Niels Roefs

Abstract

The technical design document outlines the architecture and implementation strategy for MyWatchList, a comprehensive platform designed to facilitate seamless tracking, discovery, and social sharing of movies and shows.

Technical Design Document

Individual Project

Contents

[Project Overview 2](#_Toc160698172)

[Introduction 2](#_Toc160698173)

[Scope 2](#_Toc160698174)

[System Overview 2](#_Toc160698175)

[Key Components 2](#_Toc160698176)

[Frontend Components 2](#_Toc160698177)

[Backend Services and APIs 2](#_Toc160698178)

[Architecture 2](#_Toc160698179)

[C4 Model 2](#_Toc160698180)

[Deployment 3](#_Toc160698181)

[Design Decisions 3](#_Toc160698182)

[Microservices Architecture 3](#_Toc160698183)

[Frontend Framework Selection (React.js) 3](#_Toc160698184)

[Backend Framework Selection (NestJS) 3](#_Toc160698185)

[Database Selection (PostgreSQL and MongoDB) 3](#_Toc160698186)

[Authentication and Authorization Mechanism (JWT and RBAC) 3](#_Toc160698187)

[Deployment Strategy (CI/CD with Docker and Kubernetes) 4](#_Toc160698188)

[Components 4](#_Toc160698189)

[Frontend 4](#_Toc160698190)

[Backend Services and APIs 4](#_Toc160698191)

[Database Schema 4](#_Toc160698192)

[Data Model 5](#_Toc160698193)

[Entity-Relationship Diagram 5](#_Toc160698194)

[Database Schema 5](#_Toc160698195)

[Technologies Used 7](#_Toc160698196)

[Deployment 8](#_Toc160698197)

# Project Overview

## Introduction

MyWatchList is a platform designed to redefine how users track, discover, and share their favorite movies and shows. With a focus on user engagement and seamless functionality, MyWatchList offers a centralized hub for managing watched content, connecting with friends, and writing reviews.

## Scope

The scope of the MyWatchList project includes:

* Development of a web-based platform accessible via browsers on desktop and mobile devices.
* Implementation of core features such as user registration, watchlist management, friend connections, and review sharing.
* Integration with external APIs for accessing movie and show metadata, including titles, genres, cast, and ratings.
* Implementation of security measures such as authentication, and authorization.
* Deployment of the platform to a scalable and reliable infrastructure to support future growth and demand.

# System Overview

## Key Components

### Frontend Components

MyWatchList's frontend is developed using React.js, a modern JavaScript library for building user interfaces. The frontend components include interactive elements such as watchlist management, browsing features, user profiles, and review sharing functionalities. The user interface is designed to be intuitive, engaging, and responsive across various devices and screen sizes.

### Backend Services and APIs

Selected for the backend services, .NET Core offers a high-performance, cross-platform framework for building modern, cloud-based, Internet-connected applications. It provides a robust set of APIs for building scalable web services.

## Architecture

MyWatchList adopts a microservices architecture to ensure scalability, modularity, and flexibility. The platform is decomposed into smaller, independently deployable services, each responsible for specific functionalities. These services communicate with each other via APIs or messaging protocols, enabling distributed development, deployment, and scaling of the platform. Containerization technologies such as Docker are utilized to package and deploy services as lightweight, portable containers, while orchestration tools like Kubernetes manage containerized applications, ensuring scalability, availability, and resilience.

## C4 Model

The C4 model provides a structured approach to visualize and document the system architecture at different levels of abstraction.

A diagram of a user

Description automatically generated

A diagram of a company

Description automatically generated

## Deployment

MyWatchList is deployed to cloud infrastructure providers such as AWS, Azure, or Google Cloud Platform. Continuous integration and deployment (CI/CD) pipelines automate the build, test, and deployment processes, enabling rapid and reliable delivery of updates and enhancements to the platform. Deployment scripts and configuration files ensure consistency and reliability in the deployment process, while monitoring and logging mechanisms provide visibility into system performance and health.

# Design Decisions

## Microservices Architecture

We chose to adopt a microservices architecture for MyWatchList to ensure scalability, modularity, and flexibility in the development and deployment of the platform. By decomposing the system into smaller, independently deployable services, we can facilitate distributed development, deployment, and scaling of the platform. This architecture also enables easier maintenance and updates to individual components, contributing to the long-term scalability and sustainability of MyWatchList.

## Frontend Framework Selection (React.js)

In selecting the frontend framework for MyWatchList, we opted for React.js due to its component-based architecture and efficient rendering capabilities. By using React.js, we can build a responsive and interactive user interface that is consistent across various devices and screen sizes. The modular nature of React.js allows for easier development and reusability of UI components, promoting maintainability and extensibility of the frontend codebase.

## Backend Framework Selection (.NET Core)

For the backend development of MyWatchList, we selected .NET Core as the server-side framework. .NET Core provides a powerful and scalable foundation for building efficient, maintainable backend services. With its performance, cross-platform support, and extensive library ecosystem, .NET Core enables us to develop robust backend services that are easy to maintain and test. Additionally, .NET Core’s integration with Entity Framework for ORM, its support for microservices architecture, and compatibility with various cloud platforms offer the flexibility and extensibility needed for complex backend systems.

