Auth Microservice Handbook

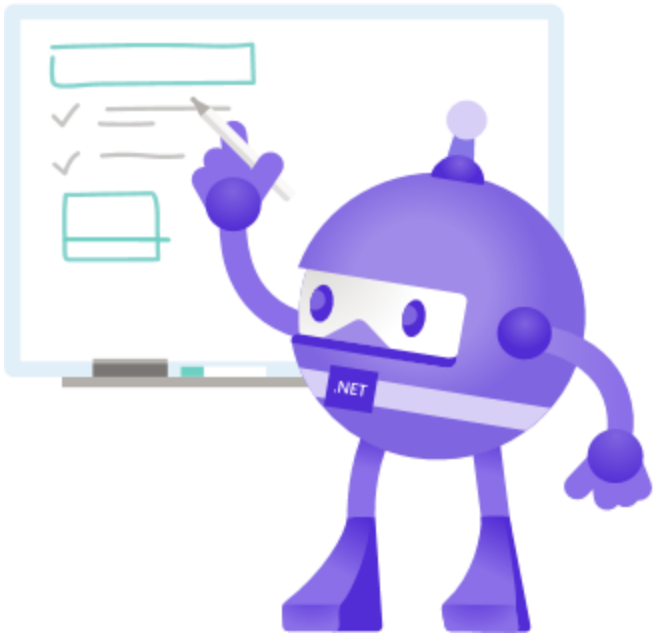
This handbook is a visual companion to the Auth Microservice module of the course.

