**Mekelle University**



EIT-M

School Of Computing

Software Architecture and design

Project

Name: Id.No

1. Dawit G/egziabiher Ugr/172121/12

**Submitted to: ins. Messele**

**Date: 3/1/2025GC**

**Software Requirements Specification (SRS) for Student Registration System**

**Table of Contents**

1. **Introduction**
   * Purpose
   * Scope
   * Definitions, Acronyms, and Abbreviations
   * References
   * Overview
2. **System Features**
3. **Functional Requirements**
4. **Non-Functional Requirements**
5. **System Models**
6. **Test Requirements**
7. **Appendices**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to outline the Software Requirements Specification (SRS) for the Student Registration System. The system is designed to streamline the management of students, courses, payments, progress tracking, notifications, and reports, ensuring a user-friendly experience for administrators, students, and instructors.

**1.2 Scope**

The Student Registration System is a web-based application with a React frontend and a Node.js backend. It serves as a centralized platform for:

* Managing student registrations.
* Handling course information.
* Tracking payment records.
* Monitoring student progress.
* Sending notifications.
* Generating reports.

Key stakeholders include students, instructors, and administrators.

**1.3 Definitions, Acronyms, and Abbreviations**

* **SRS**: Software Requirements Specification
* **API**: Application Programming Interface
* **CRUD**: Create, Read, Update, Delete
* **UI**: User Interface
* **UX**: User Experience
* Documentation: <https://www.mongodb.com/docs>

**1.4 Overview**

This document describes the functional and non-functional requirements, system features, and architectural components of the Student Registration System.

**2. System Features**

**2.1 User Authentication**

* Login functionality for users (students, instructors, administrators).
* Role-based access control using middleware.

**2.2 Course Management**

* Add, update, view, and delete courses.
* List all available courses.

**2.3 Student Registration**

* Enroll students in courses.
* View and update registration records.

**2.4 Payment Management**

* Handle payments.
* Retrieve and update payment records.

**2.5 Progress Tracking**

* Record and view student progress.
* Update or delete progress entries.

**2.6 Notifications**

* Send notifications to users.
* Manage notification records.

**2.7 Reporting**

* Generate, view, and update system reports.
* Delete old reports.

**2.8 Scheduling**

* Create and manage schedules for courses.
* View schedules for specific courses.

**3. Functional Requirements**

**3.1 User Authentication**

* Users must be able to log in with their credentials.
* Role-based middleware ensures access control for API routes.

**3.2 Course Management**

* Endpoint to add new courses.
* Endpoint to retrieve all courses or a specific course by ID.
* Endpoint to update course details.
* Endpoint to delete a course.

**3.3 Student Registration**

* Endpoint to register students for courses.
* Endpoint to retrieve all registrations or a specific registration by ID.
* Endpoint to update registration details.
* Endpoint to delete a registration.

**3.4 Payment Management**

* Endpoint to create payment records.
* Endpoint to retrieve all payments or a specific payment by ID.
* Endpoint to update payment records.
* Endpoint to delete a payment record.

**3.5 Progress Tracking**

* Endpoint to create progress records.
* Endpoint to retrieve all progress records or a specific record by ID.
* Endpoint to update progress details.
* Endpoint to delete a progress record.

**3.6 Notifications**

* Endpoint to send notifications.
* Endpoint to retrieve all notifications or a specific notification by ID.
* Endpoint to update notification details.
* Endpoint to delete a notification.

**3.7 Reporting**

* Endpoint to create system reports.
* Endpoint to retrieve all reports or a specific report by ID.
* Endpoint to update report details.
* Endpoint to delete a report.

**3.8 Scheduling**

* Endpoint to create course schedules.
* Endpoint to retrieve all schedules or a specific schedule by ID.
* Endpoint to update schedule details.
* Endpoint to delete a schedule.

**4. Non-Functional Requirements**

**4.1 Scalability**

The system must support a growing number of users, courses, and registrations without significant performance degradation.

**4.2 Security**

* Data must be securely stored in MongoDB.
* Passwords must be hashed.
* Sensitive endpoints must be protected with authentication and authorization mechanisms.

**4.3 Usability**

* The React frontend must provide an intuitive and responsive UI.
* Users must be able to easily navigate the system.

**4.4 Availability**

The system must be available 99.9% of the time, with minimal downtime for maintenance.

**4.5 Performance**

* API responses must be delivered within 500ms under normal load.
* The frontend must load within 2 seconds on a typical broadband connection.

**5. System Models**

**5.1 Architecture Diagram**

* **Frontend**: React application for user interaction.
* **Backend**: Node.js with Express for API services.
* **Database**: MongoDB for data storage.

**5.2 Data Models**

* **User**: { id, name, role, email, password }
* **Course**: { id, title, description, scheduleId }
* **Registration**: { id, userId, courseId, date }
* **Payment**: { id, userId, amount, date }
* **Progress**: { id, userId, courseId, progress }
* **Notification**: { id, userId, message, date }
* **Report**: { id, title, content, date }
* **Schedule**: { id, courseId, time }

## 6. Testing Requirements

### 6.1 Testing Framework

* Use Mocha and Chai for backend testing.
* Use Jest for frontend testing.

### 6.2 Testing Types

1. **Unit Testing**:
   * Test individual components and functions.
   * Example: Testing the course creation API endpoint.
2. **Integration Testing**:
   * Ensure that multiple components interact correctly.
   * Example: Verifying that a new course appears in the list after creation.
3. **End-to-End Testing**:
   * Simulate user workflows.
   * Example: A student registers for a course, makes a payment, and receives a confirmation.
4. **Performance Testing**:
   * Test system performance under various loads.
5. **Security Testing**:
   * Verify that sensitive data is encrypted.
   * Test for vulnerabilities like SQL injection or cross-site scripting (XSS).

### 6.3 Testing Tools

* **Frontend**: Jest, React Testing Library
* **Backend**: Mocha, Chai
* **API Testing**: Postman, Newman
* **Load Testing**: Apache JMeter

### 6.4 Reporting

* Generate detailed reports for each testing phase.

**7. Appendices**

**7.1 Assumptions**

* Users will access the system through modern web browsers.
* MongoDB is pre-installed and configured on the deployment server.

**7.2 Future Enhancements**

* Integration with third-party payment gateways.
* Enhanced analytics and reporting.
* Mobile application development.