Software Requirements Specification (SRS)

Project Title: Nalandaa – Online eBook Platform

Team: TheKade

IT23162600 – Nayantha Nethsara

IT23226746 – Kavindu Karunarathna

IT23289598 – Ushari Yatawara

IT23279070 – Mehan Samarajeewa

1. Introduction

1.1 Purpose

The purpose of this project is to develop Nalandaa, a cloud-based eBook platform that enables authors to publish books and readers to discover and read them online. The system provides a modern, subscription-style model where free and premium readers can access digital content. Admins regulate uploaded books, users, and system activity.

The system will help:
- Authors reach a wide audience and track book performance.
- Readers find and read books online conveniently.
- Admins monitor content and user behavior efficiently.

1.2 Scope

What the system will do:
- Readers: Register/login, browse/search eBooks, read online, bookmark favorites, manage subscriptions.
- Authors: Upload/manage eBooks (title, genre, file), view usage reports.
- Admins: Approve/reject uploads, manage users, generate reports.
- System: Handle CRUD operations for Books, Users, Reviews, Bookmarks. Generate reports on popular books, author earnings, user activity, and genre trends.
- Support free and premium subscription tiers (with simulated payment).

What the system will NOT do:
- Full payment gateway integration (payments are simulated).
- Offline reading/downloads.
- Physical book delivery.
- AI-based book recommendations (out of scope for this version).

1.3 Overview

Nalandaa will be a microservice-based cloud platform:
- Frontend: Next.js web application for readers, authors, and admins.
- Backend: C# ASP.NET Core microservices handling users, books, subscriptions, and reports.
- Database: MySQL to store users, books, subscriptions, and activity logs.
- Deployment: Docker containers on Azure App Service/AKS.
- Testing: Unit tests, Selenium UI tests, and JMeter load testing.

1.4 Definitions, Acronyms, Abbreviations

CRUD – Create, Read, Update, Delete
CI/CD – Continuous Integration / Continuous Deployment
UI – User Interface
API – Application Programming Interface

2. Overall Description

2.1 Product Perspective

Nalandaa is a distributed microservice system:
- Frontend communicates with backend microservices via REST APIs.
- Microservices are modular: Users, Books, Subscriptions, Reports.
- Data stored in MySQL and accessed by corresponding services.
- Deployment uses Docker containers with potential auto-scaling.

2.2 Product Functions

- User Management: registration, login, profile, role-based access.
- Book Management: upload, edit, delete, browse, search.
- Reading Experience: online reader, bookmarks.
- Reports: popular books, author income, user activity, genre trends.
- Subscriptions: free vs premium tiers.

2.3 User Classes & Characteristics

- Readers → want intuitive UI, fast discovery, and smooth reading.
- Authors → want easy upload, management, and usage reports.
- Admins → want moderation tools, dashboards, and reporting.

2.4 Operating Environment

- Browsers: Chrome, Edge, Firefox
- Backend: ASP.NET Core in Docker containers
- Database: MySQL (Azure or local)

2.5 Design & Implementation Constraints

- Must use Next.js, ASP.NET Core, and MySQL.
- Docker containerization required for microservices.
- System must support at least 100 concurrent readers.
- CI/CD must be implemented via GitHub Actions.

2.6 Assumptions & Dependencies

- Users have stable internet connection.
- Authors upload books in supported formats (PDF/EPUB).
- Payments simulated; real transactions not required.
- Browser-based reading (no native mobile app yet).

2.7 Risk Management

Slow upload for large eBooks: Medium likelihood, Medium impact. Mitigation: Limit file size and optimize upload service

Database failure: Low likelihood, High impact. Mitigation: Regular backups and utilize Azure reliability features.

Unauthorized access: Low likelihood, High impact. Mitigation: Implement HTTPS, JWT authentication, and role-based access control.

Microservice downtime: Medium likelihood, Medium impact. Mitigation: Use container health checks and auto-restart mechanisms.

CI/CD pipeline failure: Low likelihood, Medium impact. Mitigation: Test pipelines in staging before deploying to production

3. System Features

3.1 Reader Features

- Register/login.
- Browse/search books by title, genre, author.
- Read online.
- Bookmark/favorite books.
- Manage subscription tier.

3.2 Author Features

- Register/login.
- Upload new book (metadata + file).
- Edit/delete books.
- View reports (downloads, readers).

3.3 Admin Features

- Approve/reject uploads.
- Manage users (activate/deactivate).
- View reports (system-wide activity, income, trends).

4. Functional Requirements

Functional Requirements

- User authentication & role-based access control.
- CRUD operations for Users, Books, Reviews, Bookmarks.
- Book search/filter functionality.
- Online book reading with bookmarking.
- Report generation (popular books, income, user activity).
- Subscription management (free vs premium).

5. Non-Functional Requirements

Non-Functional Requirements

- Performance: Support 100 concurrent readers.
- Security: HTTPS, JWT authentication, role-based authorization.
- Reliability: Cloud deployment, minimal downtime.
- Usability: Responsive, intuitive interface.
- Scalability: Microservices can scale independently.
- Maintainability: Version control, CI/CD pipelines.
- Portability: Works on modern browsers.

6. External Interface Requirements

6.1 User Interfaces

- Responsive Next.js UI.
- Reader dashboard (library view).
- Author dashboard (upload/manage books).
- Admin dashboard (moderation + reports).

6.2 Software Interfaces

- REST APIs with JSON between frontend & microservices.
- MySQL backend.

6.3 Communication Interfaces

- HTTPS for client-server communication.
- Internal REST/gRPC calls between services.

7. Reports

Reports

- Popular Books → Top 10 most read.
- Author Income → Simulated earnings.
- User Activity → Signups per month.
- Genre Trends → Reading trends by genre.

8. Testing Plan

Testing Plan

- Unit Tests: Backend APIs.
- Integration Tests: Book upload & reading workflow.
- UI Tests: Selenium scripts for login → browse → read.
- Load Tests: JMeter with 100 concurrent readers.

9. Deployment Plan

Deployment Plan

- Version Control: GitHub.
- CI/CD: GitHub Actions.
- Containerization: Docker images per microservice.
- Deployment: Azure App Service / AKS.
- Frontend Hosting: Next.js on Vercel or Azure.