## Database Selection (PostgreSQL and MongoDB)

In considering the database architecture for MyWatchList, we are evaluating the use of PostgreSQL for structured data storage and MongoDB for unstructured data storage. PostgreSQL offers ACID compliance, data integrity, and support for complex queries, making it suitable for managing user data, authentication, and relational data. On the other hand, MongoDB provides flexibility and scalability for storing metadata, reviews, and user-generated content in a document-based format, enabling fast and efficient data retrieval and storage. The combination of PostgreSQL and MongoDB allows us to leverage the strengths of each database solution based on the specific requirements of MyWatchList.

## Authentication and Authorization Mechanism (JWT and RBAC)

For user authentication and authorization in MyWatchList, we decided to implement JSON Web Token (JWT) authentication along with Role-Based Access Control (RBAC). JWT tokens are issued upon successful login and used to authenticate subsequent API requests, providing a secure and efficient means of user authentication. RBAC ensures that users have appropriate access to resources and functionalities based on their roles and privileges, enhancing data security and integrity within the platform.

## RabbitMQ for Asynchronous Messaging

To enhance the scalability and reliability of MyWatchList, we decided to adopt RabbitMQ as our messaging system. This decision was driven by the need for a robust, scalable, and flexible message broker to facilitate asynchronous communication between microservices. RabbitMQ's support for multiple messaging protocols, its high throughput, and its ability to handle heavy loads make it an ideal choice for enabling decoupled, asynchronous processing across the system.

## Integration of an API Gateway

Incorporating an API Gateway as the single entry point for all incoming requests was a strategic decision aimed at simplifying client interactions with our microservices architecture. The API Gateway will handle request routing, authentication, and rate limiting, among other cross-cutting concerns. This not only enhances security by centralizing authentication and authorization mechanisms but also improves maintainability and scalability by abstracting the complexity of microservices from clients.

## Deployment Strategy (CI/CD with Docker and Kubernetes)

To streamline the deployment process and ensure reliability and scalability, we opted for a continuous integration and deployment (CI/CD) strategy using Docker containers and Kubernetes orchestration. CI/CD pipelines automate the build, test, and deployment processes, enabling rapid and reliable delivery of updates and enhancements to the MyWatchList platform. Docker containers provide portability and consistency in deployment, while Kubernetes manages containerized applications, ensuring scalability, availability, and resilience in production environments.

# Components

## Frontend

The frontend of the MyWatchList project is a single-page application (SPA) built using React. It provides an interactive user interface where users can manage their watchlist, search for movies and TV shows, and view details about them. The frontend communicates with the backend through RESTful APIs, utilizing Redux for state management to ensure a seamless user experience.

## Backend Services and APIs

The backend is developed using .NET Core APIs, offering a robust and scalable solution for handling the application logic, user authentication, and database interactions. It is structured into several key components:

* **API Layer:** This layer serves as the entry point for the frontend to interact with the backend. It defines various endpoints for user authentication, managing watchlists, and fetching movies and TV show details. The API documentation is provided through Swagger, enabling easy exploration and testing of the API endpoints.
* **Business Logic Layer:** Contains the core functionality and rules of the application. It processes requests from the API layer, interacts with the database to persist or retrieve data, and handles business logic such as user authentication, watchlist management, and integration with external movie databases.
* **Data Access Layer:** Utilizing Entity Framework Core, this layer abstracts the interactions with the PostgreSQL database. It includes models, repositories, and services for accessing and manipulating database entities such as users, watchlists, and movie/show details.
* **Authentication and Authorization:** Built on ASP.NET Identity for managing users, authentication, and authorization. It ensures that only authenticated users can access their watchlists and perform operations like adding or removing items.
* **Logging and Error Handling:** Implements Serilog for structured logging, facilitating easier monitoring and debugging. The backend is designed to gracefully handle and log errors, providing feedback for troubleshooting and ensuring the application's reliability.
* **API Gateway:** The API Gateway serves as the front-facing component of MyWatchList, directing incoming API requests to the appropriate microservices. It aggregates responses and provides essential functionalities such as SSL termination, authentication, rate limiting, and logging.

## Database Schema

The database schema for MyWatchList includes tables and collections to store user data, watchlists, reviews, friendships, and other relevant information. It supports both relational and non-relational data storage requirements.

## RabbitMQ

RabbitMQ acts as the messaging backbone, enabling microservices to communicate asynchronously through message queuing. This component decouples service dependencies, allowing for independent scaling and enhancing overall system resilience.

# Data Model

## Entity-Relationship Diagram

The Entity-Relationship Diagram (ERD) illustrates the data model for MyWatchList, including the entities, attributes, and relationships.

A diagram of a watchlist

Description automatically generated

## Database Schema

The database schema outlines the tables or collections and their respective fields, along with data types and constraints.

