Niels Roefs

Abstract

A comprehensive project plan outlining the development of MyWatchList, an innovative platform for tracking, discovering, and sharing movies and shows.

Project plan

Individual Project

Contents

[1. Project Overview 3](#_Toc160745184)

[1.1. Introduction 3](#_Toc160745185)

[1.2. Objectives 3](#_Toc160745186)

[1.3. Target Audience 3](#_Toc160745187)

[2. Project Scope & Deliverables 3](#_Toc160745188)

[2.1. Scope 3](#_Toc160745189)

[2.2. Key Features 3](#_Toc160745190)

[2.2.1. User Authentication and Authorization 3](#_Toc160745191)

[2.2.2. Watchlist Management 3](#_Toc160745192)

[2.2.3. Movie and Show Discovery 4](#_Toc160745193)

[2.2.4. Interactive User Interface 4](#_Toc160745194)

[2.2.5. Review Sharing 4](#_Toc160745195)

[2.2.6. Friend Connections 4](#_Toc160745196)

[2.3. Deliverables 4](#_Toc160745197)

[3. Requirements 4](#_Toc160745198)

[3.1. Functional Requirements 4](#_Toc160745199)

[3.2. Non Functional Requirements 5](#_Toc160745200)

[3.2.1. Scalability Requirement 5](#_Toc160745201)

[3.2.2. Performance 5](#_Toc160745202)

[3.2.3. Compliance 5](#_Toc160745203)

[3.2.4. Continuous Integration and Deployment (CI/CD) 5](#_Toc160745204)

[3.2.5. Cloud Platform Integration 5](#_Toc160745205)

[3.2.6. Security Requirements 6](#_Toc160745206)

[3.2.7. Data Storage and Handling 6](#_Toc160745207)

[4. Research 6](#_Toc160745208)

[4.1. Research Questions 6](#_Toc160745209)

[4.1.1. Main Research Question 6](#_Toc160745210)

[4.1.2. Sub Research Questions 6](#_Toc160745211)

[4.2. Research Methodology 7](#_Toc160745212)

[5. Project Approach 7](#_Toc160745213)

[5.1. Approach 7](#_Toc160745214)

[5.2. Timeline 7](#_Toc160745215)

[6. Technology Stack and Architecture 7](#_Toc160745216)

[6.1. Frontend Development (React.js) 7](#_Toc160745217)

[6.2. Backend Development (.NET Core) 7](#_Toc160745218)

[6.3. Database 8](#_Toc160745219)

[6.4. Authentication and Authorization (JWT and RBAC) 8](#_Toc160745220)

[6.5. Microservices Architecture 8](#_Toc160745221)

[6.6. Deployment and Infrastructure 8](#_Toc160745222)

[7. Testing Strategy and Configuration Management 9](#_Toc160745223)

[7.1. Testing Strategy 9](#_Toc160745224)

[7.2. Test Environment and Required Resources 9](#_Toc160745225)

[7.3. Configuration Management 9](#_Toc160745226)

# 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.

## Objectives

The primary objectives of MyWatchList are to:

* Provide users with a centralized platform to manage and organize their watched movies and shows.
* Facilitate social interaction and discovery by enabling users to connect with friends, share recommendations, and discover new content.
* Enhance user engagement through features such as reviews and ratings.
* Deliver a seamless and intuitive user experience across desktop and mobile devices.

## Target Audience

MyWatchList caters to a diverse audience of movie and show enthusiasts, including:

* Casual viewers looking for an easy way to track their watched content and discover new recommendations.
* Social users interested in connecting with friends, sharing reviews, and engaging in discussions about movies and shows.

# Project Scope & Deliverables

## 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.

## Key Features

### User Authentication and Authorization

* Users can securely sign up, log in, and log out of the platform.
* Authentication is managed using JSON Web Tokens (JWT) for enhanced security.
* Access to different features and data is controlled based on user roles and permissions.

### Watchlist Management

* Users have the ability to create, modify, and organize their own watchlists.
* They can mark movies and shows as watched or to-watch, allowing for easy tracking of their entertainment preferences.
* The system ensures smooth navigation and efficient management of watchlists.

### Movie and Show Discovery

* The platform offers users a variety of tools and features to discover new movies and shows.
* This includes browsing by genre, popularity, release date, and personalized recommendations.
* Users can explore trending content and access detailed information about each movie or show.

