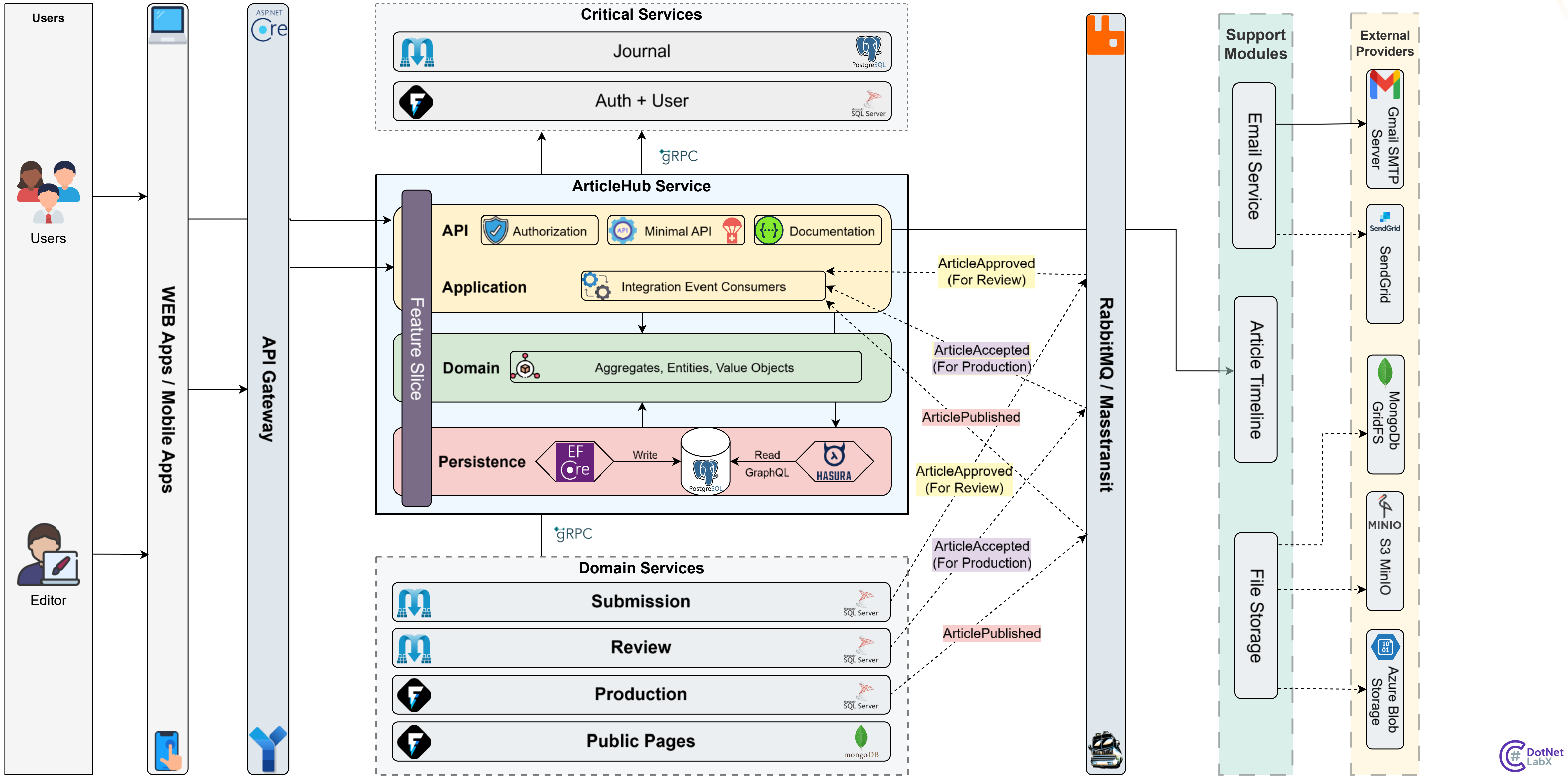
- Build **Minimal API Endpoints** powered by **Carter**
- Implement Queries **with GraphQL** and **Hasura**
- Configure domain persistence with **EF Core and Postgres**
- Consume integration events with **RabbitMQ** and **MassTransit**

High Level Architecture | C4 Level 2 (Container View)



