Project information

Created on
February 16, 2026

Name	Last update
.vscode	2 months ago
src	1 day ago
.gitignore	1 month ago
.gitlab-ci.yml	2 months ago
README.md	just now
docker-compose.yml	just now
nginx.conf	1 month ago
traefik-dynamic.yml	1 month ago
watchlist-jwt-load-test.lua	1 day ago

README.md

DSy-Watchlist

Getting Started

Prerequisites

- Docker and Docker Compose installed
- .NET SDK (for local development)
- Node.js and npm (for frontend development)
- wrk (for load testing) - optional

Architecture

Overview

DSy-Watchlist is a modern, full-stack web application featuring a containerized microservices-inspired architecture with load balancing, JWT-based authentication, and a clean separation between frontend and backend.

Technology Stack

Frontend:

- **Framework**: React 19 with TypeScript
- **Build Tool**: Vite
- **State Management**: Zustand
- **Data Fetching**: TanStack React Query
- **UI Library**: Material-UI (MUI)
- **API Client Generation**: Orval (generates type-safe API clients from OpenAPI specs)
- **Styling**: Emotion
- **Routing**: React Router DOM

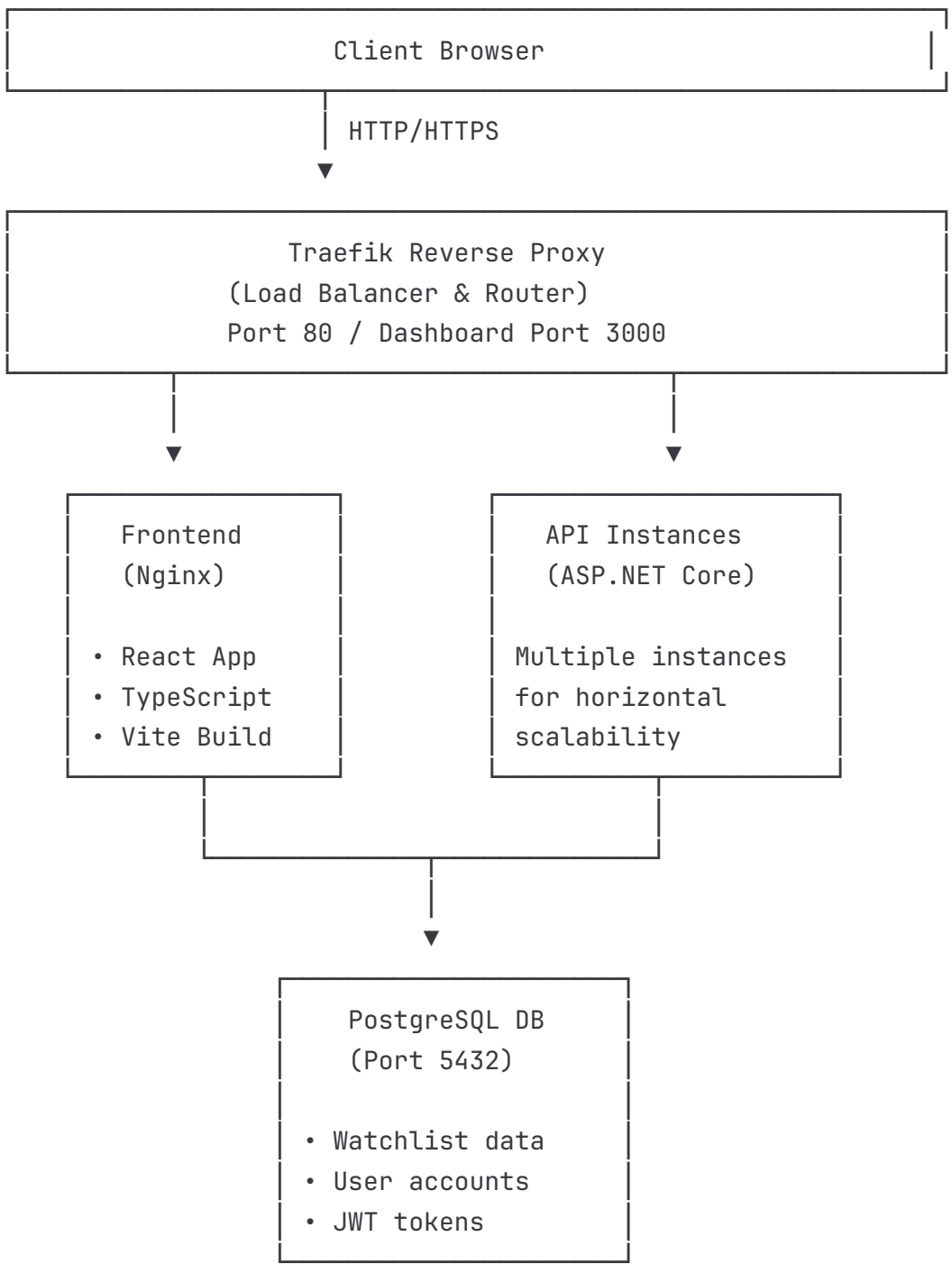
Backend:

