MULUNGUSHI UNIVERSITY

SCHOOL OF ENGINEERING AND TECHNOLOGY

COMPUTER SCIENCE DEPARTMENT

Requirements Gathering & Analysis

Class Attendance and Grades Management System

Group J Members

Edwin Mwansa	202305221
Adon Chinyamuka	202303887
Racheal Sililo	202303989
Kafiswe Chimputu	202302874
Elijah Manda	202302305
Beatrice Mulenga	202304152
Chatumba Mwnanza	202304448
Mapalo Sanika	202207131
Nissi Masobano	202305336
Kansamba Auxiria	202206607
Alfred Phiri	202201419

1. Vision Summary

The **Class Attendance and Grades Management System** is a comprehensive mobile application with a RESTful backend designed to streamline academic administration at Mulungushi University. The system provides two distinct mobile interfaces: a **Lecturer View** with full administrative privileges for managing courses, recording attendance, and entering grades; and a **Student View** for accessing personal attendance records and grades.

Built on Spring Boot with MariaDB database backend, the system features secure authentication, role-based access control, and comprehensive audit logging. The mobile application communicates with the backend through RESTful APIs (tested via Postman), ensuring data integrity and security while providing real-time access to academic information.

By digitizing attendance tracking and grade management with proper access controls and audit trails, the system aims to improve accuracy, accountability, save administrative time, provide transparent access to academic records, and ensure compliance with data privacy requirements.

---

2. Stakeholder Identification

Primary Stakeholders

1. **Lecturers/Instructors**
    - Full system access through Lecturer View
    - Manage courses, mark attendance, record grades
    - Generate reports and analytics
    - View audit logs for their courses
2. **Students**
    - Limited access through Student View
    - View personal attendance records
    - View personal grades and performance
    - Monitor academic progress

Secondary Stakeholders

3. **Department Administrators**
    - Access to aggregate reports
    - User account management oversight
    - System configuration and policies
    - Audit log review
4. **System Administrators/ICT Support**
    - Backend system maintenance

    - Database management (MariaDB)
    - User authentication management
    - Security monitoring and incident response
    - API management and monitoring
5. **University Management**
    - Access to institutional analytics
    - Compliance oversight
    - Policy enforcement monitoring

---

3. Problem Statement

Current academic management processes at Mulungushi University face multiple challenges:

Operational Challenges:

- **Time-Consuming:** Manual attendance and grade recording waste valuable time
- **Error-Prone:** Paper-based processes lead to data entry mistakes
- **No Access Controls:** Paper records lack proper security and access management
- **Limited Transparency:** Students cannot easily access their attendance and grade information
- **No Audit Trail:** No record of who made changes and when
- **Data Silos:** Attendance and grades managed in isolation

Security and Compliance Issues:

- **No Authentication:** Anyone can access paper records
- **No Accountability:** Cannot trace unauthorized changes
- **Privacy Concerns:** Student data not adequately protected
- **No Role Separation:** All users have equal access

Technical Limitations:

- **No Remote Access:** Records only available physically
- **No Real-Time Updates:** Information delayed by manual processes
- **Difficult Integration:** Cannot easily share data with other systems

These issues result in administrative burden, security vulnerabilities, lack of transparency, and inability to ensure data integrity and compliance with privacy regulations.

---

4. Project Goals

Goal 1: Implement Secure Authentication System

Develop multi-factor authentication with unique login credentials for all users (lecturers and students), ensuring only authorized access to the system.

Goal 2: Establish Role-Based Access Control

Implement distinct permission levels with Lecturer View (full privileges) and Student View (read-only personal data), ensuring proper separation of duties.

Goal 3: Enable Comprehensive Audit Logging

Track all system activities (logins, data modifications, access attempts) with timestamps and user identification for accountability and compliance.

Goal 4: Build RESTful API Backend

Create a Spring Boot backend with RESTful APIs tested via Postman, providing secure, scalable communication between mobile app and MariaDB database.