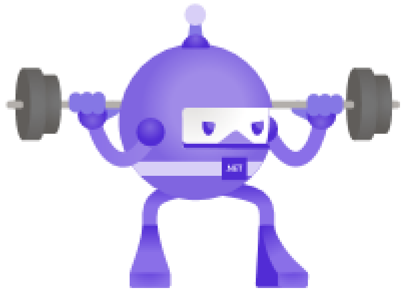
DotNet LabX

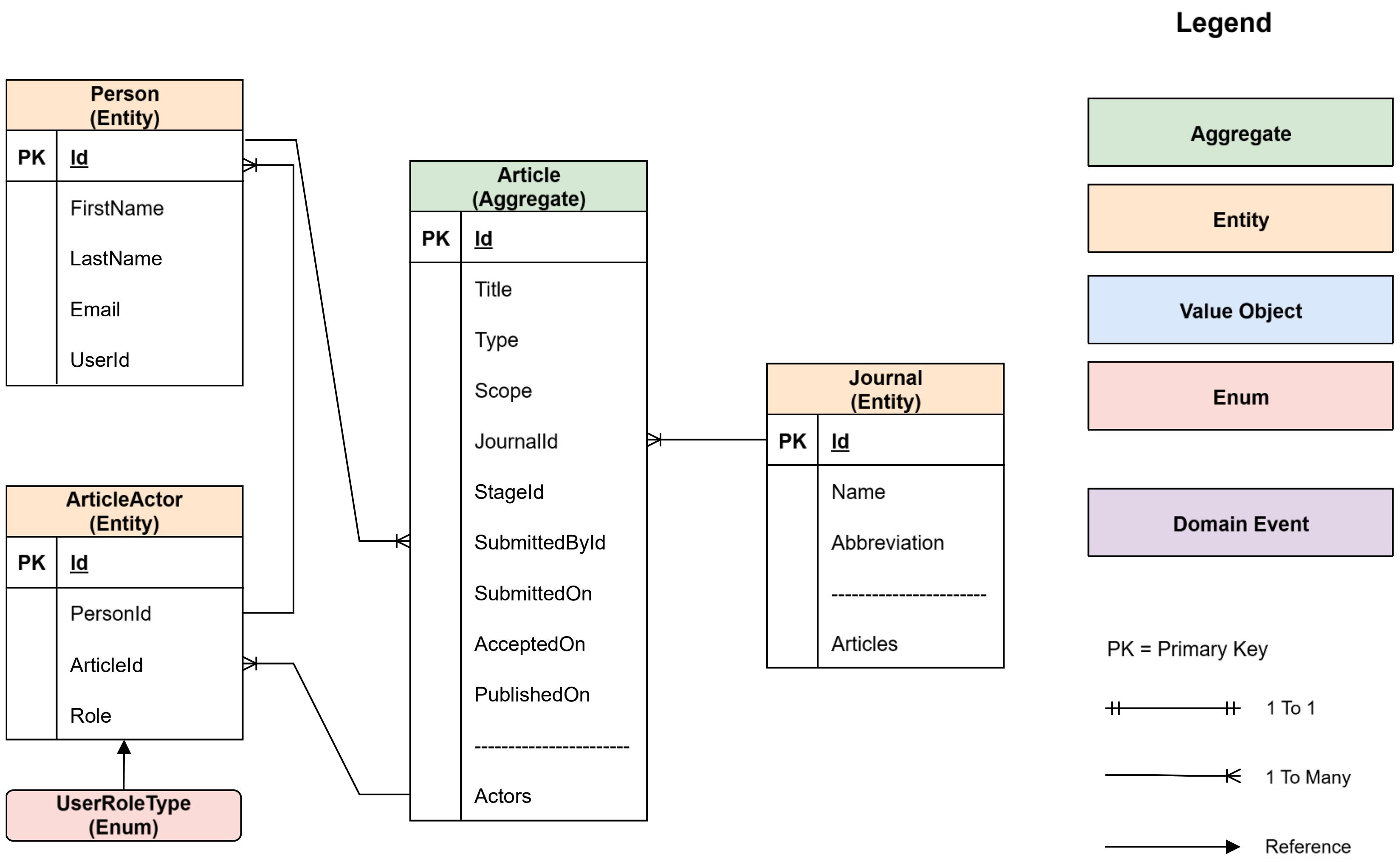
ArticleHub — High-Level Architecture (C4 Level 2)