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



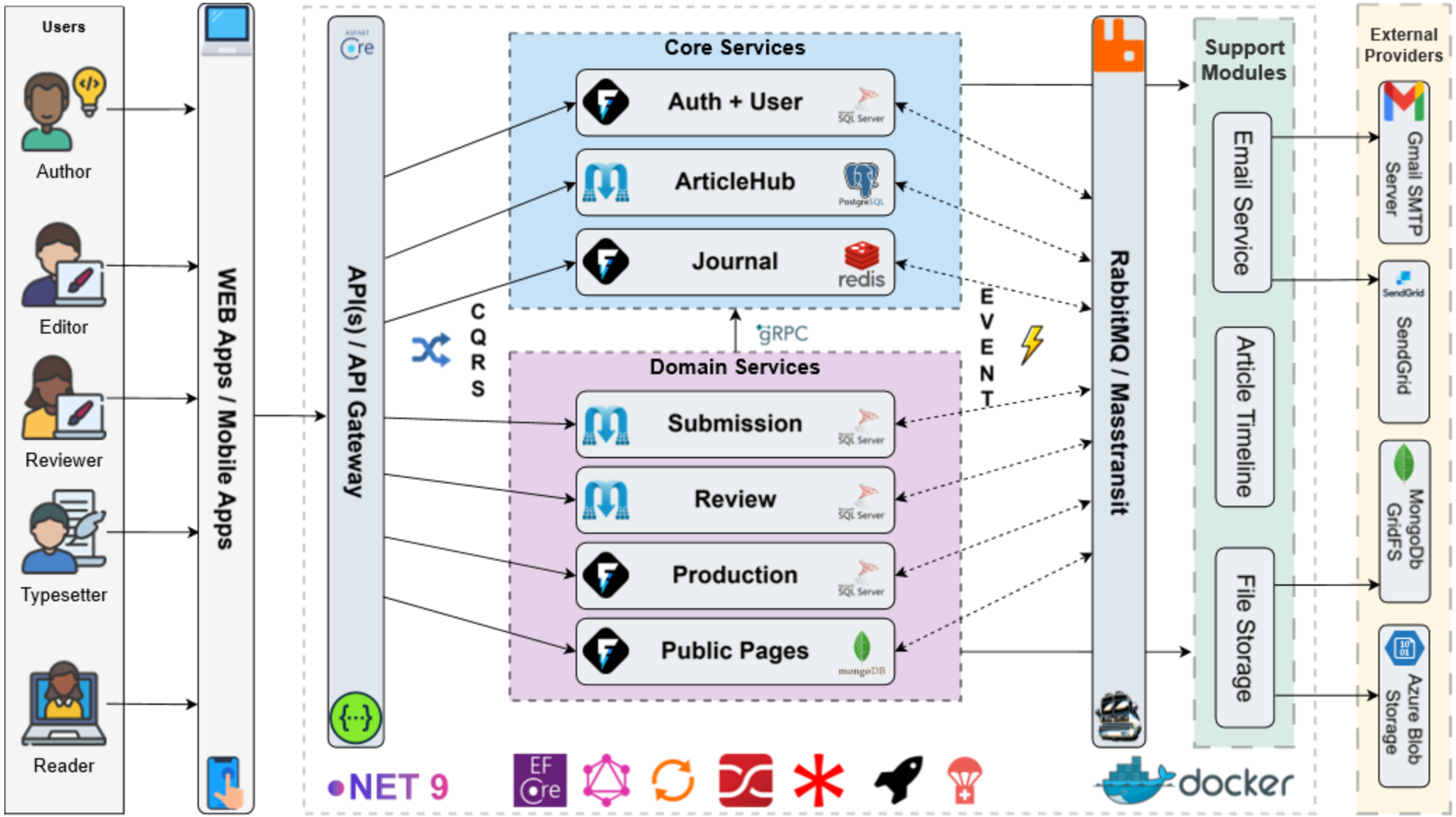
DotNet LabX

Auth Microservice

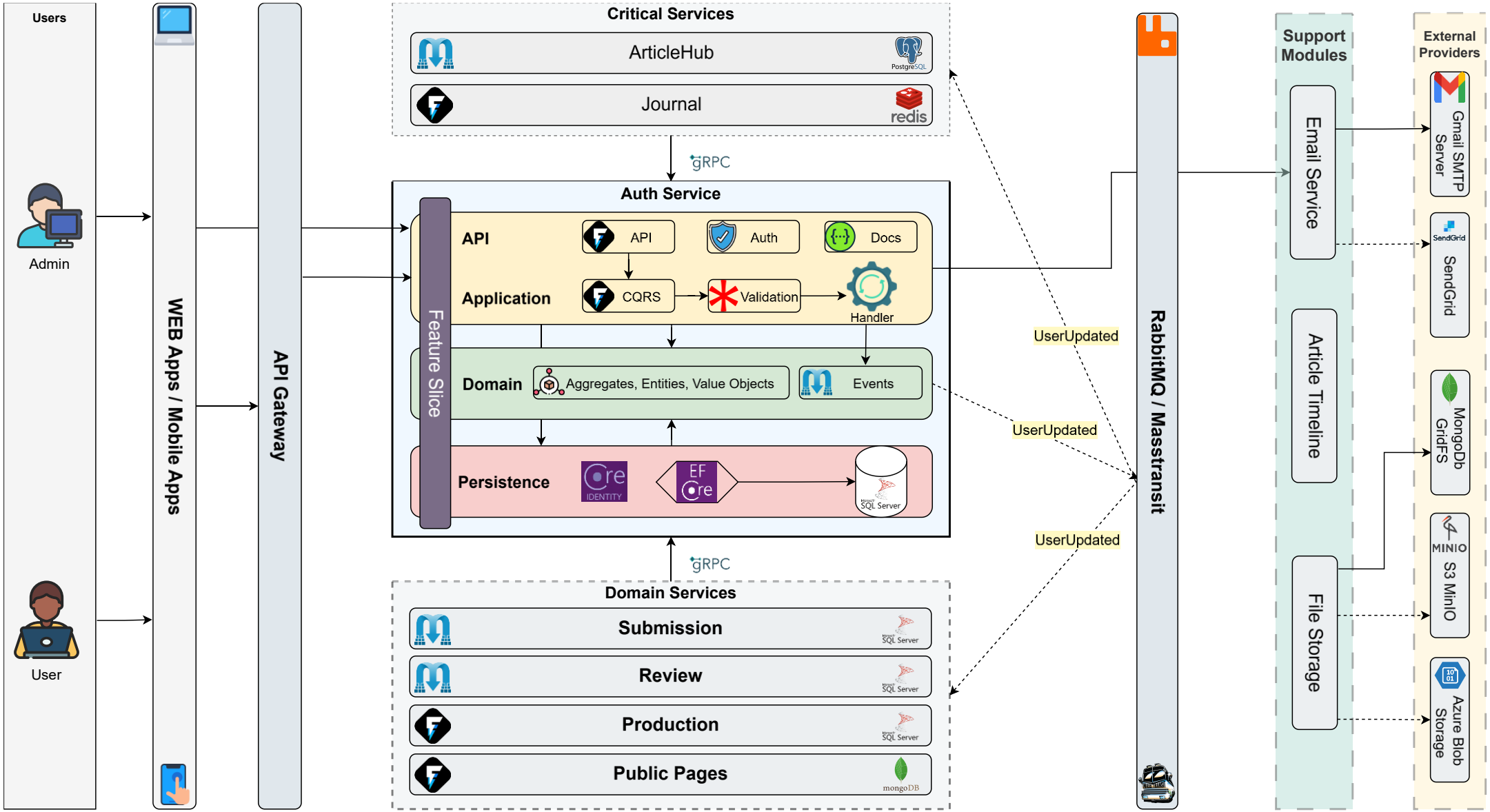
with FastEndpoints, JWT Authentication & Role-Based Access Control

