Software Requirements Specification (SRS)

for

Online Quiz Application

Version 1.0

Prepared by: Keerthi R.V

Date Created: 12/03/2025

Table of Contents

1. Introduction

1.1 Purpose

The Online Quiz Application is a web-based platform designed for education and assessment. It allows users to create quizzes, participate in them, and track performance. The system supports multiple-choice questions and includes features like timers and analytics. Administrators can manage quizzes, questions, and users securely. The platform ensures scalability and usability through modern web technologies.

1.2 Document Conventions

- Bold headings for sections and subsections.
- Bullet points for listing key features.

1.3 Intended Audience and Reading Suggestions

- **Developers:** Implementation details.
- **Testers:** Functional validation.
- **Admins & Users:** Understanding system capabilities.

1.4 Product Scope

This application provides a scalable and efficient solution for quiz-based assessments. It supports dynamic question management, authentication, analytics, and real-time interaction.

1.5 References

- React.js: https://react.dev/
- Node.js: https://nodejs.org/
- Express.js: https://expressjs.com/
- MongoDB Atlas: https://www.mongodb.com/atlas
- JWT Authentication: https://jwt.io/
- Chart.js: https://www.chartjs.org/
- Tailwind CSS: https://tailwindcss.com/
- Slido Online Quiz Application: https://www.slido.com/
- Slido What's New: https://whatsnew.slido.com/en

2. Overall Description

2.1 Product Perspective

This application is a standalone system with frontend-backend integration using modern web technologies.

2.2 Product Functions

- Quiz creation and management.
- Secure user authentication.
- Real-time timer-based assessments.
- Performance analytics and leaderboards.

2.3 User Classes and Characteristics

- **Admins:** Manage quizzes, questions, and users.
- **Candidates:** Participate in quizzes and view results.

2.4 Operating Environment

- **Frontend:** React.js, Tailwind CSS
- **Backend:** Node.js, Express.js
- **Database:** MongoDB Atlas

2.5 Design and Implementation Constraints

- Secure authentication with JWT.
- Cloud-based database for scalability.
- API-based question fetching.

2.6 User Documentation

- User guides for quiz creation and participation.

2.7 Assumptions and Dependencies

- Requires stable internet connection.
- Frontend and backend hosted separately.

3. External Interface Requirements

3.1 User Interfaces

- **Admin Panel:** Dashboard for quiz management.
- **Candidate View:** Quiz participation and performance tracking.

3.2 Hardware Interfaces

- Web-based, requires modern browsers.

3.3 Software Interfaces

- **Frontend:** React.js
- **Backend:** Express.js REST API
- **Database:** MongoDB Atlas

3.4 Communications Interfaces

- HTTPS-secured API endpoints.
- WebSocket support for real-time updates (if needed).

4. System Features

4.1 Dynamic Question Bank

- Fetch and store questions dynamically.
- Ensure variety and flexibility in quiz design.

4.2 User Authentication

- Secure login for admins and candidates.
- JWT-based authentication.

4.3 Real-Time Timer

- Implement timers for quizzes.
- Auto-submit on timeout.

4.4 Performance Analytics

- Graphical representation using Chart.js.
- Individual and comparative analysis.

4.5 Leaderboard

- Displays rankings based on performance.

4.6 Scalability

- Supports multiple concurrent quizzes.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- Quick quiz loading and response times.
- Optimized API calls for efficiency.

5.2 Security Requirements

- Encrypted user data.
- Secure API authentication.

5.3 Software Quality Attributes

- High availability and reliability.
- User-friendly UI/UX.

5.4 Business Rules

- Role-based access control (Admin & Candidate).
- One attempt per quiz per candidate.

6. Other Requirements

6.1 Hosting and Deployment

- **Frontend:** Netlify/Vercel
- **Backend:** Heroku/AWS/Render

6.2 Development Tools

- **IDE:** VS Code
- **Version Control:** GitHub

6.3 Security Considerations

- CORS configuration for frontend-backend communication.
- Token-based authentication for API access.