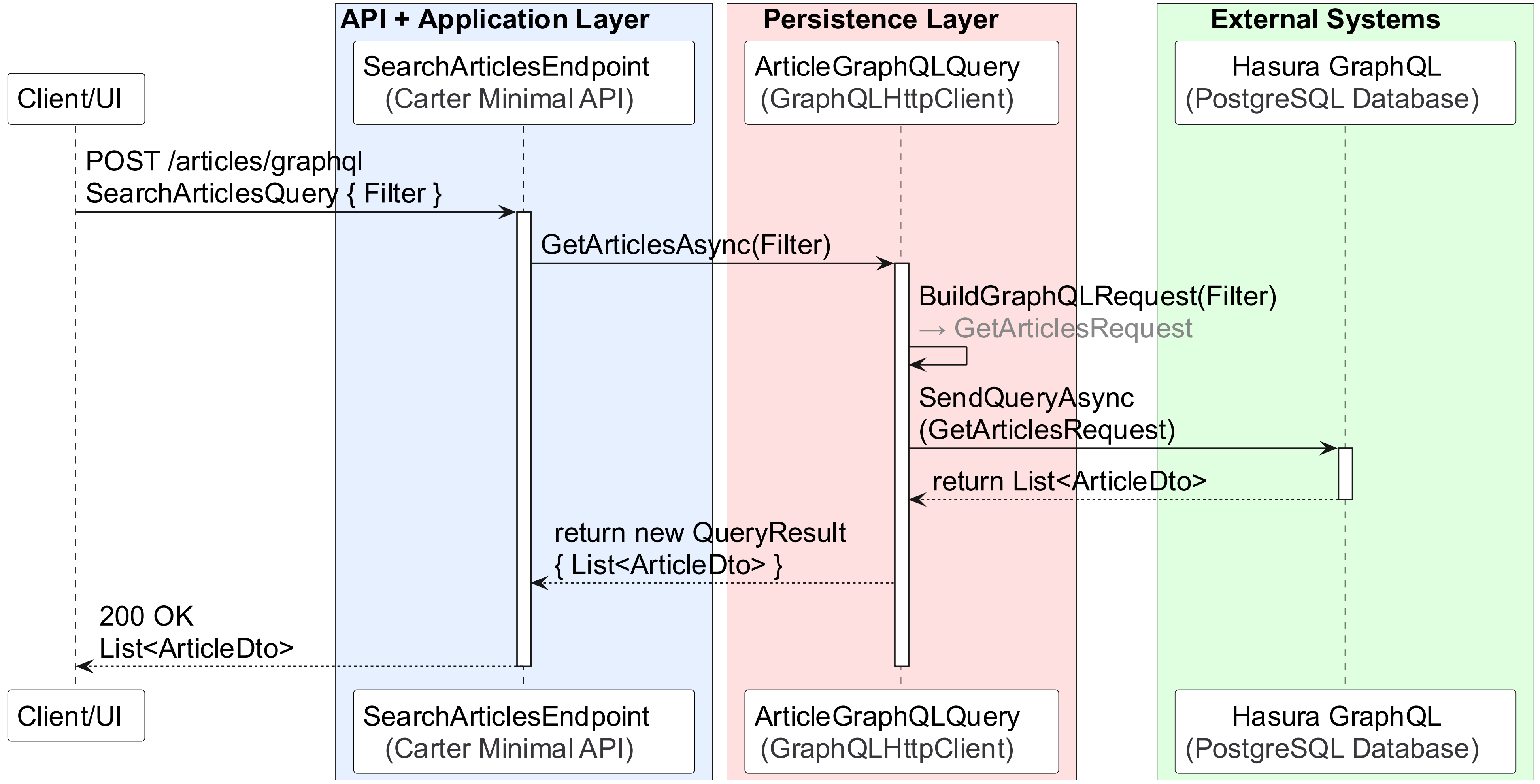
DotNet LabX

Tactical Design Diagram (DDD) - C4 Level 4