Tables/Collections:

1. **users:**

* id (Primary Key)
* username
* email
* password
* role
* created\_at
* updated\_at

1. **watchlists:**

* id (Primary Key)
* user\_id (Foreign Key)
* name
* created\_at
* updated\_at

1. **watchlist\_items:**

* id (Primary Key)
* watchlist\_id (Foreign Key)
* movie\_id (Foreign Key, Nullable)
* show\_id (Foreign Key, Nullable)
* created\_at
* updated\_at

1. **movies:**

* id (Primary Key)
* title
* genre
* release\_date
* synopsis
* average\_rating
* created\_at
* updated\_at

1. **shows:**

* id (Primary Key)
* title
* genre
* release\_date
* synopsis
* average\_rating
* created\_at
* updated\_at

1. **reviews:**

* id (Primary Key)
* user\_id (Foreign Key)
* movie\_id/show\_id (Foreign Key)
* rating
* comment
* created\_at
* updated\_at

1. **friend:**

* id (Primary Key)
* user1\_id (Foreign Key)
* user2\_id (Foreign Key)
* status
* created\_at
* updated\_at

# Technologies Used

**Frontend Development**

* **React.js:** JavaScript library for building user interfaces.
* **HTML5:** Markup language for structuring web content.
* **CSS3:** Style sheet language for styling web content.
* **JavaScript (ES6+):** Programming language for adding interactivity to web pages.
* **Redux:** State management library for React applications.
* **Axios:** Promise-based HTTP client for making AJAX requests.

**Backend Development**

* **.NET Core APIs:** The backend is powered by .NET Core APIs, a high-performance, open-source framework. .NET Core is specifically chosen for its robustness, scalability, and support for building modern, cloud-enabled, Internet-connected apps. The backend architecture is designed to be modular, with a focus on clean, testable, and maintainable code.
* **Entity Framework Core:** Used for data access, EF Core allows for a code-first approach, reducing the need for boilerplate data access code and enabling a more agile development workflow.
* **ASP.NET Identity:** For secure user authentication and authorization. It provides a comprehensive system for managing users, passwords, profile data, roles, claims, tokens, email confirmation, and more.
* **Automapper:** To simplify object-to-object mappings, Automapper reduces the amount of manual coding required to assign property values between objects, making the code cleaner and maintenance easier.
* **Ocelot:** Ocelot is chosen as the API Gateway solution for MyWatchList. It is an open-source .NET library that provides a simple and easy-to-use API gateway for routing and managing HTTP requests within a microservices architecture. Ocelot offers features such as routing, load balancing, authentication, and request aggregation, making it suitable for handling the complex routing requirements of MyWatchList's microservices. Its compatibility with the .NET ecosystem and ease of configuration make it a suitable choice for integrating with our existing technology stack.
* **RabbitMQ:** RabbitMQ is selected for its proven reliability, high availability, and support for multiple messaging protocols, making it the preferred message broker for implementing asynchronous communication and event-driven architectures.

**Database**

* **PostgreSQL:** Open-source relational database management system.
* **MongoDB:** NoSQL document database for storing unstructured data.

**Authentication and Authorization**

* **Auth0:** For secure, scalable, and easy-to-implement user authentication and authorization services.

**Deployment and Infrastructure**

* **Docker:** Platform for developing, shipping, and running applications in containers.
* **Kubernetes:** Open-source container orchestration platform for automating deployment, scaling, and management of containerized applications.
* **GitLab:** Version control system and DevOps platform for managing Git repositories, CI/CD pipelines, and collaboration.
* **Amazon Web Services (AWS):** Cloud computing platform providing a wide range of services, including compute, storage, and databases.
* **Google Cloud Platform (GCP):** Suite of cloud computing services offered by Google.
* **Microsoft Azure:** Cloud computing service for building, testing, deploying, and managing applications and services.

**Development Tools**

* **Visual Code:** Source code editor developed by Microsoft.
* **GitLab:** Hosting service for Git repositories.

**Testing and Quality Assurance**

* **SonarQube:** Open-source platform for continuous inspection of code quality.

# Deployment

**Containerization**

* **Docker:** Utilized for containerizing the MyWatchList application components, including frontend, backend, and any associated services.
* **Docker Compose:** Used for defining and running multi-container Docker applications, simplifying the management of containerized deployments.

**Orchestration**

* **Kubernetes:** Employed for orchestrating and automating the deployment, scaling, and management of containerized applications.
* **Helm:** Package manager for Kubernetes used for simplifying the deployment and management of Kubernetes applications through charts.

**Continuous Integration and Continuous Deployment (CI/CD)**

* **GitLab CI/CD:** Integrated into the GitLab platform for automating the build, test, and deployment processes.
  + **Build Stage:** Builds Docker images for frontend and backend components.
  + **Test Stage:** Executes unit tests, integration tests, and other quality checks.
  + **Deploy Stage:** Deploys the application to staging or production environments based on predefined conditions.
* **Pipeline Configuration:** Defined using YAML-based configuration files within the GitLab repository, specifying the stages, jobs, and deployment targets.