Software Requirements Specification (SRS)

# iLearnSpace - Attendance Management System using Local Area Network (LAN) using Java GUI

# Table of Contents

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, and Abbreviations

1.4 References

2. Overall Description

2.1 Product Perspective

2.2 Product Functions

2.3 User Classes and Characteristics

2.4 Operating Environment

2.5 Design and Implementation Constraints

2.6 Assumptions and Dependencies

3. Specific Requirements

3.1 Functional Requirements

3.2 Non-Functional Requirements

3.3 Interface Requirements

4. System Features

4.1 Login Page

4.2 Admin Page

4.3 Lecturer Page

4.4 Student Page

5. Database Design

5.1 Database Schema

5.2 Entity-Relationship Diagram

6. Appendices

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to provide a detailed description of the iLearnSpace Attendance Management System. It will explain the system's functionality, constraints, and requirements.

## 1.2 Scope

iLearnSpace is a university attendance management system that operates over a Local Area Network (LAN) to ensure secure and real-time attendance tracking. The system includes three user roles: Admin, Lecturer, and Student, each with specific functionalities.

## 1.3 Definitions, Acronyms, and Abbreviations

- LAN: Local Area Network  
- GUI: Graphical User Interface  
- OTP: One-Time Password

## 1.4 References

- Project Proposal Document  
- Java Swing/AWT Documentation

# 2. Overall Description

## 2.1 Product Perspective

iLearnSpace is a standalone application that will be deployed on a university's local network. It will use Java Swing/AWT for the GUI and a relational database for data storage.

## 2.2 Product Functions

- User authentication and role-based access control  
- Attendance tracking with OTP verification  
- User and course management  
- Attendance statistics visualization  
- Notifications and communication

## 2.3 User Classes and Characteristics

- Admin: Manages users, courses, and oversees the attendance process.  
- Lecturer: Schedules classes, takes attendance, and communicates with students.  
- Student: Views scheduled classes, marks attendance, and checks attendance records.

## 2.4 Operating Environment

- Operating System: Linux  
- Development Environment: Visual Studio Code  
- Database: MySQL or SQLite

## 2.5 Design and Implementation Constraints

- The system must operate within a LAN.  
- Attendance marking is restricted to the LAN.

## 2.6 Assumptions and Dependencies

- Users have access to the LAN.  
- The database server is accessible within the LAN.

# 3. Specific Requirements

## 3.1 Functional Requirements

- Login Page: Authenticate users based on their role.  
- Admin Dashboard: Manage users, courses, and attendance.  
- Lecturer Dashboard: Schedule classes and take attendance.  
- Student Dashboard: View classes and mark attendance.

## 3.2 Non-Functional Requirements

- Performance: The system should handle up to 1000 concurrent users.  
- Security: User data and attendance records must be securely stored.  
- Usability: The GUI should be intuitive and easy to use.

## 3.3 Interface Requirements

- User Interface: Java Swing/AWT-based GUI.  
- Database Interface: JDBC for database connectivity.

# 4. System Features

## 4.1 Login Page

- Admin Login: Requires Username, Password, and User Type (Admin). OTP verification via email.  
- Lecturer & Student Login: Requires Email and Password.

## 4.2 Admin Page

- User & Course Management: Add, modify, and delete users and courses.  
- Attendance System: Oversee attendance process with OTP verification.  
- Official Notices: Send notices to users.

## 4.3 Lecturer Page

- Profile Information: View personal and course information.  
- Class Management: Schedule and manage classes.  
- Attendance Visualization: Display attendance statistics.

## 4.4 Student Page

- Class Information: View scheduled and ongoing classes.  
- Attendance Marking: Mark attendance using OTP.  
- Attendance Records: View past attendance logs and statistics.