Goal 5: Provide Student Self-Service Portal

Enable students to independently view their attendance records and grades through mobile app, reducing administrative queries.

Goal 6: Centralize Data Management

Implement MariaDB database with proper normalization, indexing, and transaction management for data integrity and performance.

Goal 7: Ensure Data Privacy and Security

Implement encryption, secure API endpoints, SQL injection prevention, and compliance with data protection regulations.

5. Functional Requirements

FR1: User Authentication and Authorization

Priority: Must Have
User Story: As a system user, I want to log in securely with my unique credentials so that I can access appropriate system features based on my role.

Requirements:

- FR1.1: System shall provide login screen with username and password fields
- FR1.2: System shall authenticate users against MariaDB user credentials table
- FR1.3: System shall generate JWT tokens upon successful authentication
- FR1.4: System shall enforce password complexity requirements (8+ chars, mixed case, numbers)
- FR1.5: System shall lock accounts after 5 failed login attempts
- FR1.6: System shall support password reset functionality
- FR1.7: System shall maintain session timeout of 30 minutes inactivity
- FR1.8: System shall log all authentication attempts (successful and failed)
- FR1.9: System shall assign user roles (LECTURER, STUDENT) during account creation

FR2: Role-Based Access Control

Priority: Must Have
User Story: As a system administrator, I want different user roles with specific permissions so that users can only access features appropriate to their role.

Requirements:

- FR2.1: System shall implement LECTURER role with full CRUD operations
- FR2.2: System shall implement STUDENT role with read-only access to personal data
- FR2.3: System shall prevent students from accessing other students' records
- FR2.4: System shall prevent students from modifying any data
- FR2.5: System shall validate user permissions for every API request
- FR2.6: System shall return HTTP 403 Forbidden for unauthorized access attempts
- FR2.7: System shall log all access control violations

FR3: Audit Logging System

Priority: Must Have
User Story: As a system administrator, I want comprehensive audit logs so that I can track all system activities and investigate incidents.

Requirements:

- FR3.1: System shall log all user login/logout events with timestamps
- FR3.2: System shall log all data modifications (INSERT, UPDATE, DELETE) with user ID
- FR3.3: System shall log API endpoint access with request parameters
- FR3.4: System shall capture before and after values for data modifications
- FR3.5: System shall store audit logs in dedicated MariaDB table
- FR3.6: System shall prevent modification or deletion of audit logs
- FR3.7: System shall provide audit log query functionality for administrators
- FR3.8: System shall include IP address and device information in logs

- FR3.9: System shall log failed authentication attempts
- FR3.10: System shall retain audit logs for minimum 1 year

FR4: RESTful API Backend (Spring Boot)

Priority: Must Have
User Story: As a mobile app, I want secure RESTful APIs so that I can communicate with the backend database reliably.

Requirements:

- FR4.1: System shall implement Spring Boot REST controllers for all entities
- FR4.2: System shall use HTTP methods appropriately (GET, POST, PUT, DELETE)
- FR4.3: System shall return proper HTTP status codes (200, 201, 400, 401, 403, 404, 500)
- FR4.4: System shall validate all input data using Spring Validation
- FR4.5: System shall implement JWT-based authentication for API security
- FR4.6: System shall use JSON format for request/response payloads
- FR4.7: System shall implement exception handling with meaningful error messages
- FR4.8: System shall support pagination for large datasets
- FR4.9: System shall implement CORS configuration for mobile app access
- FR4.10: System shall provide comprehensive API documentation

FR5: Course Management (Lecturer View)

Priority: Must Have
User Story: As a lecturer, I want to manage my courses through the mobile app so that I can organize my teaching schedule.