- Build API Endpoints & implement CQRS with **FastEndpoints**
- **FluentValidation** integration with FastEndpoints
- Generate **JWT & Refresh Tokens** using a Factory
- Send **confirmation emails** via **domain event handlers**
- Manage Users & Roles with **ASP.NET Identity**
- Expose Person data through a **gRPC** service (**protobuf-net**)
- Configure Domain **Persistence** with **EF Core**

High Level Architecture | C4 Level 2 (Container View)

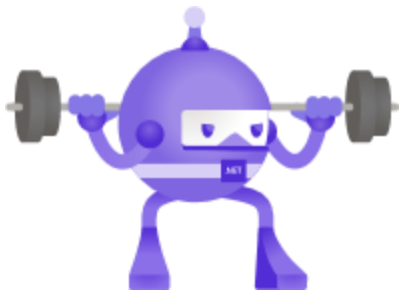


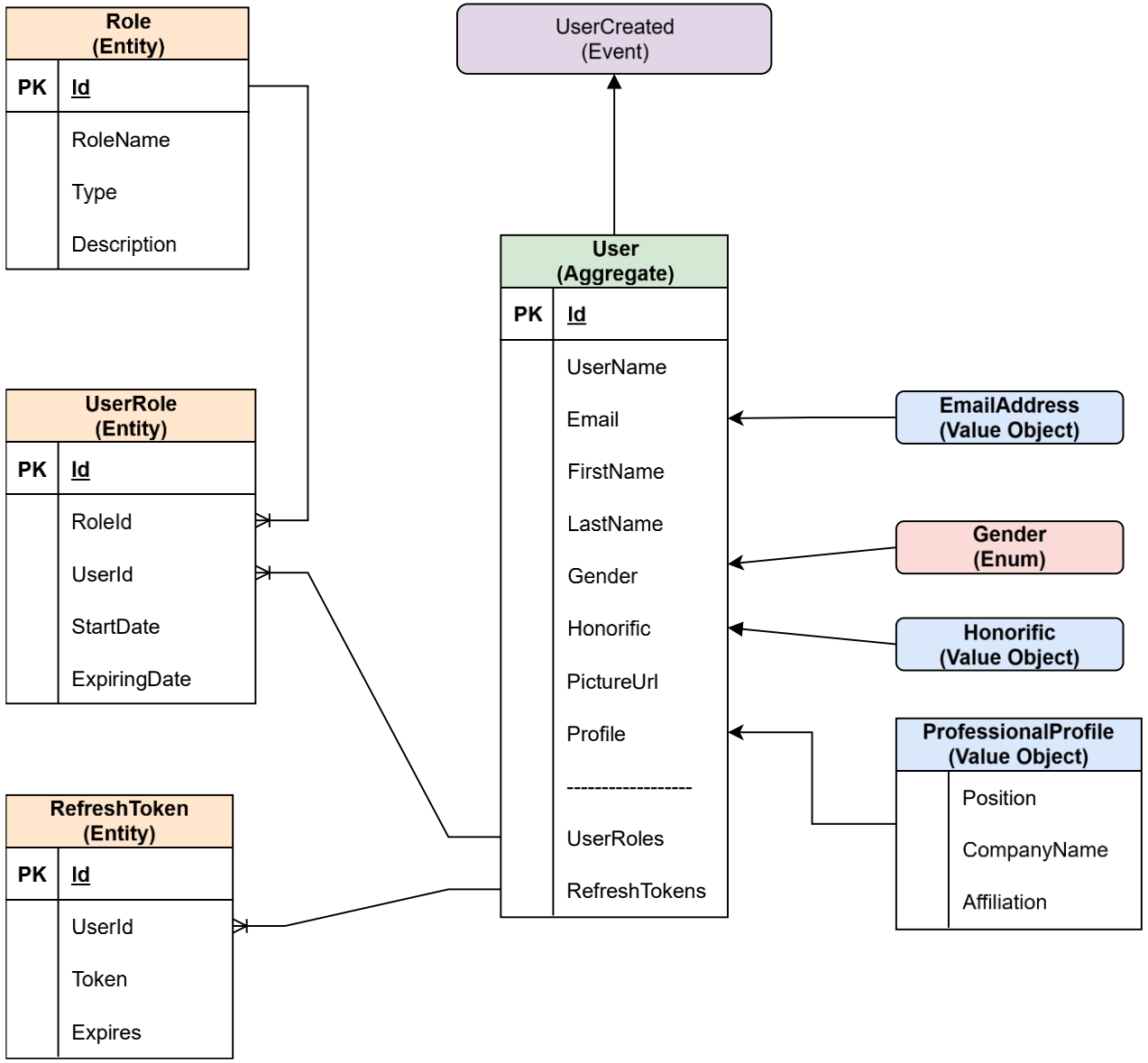
Auth Architecture – C4 Level 2 (Container View)