### Interactive User Interface

* The user interface is designed to be engaging, intuitive, and user-friendly.
* Interactive elements such as search filters, sorting options, and dynamic updates enhance the user experience.
* Seamless navigation and visually appealing design elements contribute to user satisfaction.

### Review Sharing

* Users can write, share, and read reviews for movies and shows.
* They can rate content and engage in discussions with other users through comments.
* Reviews contribute to a vibrant community where users can exchange opinions and recommendations.

### Friend Connections

* Users have the ability to connect with friends and build social networks within the platform.
* They can view their friends' watchlists, see what they're watching, and share reviews and recommendations.
* Friend connections foster a sense of community and enable users to discover content through social interactions.

## Deliverables

* Project Documentation, such as SRS, technical design document, testing strategy document, DevOps documentation, and research document.
* Developed frontend components and user interfaces using React.js, including HTML, CSS, and JavaScript files.
* Developed backend services and APIs using .NET Core, including controllers, services, and data models.
* Deployment scripts, configuration files, and documentation for deploying the MyWatchList platform to production environments.

# Requirements

## Functional Requirements

Must Have:

* Users must be able to securely sign up, log in, and log out.
* Users must be able to create, edit, and organize their personal watchlist.
* Users must be able to mark movies/shows as watched or plan to watch
* Users must be able to write and share reviews for movies/shows.

Should Have:

* Users should be able to connect with friends and view their watchlists.
* The platform should have advanced browsing for discovering new movies and shows.
* Users should be able to rate content.

Could Have:

* Personalized recommendations.
* Additional features such as trending content.
* Search filters, sorting options.

Won’t Have:

* Real-time Chat

## Non Functional Requirements

### Scalability Requirement

The system must be designed to handle increased user loads (e.g. a million concurrent users) and data volume as the platform grows over time. It should support horizontal scaling to accommodate additional users and data without significant performance degradation.

### Performance

The system shall support a minimum throughput of 1000 requests per second during peak usage hours. This ensures that the platform can handle concurrent user interactions, such as searching, browsing, and adding content to watchlists, without experiencing degradation in response times. Load balancing mechanisms and efficient resource allocation shall be implemented to distribute incoming requests evenly across backend services and maintain consistent throughput levels.

The average latency for user interactions on the MyWatchList platform shall not exceed 500 milliseconds. Latency, defined as the time elapsed between sending a request from the client and receiving a response from the server, directly impacts user perception of system responsiveness. To minimize latency, optimizations such as caching, asynchronous processing, and optimized database queries shall be employed. Additionally, content delivery networks (CDNs) may be utilized to reduce latency for static assets and media content.

### Compliance

Security measures, including data encryption, access control, and compliance with privacy regulations (e.g., GDPR), must be integrated into the architecture to ensure the protection of user data and compliance with legal requirements.

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

The development process should include automated CI/CD pipelines to enable frequent and reliable deployments of new features and updates.

### Cloud Platform Integration

The application should leverage cloud-native services and architectures, such as serverless computing, cloud storage, and container orchestration, to optimize scalability, reliability, and cost-effectiveness. Integration with cloud platforms like AWS, Azure, or Google Cloud should be seamless and well-documented.

### Security Requirements

The development process should include a thorough analysis of security risks and the implementation of best practices to mitigate potential vulnerabilities. Measures such as encryption, secure authentication, and authorization mechanisms should be integrated into the architecture from the outset. Todo: observe in pipeline

### Data Storage and Handling

The system should employ best practices for handling and storing large volumes of various data types. Data storage solutions should be selected based on scalability, access patterns, and data variety to ensure optimal performance and reliability.

# Research

## Research Questions

### Main Research Question

How can MyWatchList be effectively prepared for deployment while considering diverse factors such as scalability, performance optimization, and GDPR compliance?

### Sub Research Questions

Scalability:

* What are the key principles and strategies for designing a scalable architecture?
* What are the best practices for load balancing and auto-scaling configurations?

Performance:

* What are the common performance bottlenecks in web applications and how can they be mitigated?
* How can caching mechanisms and CDNs enhance application performance?

GDPR Compliance:

* What are the GDPR requirements for data handling and user privacy?
* What measures need to be taken to ensure GDPR compliance in storage systems?

# Project Approach

## Approach

The MyWatchList project will adopt an Agile methodology for software development, emphasizing iterative and incremental delivery, collaboration, and flexibility in responding to changing requirements.