- **Runtime**: ASP.NET Core (.NET)
- **Database**: PostgreSQL
- **Architecture Pattern**: Layered architecture (API, Core, Persistence layers)
- **Authentication**: JWT (JSON Web Tokens)
- **API Documentation**: OpenAPI/Swagger

Infrastructure:

- **Reverse Proxy & Load Balancer**: Traefik v3.0
- **Containerization**: Docker & Docker Compose
- **Web Server**: Nginx (frontend)
- **Networking**: Docker bridge network

Application Architecture



Backend Layered Architecture

The backend is organized into multiple projects following a layered architecture pattern:

1. **Watchlist.WebApi** - API Layer
    - HTTP endpoints and controllers
    - Request/response handling
    - Exception handling and middleware
    - OpenAPI/Swagger integration
2. **Watchlist.Core** - Business Logic & Domain Layer
    - Domain models and entities
    - Services and business rules
    - Authentication and authorization (JWT)
    - Configuration management
    - OMDb API integration

3. **Watchlist.Persistence.PostgreSql** - Data Access Layer
    - Entity Framework Core integration
    - Database migrations
    - Repository pattern implementation
    - PostgreSQL-specific implementations
4. **Watchlist.Tests** - Testing
    - Unit tests
    - Integration tests
    - Testing infrastructure

Key Architectural Features

- **Load Balancing**: Multiple API instances (watchlist.webapi-1, watchlist.webapi-2) load-balanced by Traefik
- **Horizontal Scalability**: Easy to add more API instances by extending docker-compose.yml
- **API-First Design**: OpenAPI-compliant REST API with auto-generated client types
- **Type Safety**: Full type coverage from backend to frontend via Orval code generation
- **JWT Authentication**: Stateless authentication mechanism for scalability
- **Database Seeding**: Automatic database initialization and seeding on first run
- **Exception Handling**: Centralized exception handling with problem details response format (RFC 7807)
- **Configuration Management**: Environment-based configuration with validation

Deployment

All services run in Docker containers orchestrated with Docker Compose:

- **Production Environment**: Uses multiple API instances for load distribution
- **Development Environment**: Can run locally with .NET SDK and Node.js
- **Database**: PostgreSQL with automatic migrations and seeding

Starting Docker Compose

To start the entire application stack using Docker Compose:

```
docker-compose up -d
```

This will start:

- **Traefik**: Reverse proxy and load balancer (accessible at http://localhost:3000)
- **PostgreSQL**: Database service
- **Watchlist API instances**: Multiple API containers for load distribution

To view logs:

```
docker-compose logs -f
```

To stop the services:

```
docker-compose down
```

To rebuild images and start fresh:

```
docker-compose up -d --build
```

Running the Load Test

Before running the load test, ensure the services are running. The project includes a load test script for testing the API with JWT authentication.

Prerequisites for load testing:

- Install `wrk` (HTTP benchmarking tool)
    - On Windows: Use WSL or install from https://github.com/wg/wrk
    - On macOS: `brew install wrk`
    - On Linux: `apt-get install wrk` or equivalent

To run the load test:

Start docker compose:

```
docker-compose up -d
```

Login as user and extract jwt-token

Start load test:

```
wrk -t4 -c100 -d30s -s watchlist-jwt-load-test.lua http://localhost:80
```

Parameters:

- `-t4`: Number of threads (4)
- `-c100`: Number of concurrent connections (100)
- `-d30s`: Duration of test (30 seconds)
- `-s watchlist-jwt-load-test.lua`: Load the Lua script for custom request setup

Example Results:

```
Running 30s test @ http://localhost/api/watchlist
  4 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    23.00ms   16.06ms 350.98ms   84.54%
    Req/Sec     1.16k   302.73     2.18k    72.82%
  138393 requests in 29.31s, 121.56MB read
  Socket errors: connect 0, read 0, write 0, timeout 184
Requests/sec:   4721.56
Transfer/sec:      4.15MB
```

Note: Update the `watchlist-jwt-load-test.lua` script with a valid JWT token in the `token` variable before running the test.