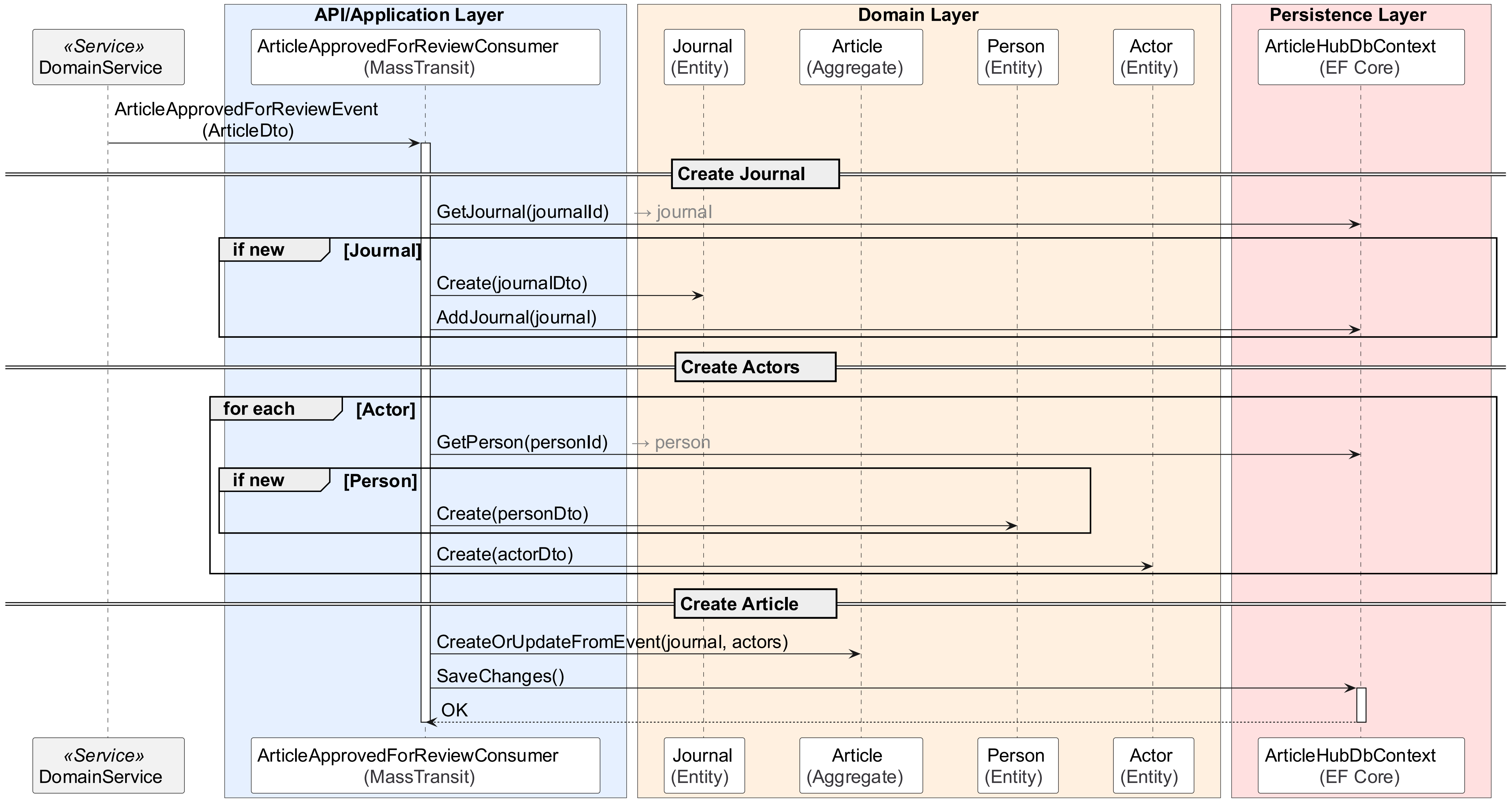




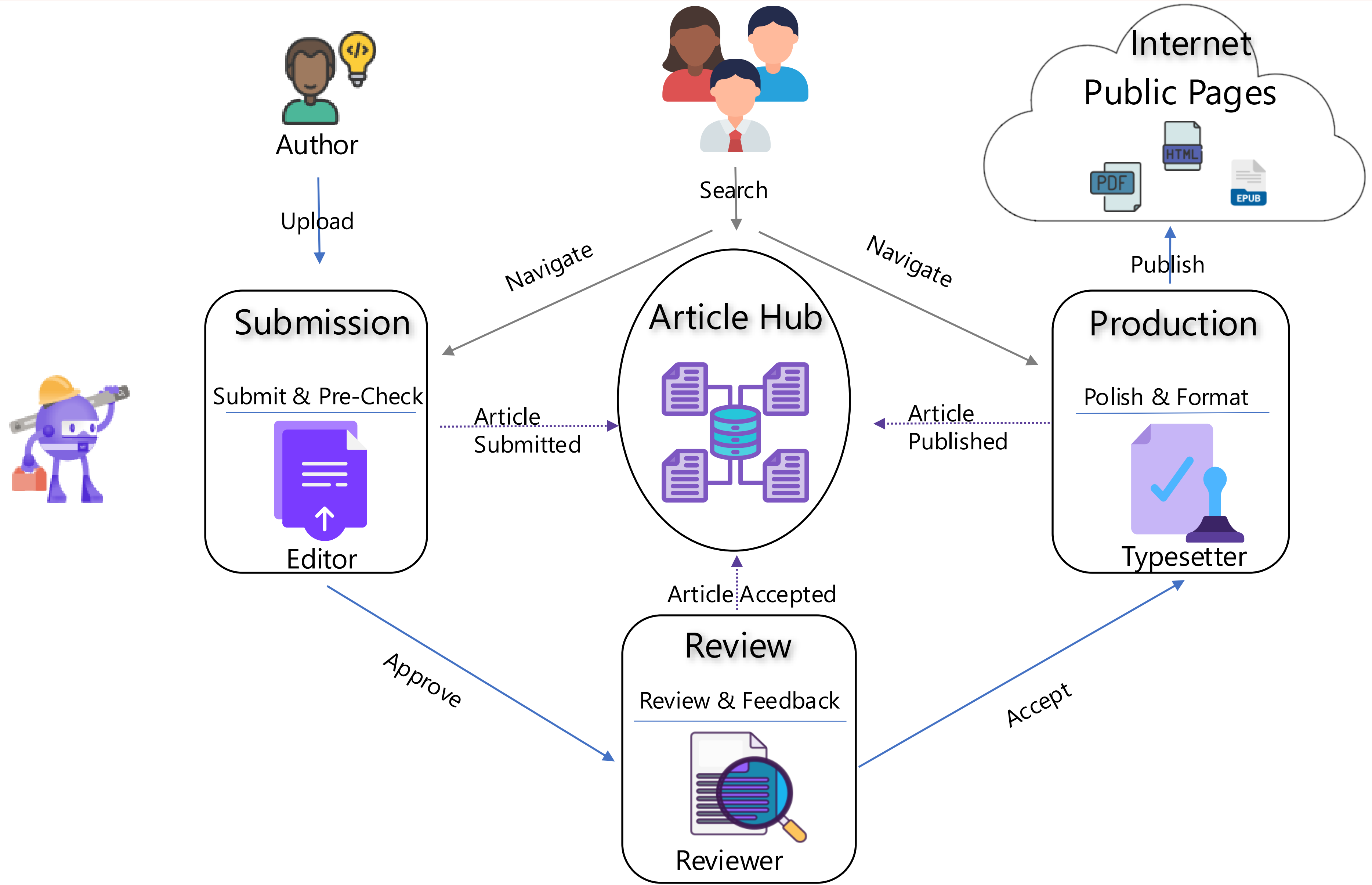
Search Articles – Sequence Diagram (C4 Level 3)