Requirements:

- FR5.1: API shall allow lecturers to create courses (POST /api/courses)
- FR5.2: API shall allow lecturers to retrieve their courses (GET /api/courses)
- FR5.3: API shall allow lecturers to update course details (PUT /api/courses/{id})
- FR5.4: API shall allow lecturers to delete courses (DELETE /api/courses/{id})
- FR5.5: Mobile app shall display course list with enrollment statistics
- FR5.6: Mobile app shall provide course search and filter functionality
- FR5.7: System shall associate courses with lecturer who created them
- FR5.8: System shall prevent lecturers from modifying other lecturers' courses

FR6: Student Management (Lecturer View)

Priority: Must Have
User Story: As a lecturer, I want to manage student enrollments so that I can track the correct students in my courses.

Requirements:

- FR6.1: API shall allow creating student accounts (POST /api/students)
- FR6.2: API shall generate unique login credentials for new students
- FR6.3: API shall allow enrolling students in courses (POST /api/enrollments)
- FR6.4: API shall allow removing students from courses (DELETE /api/enrollments/{id})
- FR6.5: API shall prevent duplicate registration numbers
- FR6.6: API shall retrieve enrolled students per course (GET /api/courses/{id}/students)
- FR6.7: Mobile app shall provide bulk enrollment functionality
- FR6.8: System shall send login credentials to students upon account creation

FR7: Attendance Tracking (Lecturer View)

Priority: Must Have
User Story: As a lecturer, I want to mark attendance through the mobile app so that records are immediately saved to the database.

Requirements:

- FR7.1: API shall create attendance sessions (POST /api/sessions)
- FR7.2: API shall record attendance marks (POST /api/attendance)
- FR7.3: API shall retrieve attendance records (GET /api/attendance)
- FR7.4: API shall calculate attendance percentages (GET /api/courses/{id}/attendance-stats)
- FR7.5: Mobile app shall display enrolled students with toggle switches
- FR7.6: Mobile app shall save attendance in real-time via API
- FR7.7: System shall timestamp all attendance records
- FR7.8: System shall prevent modifying attendance after 24 hours (configurable)

FR8: Grade Management (Lecturer View)

Priority: Must Have
User Story: As a lecturer, I want to record and update grades so that student performance is accurately tracked.

Requirements:

- FR8.1: API shall allow adding grades (POST /api/grades)
- FR8.2: API shall allow updating grades (PUT /api/grades/{id})
- FR8.3: API shall allow deleting grades (DELETE /api/grades/{id})
- FR8.4: API shall retrieve grades per course (GET /api/courses/{id}/grades)
- FR8.5: API shall calculate average scores (GET /api/grades/averages)
- FR8.6: Mobile app shall provide gradebook interface with filtering
- FR8.7: System shall validate grade values (0-100 range)
- FR8.8: System shall log all grade modifications in audit log

FR9: Student Self-Service View

Priority: Must Have
User Story: As a student, I want to view my attendance and grades so that I can monitor my academic progress.

Requirements:

- FR9.1: API shall retrieve student's personal attendance (GET /api/students/me/attendance)
- FR9.2: API shall retrieve student's personal grades (GET /api/students/me/grades)
- FR9.3: API shall retrieve student's enrollment list (GET /api/students/me/courses)
- FR9.4: API shall calculate student's attendance percentage per course
- FR9.5: API shall calculate student's grade averages per course
- FR9.6: Mobile app shall provide dashboard showing overall statistics
- FR9.7: Mobile app shall display attendance history with dates
- FR9.8: Mobile app shall display grades by course and assessment type
- FR9.9: System shall restrict students to viewing only their own data
- FR9.10: Mobile app shall provide read-only interface (no edit capabilities)

FR10: Reporting and Analytics (Lecturer View)

Priority: Should Have
User Story: As a lecturer, I want to generate reports so that I can analyze student performance and share data with administration.

Requirements:

- FR10.1: API shall generate attendance reports (GET /api/reports/attendance)
- FR10.2: API shall generate grade reports (GET /api/reports/grades)
- FR10.3: API shall export data as CSV format
- FR10.4: API shall support date range filtering for reports
- FR10.5: Mobile app shall allow viewing and sharing report files

FR11: Database Management (MariaDB)

Priority: Must Have
User Story: As a backend system, I need reliable data persistence so that all academic records are safely stored.