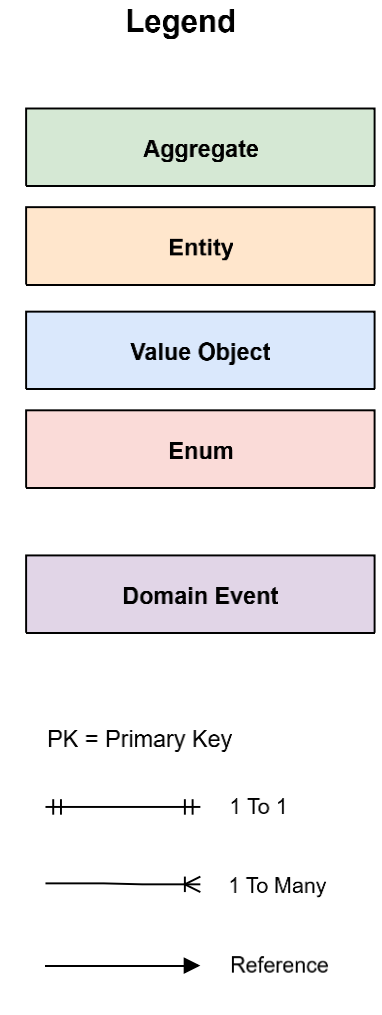


DotNet LabX

Tactical Design Diagram (DDD) - C4 Level 4

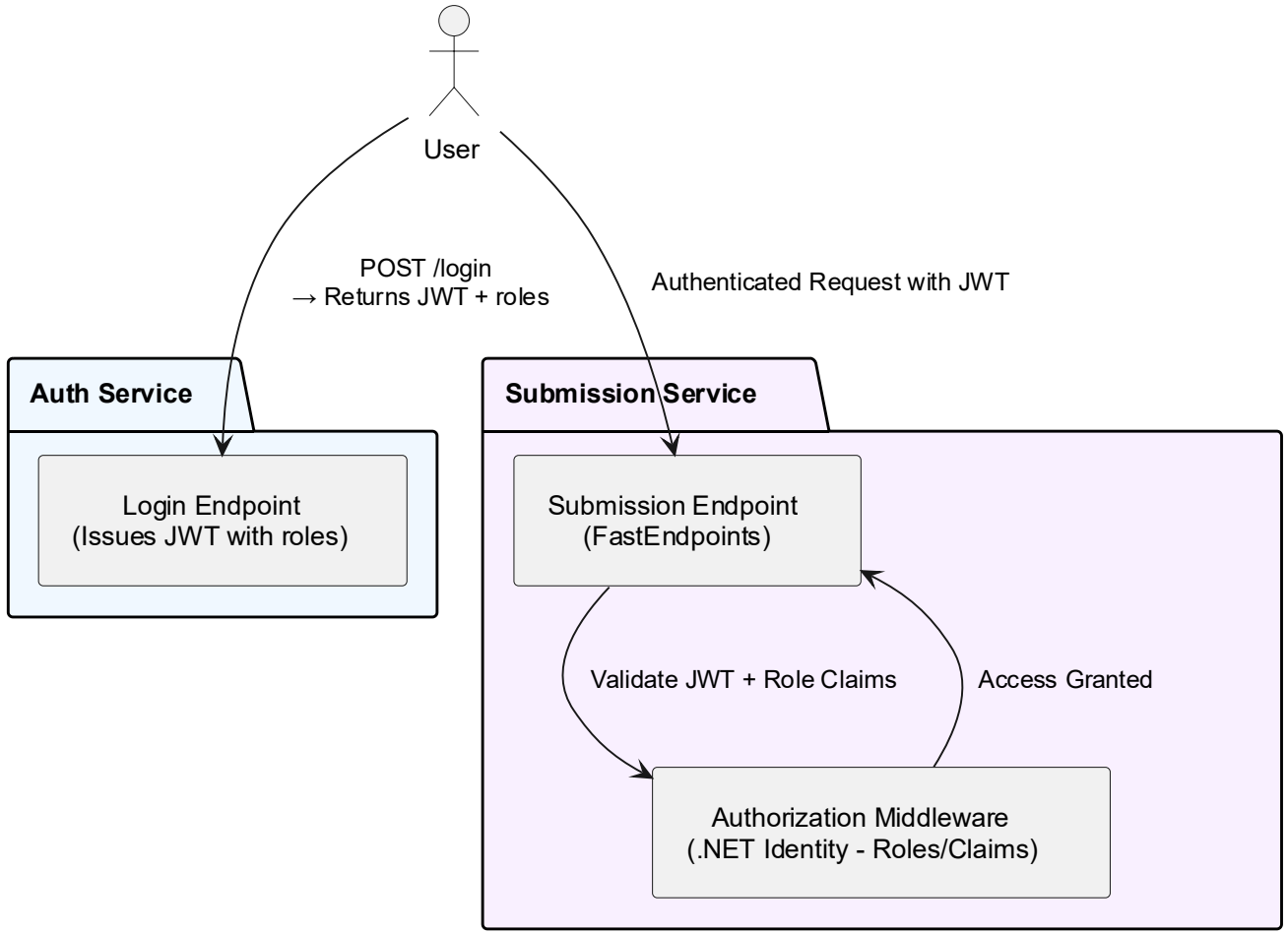






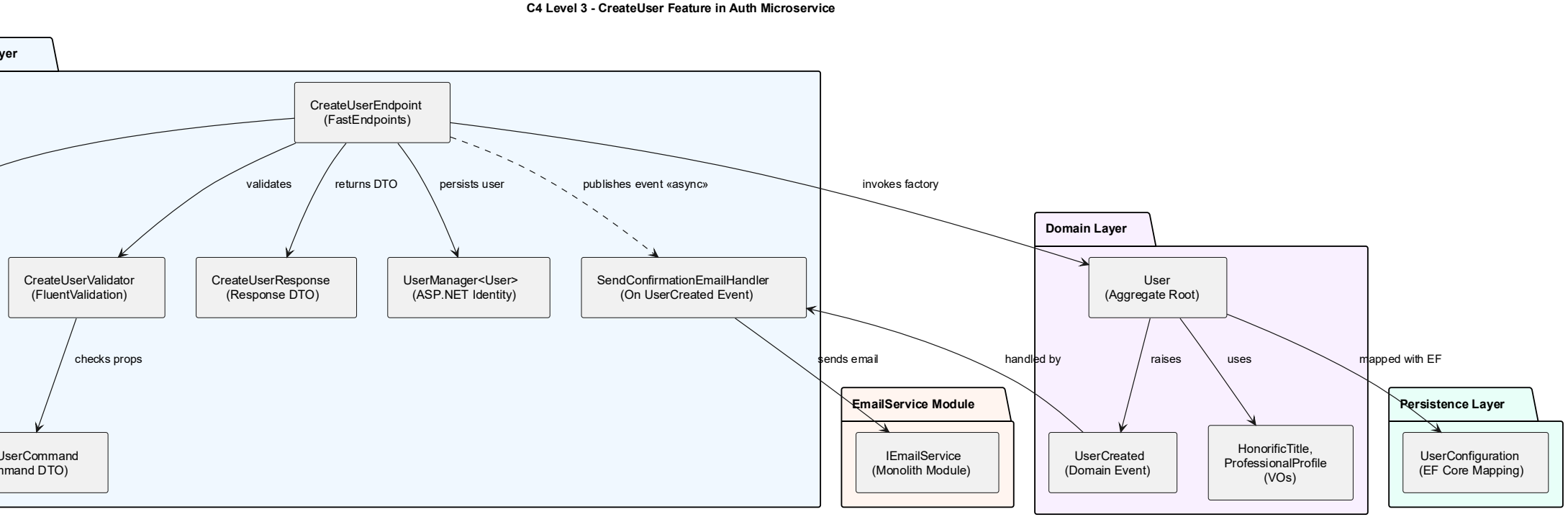
DotNet LabX

Authentication + Authorization Flow – C4 Level 2



User logs in via AuthService and accesses protected endpoints using JWT + roles via .NET Identity