API + Application Layer
Persistence Layer
External Systems
Client/UI
SearchArticlesEndpoint (Carter Minimal API)
ArticleGraphQLQuery (GraphQLHttpClient)
Hasura GraphQL (PostgreSQL Database)
POST /articles/graphql SearchArticlesQuery { Filter }
GetArticlesAsync(Filter)
BuildGraphQLRequest(Filter) → GetArticlesRequest
SendQueryAsync (GetArticlesRequest)
return List<ArticleDto>
return new QueryResult { List<ArticleDto> }
200 OK List<ArticleDto>
Client/UI
SearchArticlesEndpoint (Carter Minimal API)
ArticleGraphQLQuery (GraphQLHttpClient)
Hasura GraphQL (PostgreSQL Database)

Feed Articles Consumers – Sequence Diagram (C4 Level 3)

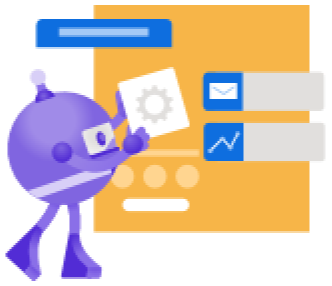


ArticleHub Workflow



User Stories

- **Search Articles**
  - As a **User**, I want to **search articles with filters** (author, journal, stage, date range, editor etc.) then order them, so I can quickly find the article(s) I need.
- **Get Article**
  - *As a* **User**, *I want* **to view the details of an article** and navigation link, *so that* I can open it in the right service.
- **Get Article Timeline**
  - *As an* **Editor**, I want **to view an article's timeline of key transitions** (with timestamps) so I can track progress and audit changes.

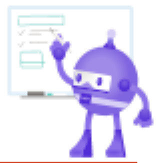


Endpoints

Name	Method	Roles	Endpoint
Search Articles	POST	USER	/api/articles/graphql
Get Article	GET	USER	/api/articles/{articleId}
Get Article Timeline	GET	EDIT	/api/articles/{articleId}/timeline
Get Journals	GET	USER	/api/journals
Get Catalogs	GET	USER	/api/articles/catalogs {ArticleStage, RoleType etc.}

USER	- Any Authenticated User
EDIT	- Editor

Requirements



Functional

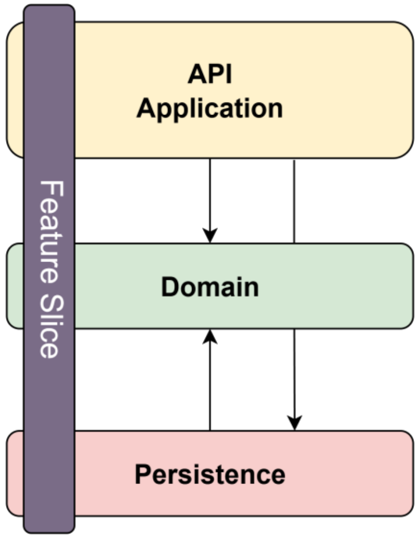
- **Ingest & Project**
  - **Consume integration events** from Submission/Review/Production (Created, Submitted, InReview, Accepted, InProduction, Published, Rejected)
- **Search & Read**
  - Proxy to **Hasura** (UI builds advanced GraphQL filters)
- **Smart Navigation**
  - Map **stage** → **service URL** (Submission / Review / Production)
- **Security**
  - **All authenticated users** can search, read & navigate to the **article**
  - Only the **editor** can read the **article timeline**