Requirements:

- FR11.1: System shall use MariaDB as primary database
- FR11.2: System shall implement JPA/Hibernate for ORM
- FR11.3: System shall use connection pooling (HikariCP)
- FR11.4: System shall implement database transactions with rollback capability
- FR11.5: System shall create database indexes on frequently queried columns
- FR11.6: System shall implement foreign key constraints for referential integrity

- FR11.7: System shall use prepared statements to prevent SQL injection
- FR11.8: System shall implement database backup strategy
- FR11.9: System shall use UTF-8 encoding for international character support

---

6. Non-Functional Requirements

NFR1: Security

Priority: Must Have

- System shall encrypt passwords using BCrypt with minimum 10 rounds
- System shall use HTTPS for all API communications
- System shall implement JWT token expiration (30 minutes)
- System shall validate and sanitize all user inputs
- System shall protect against SQL injection using parameterized queries
- System shall implement CORS with whitelist of allowed origins
- System shall store sensitive data encrypted in database
- System shall comply with GDPR/data protection requirements
- System shall implement rate limiting on API endpoints (100 requests/minute)
- System shall mask sensitive data in logs

NFR2: Performance

Priority: Must Have

- API response time shall be < 500ms for 95% of requests
- Database queries shall complete within 300ms
- System shall support 100 concurrent users
- Mobile app shall cache frequently accessed data
- API shall implement pagination (max 50 records per page)
- System shall use database connection pooling
- Mobile app UI shall respond to user input within 100ms

NFR3: Reliability

Priority: Must Have

- System uptime shall be 99% during academic term
- Database shall implement automatic backup every 24 hours
- System shall recover from failures within 5 minutes
- System shall implement transaction rollback on errors
- API shall return meaningful error messages
- System shall log all errors for troubleshooting

NFR4: Usability

Priority: Must Have

- Mobile app shall follow Material Design 3 guidelines
- Interface shall be intuitive requiring minimal training
- Error messages shall be clear and actionable
- Mobile app shall support both portrait and landscape orientations
- All interactive elements shall be minimum 48dp
- System shall provide helpful validation messages
- Student view shall have simplified interface focused on viewing

NFR5: Compatibility

Priority: Must Have

- Mobile app shall run on Android 8.0 (API 26) and above
- Backend shall run on Java 17 or higher
- System shall support MariaDB 10.5 or higher
- API shall be RESTful and platform-agnostic
- System shall work on screen sizes from 5" to 10"

NFR6: Maintainability

Priority: Should Have

- Code shall follow Spring Boot best practices
- System shall use layered architecture (Controller-Service-Repository)
- API shall be documented using OpenAPI/Swagger
- Code shall include comprehensive logging
- Database schema shall support versioning and migrations
- System shall use dependency injection
- Code shall include unit and integration tests

NFR7: Scalability

Priority: Should Have

- Database shall handle 10,000+ student records
- System shall support 500+ courses
- System shall handle 100,000+ attendance records
- API shall support horizontal scaling
- Database shall support read replicas for scaling

NFR8: Auditability

Priority: Must Have

- All user actions shall be logged with timestamp and user ID
- System shall maintain immutable audit trail
- Audit logs shall be querable by administrators
- System shall retain logs for minimum 1 year
- Logs shall include sufficient detail for forensic analysis

NFR9: Testability

Priority: Must Have

- API shall be testable using Postman
- System shall provide test data seeding scripts
- API shall include automated test suite
- System shall support integration testing
- API documentation shall include example requests/responses

---

7. User Stories with Acceptance Criteria

US1: Lecturer Login

As a lecturer
I want to log in with my unique credentials
So that I can securely access the system

Acceptance Criteria:

