Files

Fedlech

Name	Last update
.devcontainer	1 day ago
src	16 hours ago
tests	1 day ago
.dockerignore	5 days ago
.editorconfig	1 week ago
.env	5 days ago
.env.example.prod	5 days ago
.gitignore	5 days ago
.gitlab-ci.yml	5 days ago
Directory.Build.props	1 week ago
Directory.Packages.props	1 week ago
Fedlech.slnx	1 week ago
README.md	16 hours ago
docker-compose.prod.yml	16 hours ago
docker-compose.yml	16 hours ago
global.json	1 week ago

README.md

Fedlech - Event Management Application

A clean architecture event management application built with .NET, React, and PostgreSQL. Featuring real-time updates via SignalR, JWT authentication, and a fully typed TypeScript frontend.

Architecture

This project follows **Clean Architecture** principles with layer separation:

- **Domain Layer**: Core business logic, entities, value objects, and domain events
- **Application Layer**: CQRS pattern with MediatR, use cases, DTOs, and validators
- **Infrastructure Layer**: Data persistence, external services, and security implementation
- **Web Layer**: Minimal API endpoints, middleware, and HTTP configuration

For this project, a template was used. For more information regarding the architecture, the following page can be visited: https://cleanarchitecture.jasontaylor.dev/

Quick Start with Docker

Development Environment

```
# Start all services (default docker-compose.yml for development)
docker compose up --build

# Access the application
# Frontend: http://localhost:3000
# Backend API: http://localhost:5000/api
# API Docs: http://localhost:5000/swagger
```

Production Environment

```
# Create production environment file and fill in your values
cp .env.example.prod .env  # or create .env manually

# Start services
docker compose -f docker-compose.prod.yml up -d

# View logs
docker compose -f docker-compose.prod.yml logs -f
```

Local Development Setup

Prerequisites

- **.NET SDK 10.0.201** (Download)
- **PostgreSQL 18** (Download)
- **Node.js 25** (Download)

Backend Setup

```
dotnet restore
dotnet run --project src/Web
# API:     http://localhost:5000/api
# Swagger: http://localhost:5000/swagger
```

> The local development setup uses an in-memory database by default.

Frontend Setup

```
cd src/Web/ClientApp
npm install
npm run dev
# Frontend: http://localhost:5173
```

Authentication & Authorization

JWT Configuration

The application uses JWT Bearer tokens for API authentication:

```
{
  "Jwt": {
    "Secret": "your-secret-key-here-min-32-characters-long",
    "Issuer": "http://localhost:5000",
    "Audience": "FedlechApiClient",
    "ExpirationMinutes": 60
  },
  "RefreshToken": {
    "ExpirationDays": 7
  }
}
```

API Authentication Flow

1. **Register**: POST `/api/auth/register` with email/password
2. **Login**: POST `/api/auth/login` returns access + refresh tokens
3. **Access Protected Resources**: Include `Authorization: Bearer {token}` header
4. **Refresh Token**: POST `/api/auth/refresh` with refresh token to get new access token

Real-Time Updates with SignalR

The application uses SignalR for real-time WebSocket communication for instant notifications and updates across all connected clients.

SignalR behind Traefik (Production)

When running multiple `webapi` instances behind Traefik, SignalR requires sticky sessions (session affinity). Without affinity, negotiate and WebSocket upgrade can hit different instances, which may cause errors like:

`The connection could not be found on the server ... If you have multiple servers check that sticky sessions are enabled.`

This project already enables sticky sessions in traefik.dynamic.yml for the `app-service` load balancer.

Configuration

Environment Variables

This project uses:

- `.env` for production Docker Compose (loaded automatically by Docker Compose)

Local development with `docker-compose.yml` currently does not require an `.env` file.

`.env` (production)

```
# Database
DATABASE_NAME=fedlech
DATABASE_USER=postgres
DATABASE_PASSWORD=change-me

# DB capacity tuning
POSTGRES_MAX_CONNECTIONS=500
NPGSQL_MAX_POOL_SIZE=80
NPGSQL_MIN_POOL_SIZE=5

# JWT
JWT_SECRET=change-me-min-32-chars
JWT_ISSUER=http://localhost
JWT_AUDIENCE=FedlechApiClient
JWT_EXPIRATION_MINUTES=60

REFRESH_TOKEN_EXPIRATION_DAYS=7
```

Database

Migrations

```
# Create a new migration
dotnet ef migrations add MigrationName --project src/Infrastructure --sta

# Apply migrations
dotnet ef database update --project src/Infrastructure --startup-project

# Remove last migration (unapplied only)
dotnet ef migrations remove --project src/Infrastructure --startup-projec
```

Testing

```
dotnet test                                          # all tests
dotnet test tests/Domain.UnitTests
dotnet test tests/Application.UnitTests
dotnet test tests/Infrastructure.IntegrationTests
dotnet test tests/Application.FunctionalTests
dotnet test tests/Web.AcceptanceTests
```

Load tests: see tests/load-tests/README.md

Security Considerations

- JWT secrets are stored in environment variables, never in source code
- Password hashing uses BCrypt with salt
- Database passwords use strong randomly generated values
- Refresh tokens stored securely with expiration
- API endpoints protected with authorization attributes