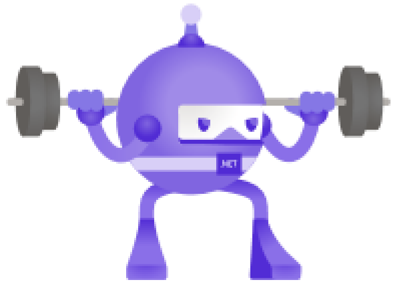
Non-Functional

- **Performance**
  - **Search** (GraphQL): ≤ **600** ms, (warm ≤ **300 ms**) - built for many users at once.
  - **Read** by ID: ≤ **300** ms - also high-traffic path
- **Caching (server-side):**
  - **Article detail:** 30–60 s; clear on article events
  - **Search pages:** 15–30 s; optional global bump on updates
  - **Autocomplete:** 5–15 min; refresh on change or expiry (Journal & Person Names)
- **Availability & Scale**
  - **99.9%** read availability
  - Steady load: **50 rps,** Burst capacity: **200 rps** (short spikes)
- **Consistency & Reliability**
  - Eventual consistency
  - **Process each event only once:** remember `EventId`, skip repeats(use Inbox pattern); update only if stage is newer.
- **Security**
  - **Row Level Security/Permission** (Hasura RLS)
- **Observability**
  - Metrics: events/sec, **GraphQL latency**, cache hit%

Clean Architecture

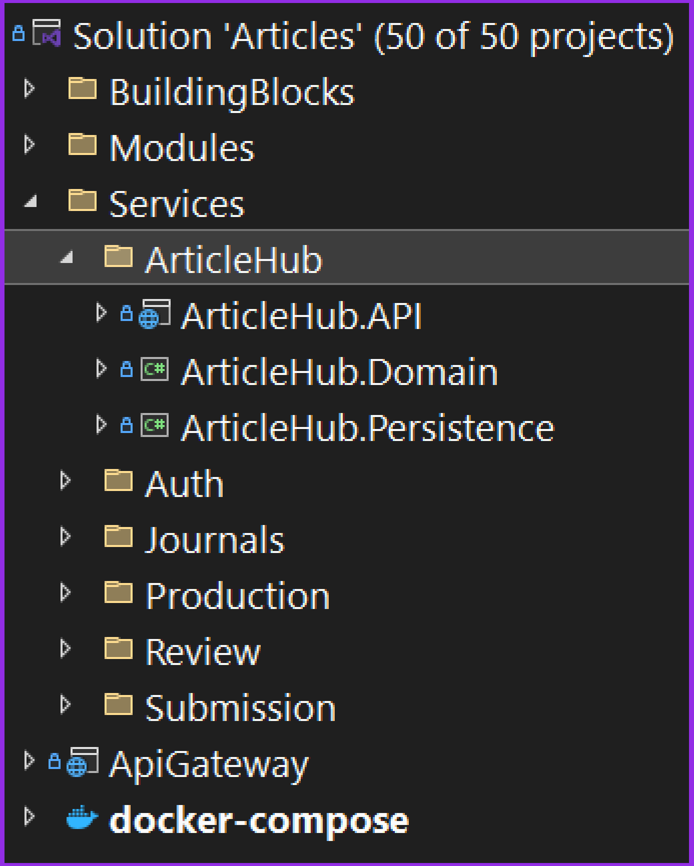
- **API / Application**
  - Endpoints with Carter Minimal API
  - Integrates Authorization & other middleware(s)
  - Coordinates the use case logic of the system.
  - Mapping with Mapster
  - Integration Event Consumers with MassTransit
  - **Depends on**:
    - `Domain` (for domain models)
    - `Persistence`(for DbContext, Repositories , ReadStore) & other `Infrastructure` integrations





- **Domain**
  - Core business logic and rules.
  - Contains:
    - **Aggregates** (Article, Role)
    - **Entities(**Journal, Person, ArticleActor)
  - **Completely isolated** — does not depend on any other layer.
- **Infrastructure / Persistence**
  - Handles all technical concerns and integration points.
  - Contains:
    - EF Core (DbContext, Repositories)
    - GraphQL ReadStores with Hasura
  - Implements contracts or patterns defined in Application or Domain.
  - **Depends on**: `Domain`