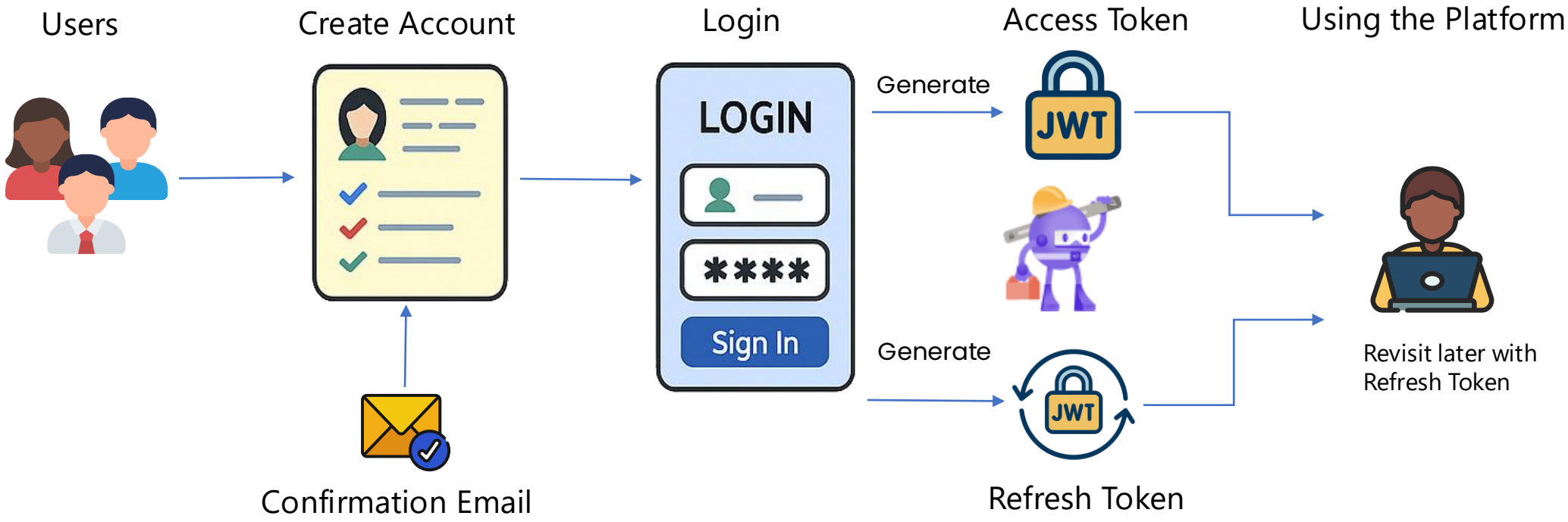
Create User - C4 Level 3 (Component)



Shows internal components and interactions of CreateUser within the Auth service.

DotNet LabX

Auth Workflow



- Centralized service for authentication & authorization
- Issues JWT Token + Refresh Token
- Controls Access via role-based authorization
- Required by all services in the system

DotNet LabX

User Stories

- **Create User**

  As an **Admin**, I want to **create a new user and assign roles**, so that the user can verify their identity and activate their account.

- **Set Password**

  As a **User**, I want to **set my password using a secure token**, so that I can activate my account or complete a password reset.

- **Request Password Reset**

  As a **User who forgot my password**, I want to **request a reset link via email**,
  so that I can regain access to my account.

- **Login**

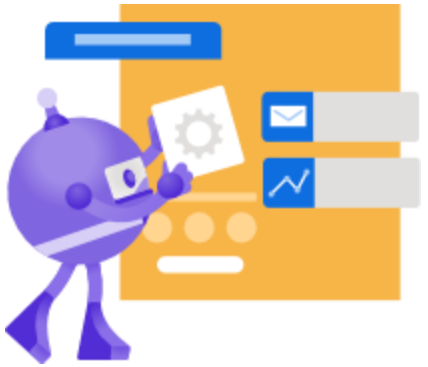
  As a **User**, I want to **log in with my email and password**, so that I can access the platform and use its features.

- **Refresh Token**

  As a **User**, I want to **stay logged in without having to log in again**,
  so that I can continue using the platform seamlessly.

- **Get User Info**

  As a **Service**, I want to **fetch user information by ID**, so that I can enforce business rules or show user details.



DotNet LabX

Endpoints

Name	Method	Roles	Endpoint
Create User	POST	ADMIN	/api/users
Set Password	POST	-	/api/password
Request Password	POST	-	/api/password:request
Login	POST	-	/api/login
Refresh Token	POST	-	/api/token:refresh
Get User	gRPC	SYSTEM	UserService.GetUserById(userId)
Optional(Not implemented)			
Get User	GET	AUTH	/api/users/{userId}
Update User	PUT	AUTH	/api/users/{userId}

DotNet LabX

Requirements

Functional

Create User (by Admin)

- Required fields → FirstName, LastName, Email, Roles, Personal & Professional Details
- Send Email confirmation? → Yes

Set Password

- Triggered by email link with token
- Required fields → Token, NewPassword, ConfirmPassword
- Constraints → Min length, complexity rules