# 5. Database Design

## 5.1 Database Schema

```sql  
-- Drop tables if they already exist (to avoid duplication errors during re-runs)  
DROP TABLE IF EXISTS attendance;  
DROP TABLE IF EXISTS notifications;  
DROP TABLE IF EXISTS courses;  
DROP TABLE IF EXISTS student\_course;  
DROP TABLE IF EXISTS lecturer\_course;  
DROP TABLE IF EXISTS students;  
DROP TABLE IF EXISTS lecturers;  
DROP TABLE IF EXISTS admins;  
DROP TABLE IF EXISTS users;  
  
-- Create a common users table to store login credentials for Admins, Lecturers, and Students  
CREATE TABLE users (  
 user\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 email VARCHAR(100) UNIQUE NOT NULL,  
 password\_hash VARCHAR(255) NOT NULL,  
 user\_type ENUM('Admin', 'Lecturer', 'Student') NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Admin Table (Linked to users table)  
CREATE TABLE admins (  
 admin\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT UNIQUE NOT NULL,  
 full\_name VARCHAR(100) NOT NULL,  
 phone VARCHAR(15),  
 FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
  
-- Lecturer Table (Linked to users table)  
CREATE TABLE lecturers (  
 lecturer\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT UNIQUE NOT NULL,  
 full\_name VARCHAR(100) NOT NULL,  
 department VARCHAR(100) NOT NULL,  
 phone VARCHAR(15),  
 FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
  
-- Student Table (Linked to users table)  
CREATE TABLE students (  
 student\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT UNIQUE NOT NULL,  
 full\_name VARCHAR(100) NOT NULL,  
 enrollment\_number VARCHAR(50) UNIQUE NOT NULL,  
 department VARCHAR(100) NOT NULL,  
 phone VARCHAR(15),  
 FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
  
-- Course Table (Stores all university courses)  
CREATE TABLE courses (  
 course\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 course\_name VARCHAR(100) NOT NULL,  
 course\_code VARCHAR(20) UNIQUE NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Mapping Table: Many-to-Many Relationship Between Students & Courses  
CREATE TABLE student\_course (  
 student\_id INT NOT NULL,  
 course\_id INT NOT NULL,  
 PRIMARY KEY (student\_id, course\_id),  
 FOREIGN KEY (student\_id) REFERENCES students(student\_id) ON DELETE CASCADE,  
 FOREIGN KEY (course\_id) REFERENCES courses(course\_id) ON DELETE CASCADE  
);  
  
-- Mapping Table: Many-to-Many Relationship Between Lecturers & Courses  
CREATE TABLE lecturer\_course (  
 lecturer\_id INT NOT NULL,  
 course\_id INT NOT NULL,  
 PRIMARY KEY (lecturer\_id, course\_id),  
 FOREIGN KEY (lecturer\_id) REFERENCES lecturers(lecturer\_id) ON DELETE CASCADE,  
 FOREIGN KEY (course\_id) REFERENCES courses(course\_id) ON DELETE CASCADE  
);  
  
-- Attendance Table (Stores attendance records)  
CREATE TABLE attendance (  
 attendance\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 student\_id INT NOT NULL,  
 lecturer\_id INT NOT NULL,  
 course\_id INT NOT NULL,  
 attendance\_date DATE NOT NULL,  
 status ENUM('Present', 'Absent') NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (student\_id) REFERENCES students(student\_id) ON DELETE CASCADE,  
 FOREIGN KEY (lecturer\_id) REFERENCES lecturers(lecturer\_id) ON DELETE CASCADE,  
 FOREIGN KEY (course\_id) REFERENCES courses(course\_id) ON DELETE CASCADE  
);  
  
-- Notifications Table (Stores messages sent by Admins or Lecturers)  
CREATE TABLE notifications (  
 notification\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 sender\_id INT NOT NULL,  
 receiver\_id INT,  
 receiver\_type ENUM('Student', 'Lecturer', 'All') NOT NULL,  
 message TEXT NOT NULL,  
 sent\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (sender\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
```

## 5.2 Entity-Relationship Diagram

![ER Diagram](path/to/er-diagram.png)

# 6. Appendices

- Appendix A: Glossary  
- Appendix B: Acronyms  
- Appendix C: References