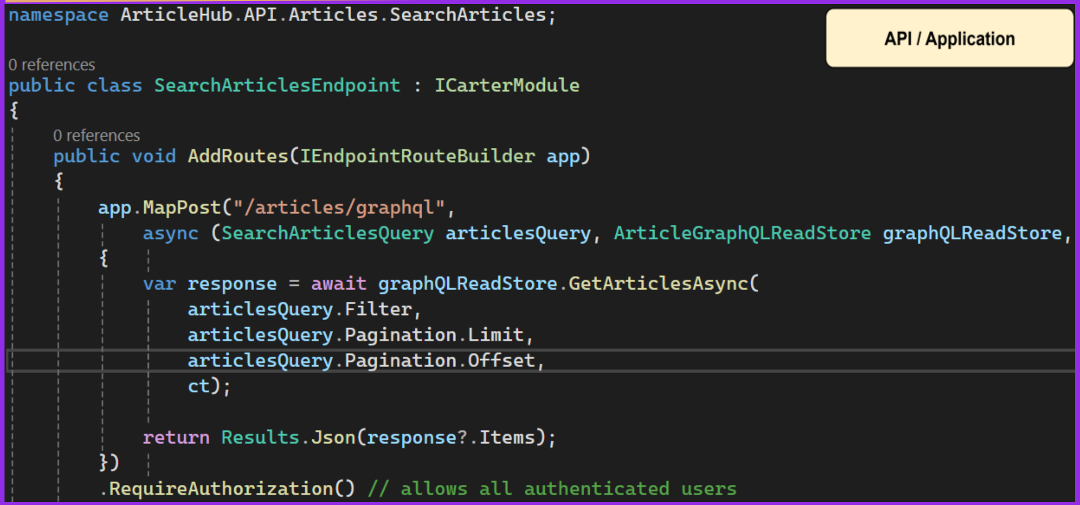
ArticleHub – Structure



Solution 'Articles' (50 of 50 projects)
- BuildingBlocks
- Modules
- Services
  - ArticleHub
    - ArticleHub.API
    - ArticleHub.Domain
    - ArticleHub.Persistence
  - Auth
  - Journals
  - Production
  - Review
  - Submission
- ApiGateway
- **docker-compose**

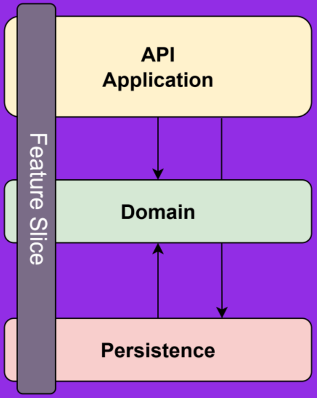
- **Clean Architecture Projects Setup**
  - Create the solution and 3 projects: **API**, **Domain**, **Persistence**
  - Add project references and essential **NuGet packages**
- **Designing the Domain Model**
  - Define Aggregates, Entities, Value Objects, Events and domain behavior
- **Configuring Persistence**
  - Set up **DbContext** and EF Core configuration
  - Create the **first migration** and apply it
  - Setup Hasura with HttpClient for GraphQL queries
- **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**