Request Password Reset

- Input → Email
- Output → Email & Secure token

Login

- Required fields → Email, Password
- Output → JWT Access Token + Refresh Token

Refresh Token

- Input → Refresh Token
- Output → New JWT Access Token

Get User Info (gRPC)

- Input → UserId
- Output → UserId, FullName, Email, Personal & Professional Details

Non-Functional

Authentication

- Token type → JWT (Access + Refresh)
- Expiration → Access: 15 min, Refresh: 7 days

Password Management

- Storage → Hashed using ASP.NET Identity default
- Requirements → Min 8 chars, 1 uppercase, 1 digit, 1 special char

Security

- Role-based access enforced via attributes or policies One role only, ADMIN
- Some endpoints allow anonymous access and some require only authentication (no specific role needed)

Scalability

- Stateless endpoints (except refresh token storage)
- Compatible with horizontal scaling

Performance

- Login and token generation under 200ms (target)
- gRPC responses within 100ms (target)

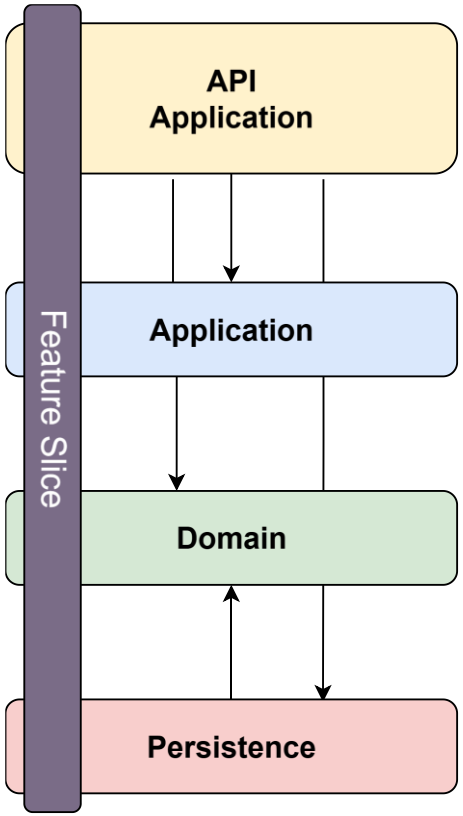
gRPC User Info

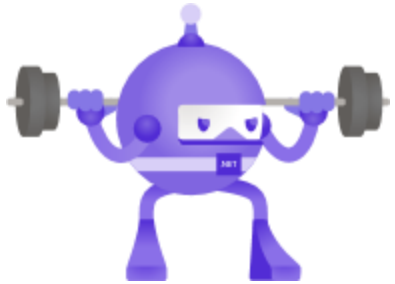
- Used only internally (SYSTEM role)
- Secured via gRPC headers + internal network access

DotNet LabX

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
- **Application**
  - Creates JWT and refresh tokens