## Timeline

|  |  |  |
| --- | --- | --- |
| Sprint | Deadline | Tasks |
| 0 | March 3rd | Project Pitch  Project Plan  Start architecture document |
| 1 | March 24th | Design backend architecture (and start with developing backend services if there is time)  Begin frontend design  Setup DevOps tools  Application Skeleton |
| 2 | April 14th | Implement frontend functionalities  Develop backend services |
| 3 | May 12th | Continue implementing frontend functionalities  Continue developing backend services |
| 4 | June 2nd | Perform end-to-end testing  Conduct load testing to evaluate scalability |
| 5 | June 23rd | Prepare for cloud deployment of the application  Set up automated deployment  Final performance and scalability testing  Ensure GDPR compliance |

# Technology Stack and Architecture

## Frontend Development (React.js)

MyWatchList's frontend will be developed using React.js, a popular JavaScript library for building user interfaces. React.js offers a component-based architecture, allowing for modular development and reusability of UI elements. With its virtual DOM and efficient rendering capabilities, React.js ensures a responsive and interactive user experience.

## Backend Development (.NET Core)

For the backend, we have chosen .NET Core due to its high performance, scalability, and support for cross-platform development. .NET Core's comprehensive ecosystem and robust framework allow for the development of efficient, secure, and scalable web APIs. This change will enable us to leverage .NET Core's advanced features, including its support for asynchronous programming and its powerful Entity Framework for ORM.

## Database

The choice of database for MyWatchList is yet to be decided. Options under consideration include 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. 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.

## Authentication and Authorization (JWT and RBAC)

MyWatchList will implement JSON Web Token (JWT) authentication for secure user authentication and authorization. JWT tokens will be issued upon successful login and used to authenticate subsequent API requests. Role-based access control (RBAC) will be employed to manage user permissions, ensuring that users have appropriate access to resources and functionalities based on their roles and privileges.

## Microservices Architecture

MyWatchList will adopt a microservices architecture to ensure scalability, modularity, and flexibility. The platform will be decomposed into smaller, independently deployable services, each responsible for specific functionalities such as user management, content discovery, and social features. Microservices will communicate with each other via APIs or messaging protocols, enabling distributed development, deployment, and scaling of the platform.

## Deployment and Infrastructure

MyWatchList will be deployed to cloud infrastructure providers such as AWS, Azure, or Google Cloud Platform. Containerization technologies like Docker will be used to package and deploy services as lightweight, portable containers. Orchestration tools like Kubernetes will manage containerized applications, ensuring scalability, availability, and resilience. Continuous integration and deployment (CI/CD) pipelines will automate the build, test, and deployment processes, enabling rapid and reliable delivery of updates and enhancements to the platform.

# Testing Strategy and Configuration Management

## Testing Strategy

In our project plan, testing plays a pivotal role in ensuring the quality and reliability of our software solution. Our strategy encompasses various testing levels:

Unit Testing: We aim for 80% code coverage through automated tests, ensuring early detection and resolution of bugs.

End-to-End Testing: Comprehensive tests validate the system's functionality across all components and interfaces, both automated and manual, to ensure seamless operation.

Load Testing: Simulated scenarios stress-test our system's performance under various load conditions, ensuring scalability and stability.

Additionally, we integrate SonarQube into our CI/CD pipeline for continuous code quality monitoring, aiming to maintain a technical debt ratio below 5% and uphold high standards of maintainability and security.

By adopting this multifaceted approach to testing, we ensure that our software meets both our standards and the expectations of our users, delivering a high-quality, reliable product.

## Test Environment and Required Resources

We will utilize a CI/CD (Continuous Integration/Continuous Deployment) environment to automate the build, test, and deployment processes. This environment will streamline development workflows, ensure code consistency, and enable faster delivery of features and updates

## Configuration Management

We have chosen GitLab as our version control system (VCS) due to its comprehensive features, integrated CI/CD pipelines, and robust collaboration tools.

Our branching strategy aligns with the GitFlow model, leveraging GitLab's features such as protected branches and merge request workflows:

* Master Branch: Represents the stable, production-ready code.
* Develop Branch: Serves as the main integration branch for ongoing development efforts.
* Feature Branches: Created for implementing new features or enhancements, branched off from the develop branch.
* Release Branches: Used for preparing releases, enabling coordination between development and release activities.
* Hotfix Branches: Created to address critical issues in the production environment, branched off from the master branch.