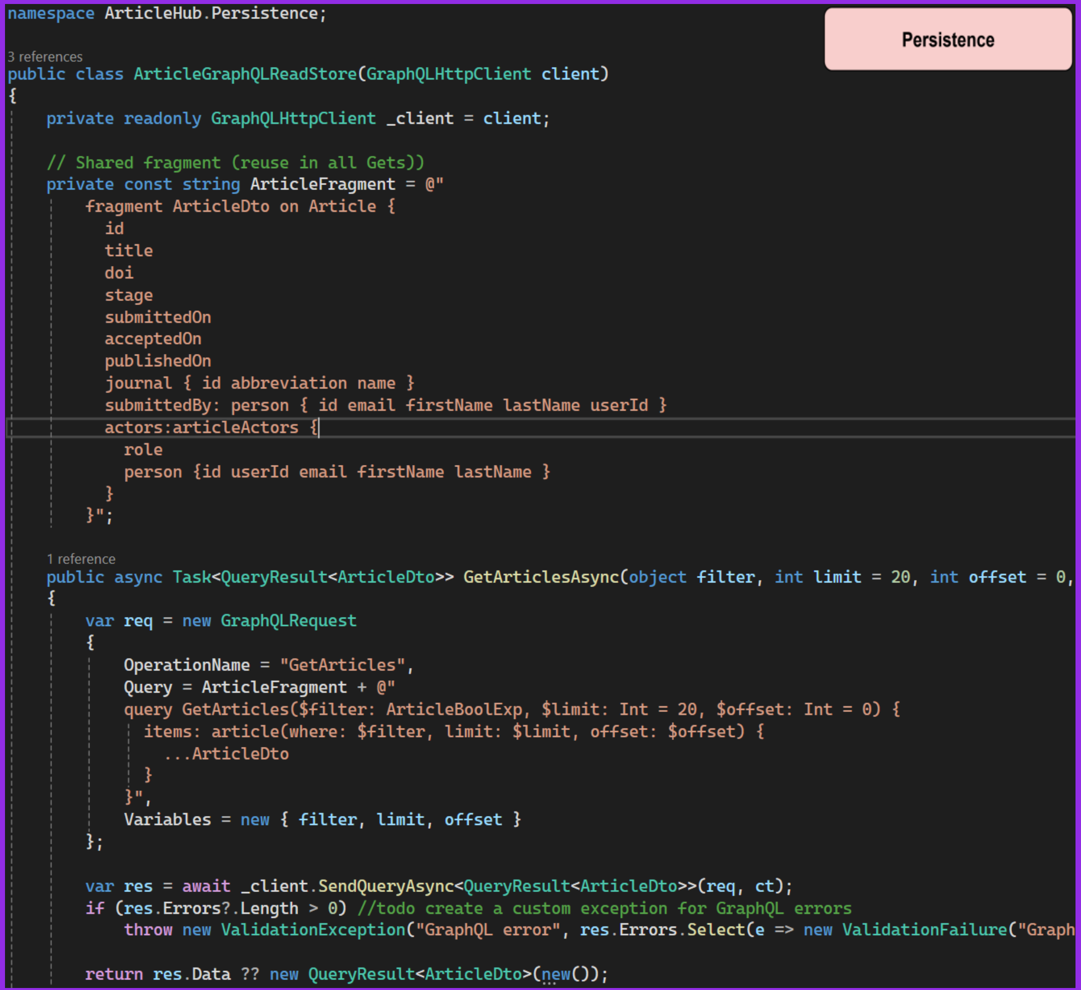
ArticleHub – Search Articles Feature



API / Application

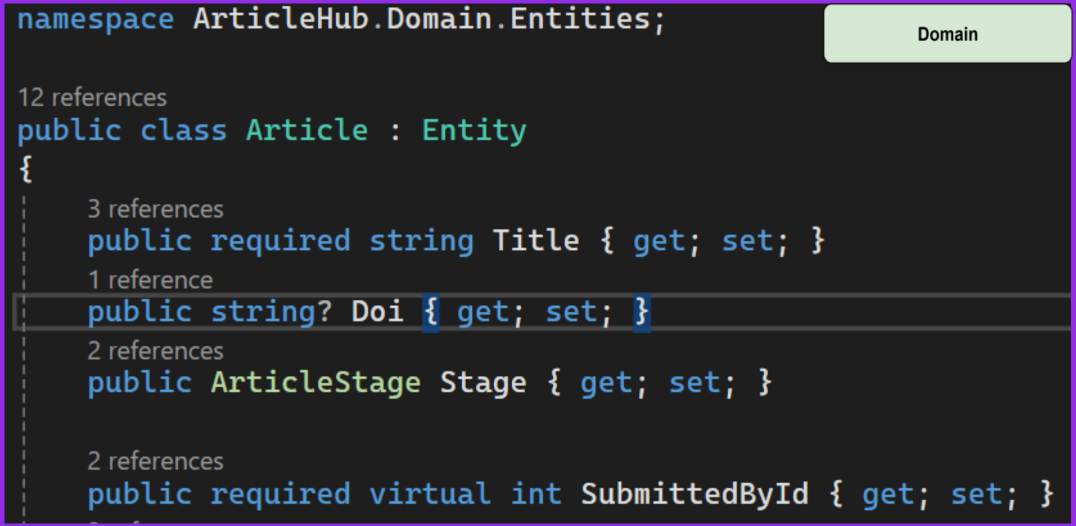
```
namespace ArticleHub.API.Articles.SearchArticles;

0 references
public class SearchArticlesEndpoint : ICarterModule
{
    0 references
    public void AddRoutes(IEndpointRouteBuilder app)
    {
        app.MapPost("/articles/graphql",
            async (SearchArticlesQuery articlesQuery, ArticleGraphQLReadStore graphQLReadStore,
        {
            var response = await graphQLReadStore.GetArticlesAsync(
                articlesQuery.Filter,
                articlesQuery.Pagination.Limit,
                articlesQuery.Pagination.Offset,
                ct);

            return Results.Json(response?.Items);
        })
        .RequireAuthorization() // allows all authenticated users
```





Persistence

```
namespace ArticleHub.Persistence;

3 references
public class ArticleGraphQLReadStore(GraphQLHttpClient client)
{
    private readonly GraphQLHttpClient _client = client;

    // Shared fragment (reuse in all Gets))
    private const string ArticleFragment = @"
      fragment ArticleDto on Article {
        id
        title
        doi
        stage
        submittedOn
        acceptedOn
        publishedOn
        journal { id abbreviation name }
        submittedBy: person { id email firstName lastName userId }
        actors:articleActors {
          role
          person {id userId email firstName lastName }
        }
      }";

    1 reference
    public async Task<QueryResult<ArticleDto>> GetArticlesAsync(object filter, int limit = 20, int offset = 0,
    {
        var req = new GraphQLRequest
        {
            OperationName = "GetArticles",
            Query = ArticleFragment + @"
            query GetArticles($filter: ArticleBoolExp, $limit: Int = 20, $offset: Int = 0) {
              items: article(where: $filter, limit: $limit, offset: $offset) {
                ...ArticleDto
              }
            }",
            Variables = new { filter, limit, offset }
        };

        var res = await _client.SendQueryAsync<QueryResult<ArticleDto>>(req, ct);
        if (res.Errors?.Length > 0) //todo create a custom exception for GraphQL errors
            throw new ValidationException("GraphQL error", res.Errors.Select(e => new ValidationFailure("Graph

        return res.Data ?? new QueryResult<ArticleDto>(new());
```



Domain

```
namespace ArticleHub.Domain.Entities;

12 references
public class Article : Entity
{
    3 references
    public required string Title { get; set; }
    1 reference
    public string? Doi { get; set; }
    2 references
    public ArticleStage Stage { get; set; }

    2 references
    public required virtual int SubmittedById { get; set; }
```