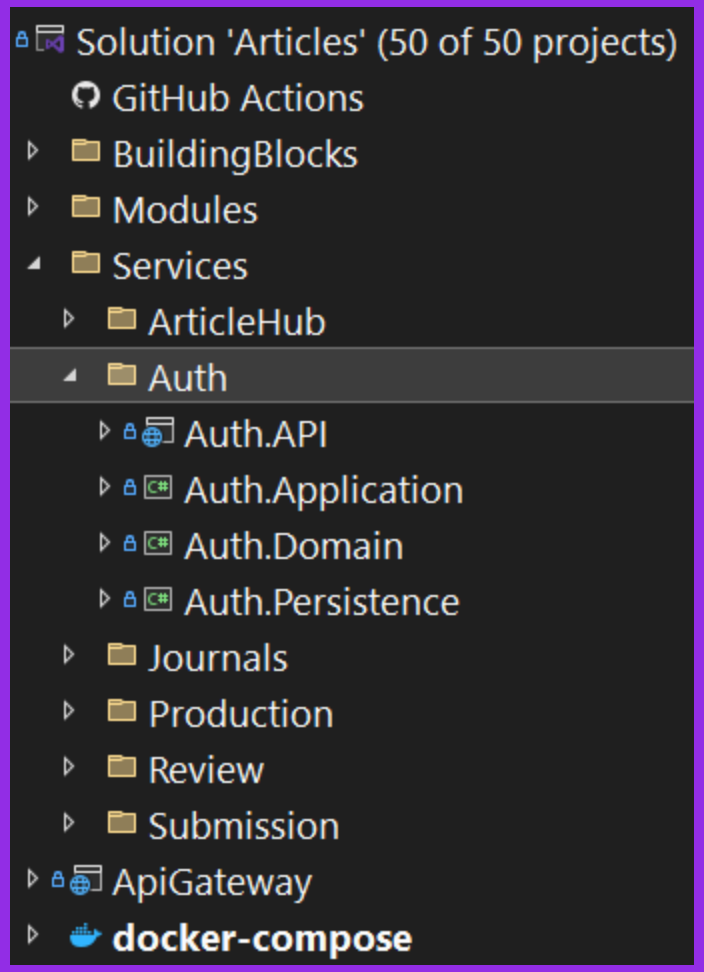




- **Domain**
  - Core business logic and rules.
  - Contains:
    - **Aggregates** (User, Role)
    - **Entities(**UserRole, RefreshToken)
    - **Value Objects**(HonorificTitle, ProfessionalProfile)
    - **Domain Events**(UserCreated, UserUpdated)
  - Domain Functions – business rules and behavior 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)
    - References to shared modules (e.g., EmailService)
  - Implements contracts or patterns defined in Application or Domain.
  - **Depends on**: `Domain`

DotNet LabX

Auth – 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 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**

DotNet LabX

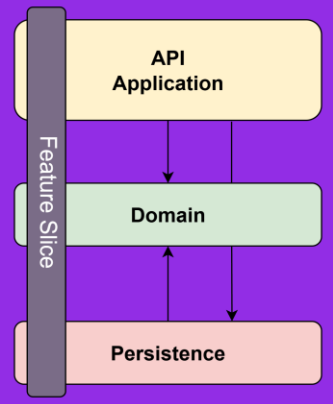
Auth – Reset Password Feature

API / Application

```
namespace Auth.API.Features.Users.SetPassword;

1 reference
public partial class SetPasswordEndpoint(UserManager<User> _userManager)
    : Endpoint<SetPasswordCommand, SetPasswordResponse>
{
    0 references
    public override async Task HandleAsync(SetPasswordCommand command, CancellationToken ct)
    {
        var user = await _userManager.FindByEmailAsync(command.Email);
        if (user == null)
            throw new BadRequestException($"User with email {command.Email} doesn't exist");

        var result = await _userManager.ResetPasswordAsync(
            user, command.TwoFactorToken, command.NewPassword);
        if (!result.Succeeded)
            throw new BadRequestException($"Unable to change password for {command.Email}");

        await Send.OkAsync(new SetPasswordResponse(command.Email));
    }
}
```

```
namespace Auth.API.Features.Users.SetPassword;

3 references
public record SetPasswordCommand(
    string Email, string NewPassword, string ConfirmPassword, string TwoFactorToken);
3 references
public record SetPasswordResponse(string Email);
```

```
public class SetPasswordValidator : AbstractValidator<SetPasswordCommand>
{
    0 references
    public SetPasswordValidator()
    {
        RuleFor(c => c.Email).NotEmpty().EmailAddress();
        RuleFor(c => c.NewPassword).NotEmpty();
        RuleFor(c => c.ConfirmPassword).NotEmpty();
        RuleFor(c => c.TwoFactorToken).NotEmpty();
        RuleFor(c => c.NewPassword).Must(
            (command, value) => command.NewPassword == command.ConfirmPassword)
            .WithMessage("Passwords doesn't match");
    }
}
```



Domain

```
namespace Auth.Domain.Users;

30 references
public partial class User : IdentityUser<int>, IAggregateRoot
{
    0 references
    public DateTime RegistrationDate { get; init; } = DateTime.UtcNow;
    0 references
    public DateTime? LastLogin { get; set; }

    4 references
    public string FullName => Person.FullName;
    2 references
    public int PersonId { get; set; }
    7 references
    public Person Person { get; init; } = null!;

    private List<UserRole> _userRoles = new List<UserRole>();
    1 reference
    public virtual IReadOnlyList<UserRole> UserRoles => _userRoles;

    private List<RefreshToken> _refreshTokens = new();
    4 references
    public virtual IReadOnlyList<RefreshToken> RefreshTokens => _refreshTokens;
```

Persistence

```
namespace Auth.Persistence.EntityConfigurations;

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

        builder.HasMany(p => p.UserRoles).WithOne().HasForeignKey(p => p.UserId)
            .IsRequired().OnDelete(DeleteBehavior.Cascade);
        builder.HasMany(p => p.RefreshTokens).WithOne().HasForeignKey(p => p.UserId)
            .IsRequired().OnDelete(DeleteBehavior.Cascade);

        builder.HasOne(u => u.Person).WithOne(p => p.User)
            .HasForeignKey<User>(u => u.PersonId)
            .IsRequired()
            .OnDelete(DeleteBehavior.Restrict);
    }
}
```

DotNet LabX