- Given I'm on the login screen, when I enter valid lecturer credentials, then I'm authenticated and see the Lecturer View dashboard
- Given I enter invalid credentials, when I click login, then I see "Invalid username or password" error
- Given I fail login 5 times, when I try again, then my account is locked and I see appropriate message
- Given I'm logged in, when I'm inactive for 30 minutes, then I'm automatically logged out
- Given my login is successful, when I check audit logs, then I see my login event recorded with timestamp

US2: Student View Personal Grades

As a student
I want to view my grades on my mobile device
So that I can track my academic performance

Acceptance Criteria:

- Given I'm logged in as student, when I navigate to grades screen, then I see only my own grades organized by course
- Given I have grades in multiple courses, when I view grades screen, then I see average score per course
- Given I try to access another student's grades, when I make the API request, then I receive HTTP 403 Forbidden
- Given I'm viewing grades, when I pull to refresh, then latest grades are fetched from API
- Given grades are displayed, when I see them, then they are read-only with no edit options

US3: Lecturer Mark Attendance

As a lecturer
I want to mark attendance through mobile app connected to backend
So that attendance is immediately saved to MariaDB database

Acceptance Criteria:

- Given I'm logged in as lecturer, when I select a course and click "Mark Attendance", then I see all enrolled students via API call
- Given students are displayed, when I toggle presence switches and save, then POST request sends data to backend
- Given attendance is saved, when I check database, then I see new session and attendance records with timestamp
- Given I save attendance, when I check audit logs, then I see my action recorded
- Given attendance is older than 24 hours, when I try to modify it, then system prevents modification

US4: Student View Attendance Records

As a student
I want to view my attendance history
So that I can see which classes I attended

Acceptance Criteria:

- Given I'm logged in as student, when I navigate to attendance screen, then I see my attendance records for all enrolled courses
- Given I view attendance, when displayed, then I see dates, course names, and present/absent status
- Given I have multiple courses, when I view attendance, then I see percentage per course
- Given I try to access another student's attendance, when I make API request, then I receive HTTP 403 Forbidden
- Given I'm viewing attendance, when I pull to refresh, then latest data is fetched via GET /api/students/me/attendance

US5: Lecturer Record Grades

As a lecturer
I want to record student grades through the mobile app
So that grades are securely stored in the database

Acceptance Criteria:

- Given I'm in gradebook, when I select a student and assessment, then I can enter grade value (0-100)
- Given I enter valid grade, when I save, then POST /api/grades sends data to backend
- Given grade is saved, when I verify in database, then I see new record with lecturer ID and timestamp
- Given I enter invalid grade (<0 or >100), when I save, then I see validation error from API
- Given I modify existing grade, when I check audit log, then I see both old and new values recorded

US6: System Administrator View Audit Logs

As a system administrator
I want to view comprehensive audit logs
So that I can monitor system activity and investigate incidents

Acceptance Criteria:

- Given I have admin access, when I query audit logs API, then I see all logged events with timestamps
- Given I search for specific user, when I filter logs, then I see all activities for that user
- Given I investigate failed logins, when I review logs, then I see all authentication attempts with IP addresses
- Given I examine data modifications, when I view logs, then I see before and after values
- Given logs are displayed, when I try to modify them, then system prevents any changes

US7: Test APIs with Postman

As a developer
I want to test API endpoints using Postman
So that I can verify functionality before mobile integration

Acceptance Criteria:

- Given I have Postman collection, when I send POST /api/auth/login with valid credentials, then I receive JWT token
- Given I have JWT token, when I include it in Authorization header, then I can access protected endpoints

- Given I test lecturer endpoints as student, when I send request, then I receive HTTP 403 Forbidden
- Given I test with invalid data, when I send request, then I receive appropriate validation error messages
- Given I test all CRUD operations, when successful, then I see proper HTTP status codes (200, 201, 204)

US8: Access Control Enforcement

As a system security component
I want to enforce role-based permissions
So that users can only perform authorized actions

Acceptance Criteria:

- Given a student user, when they try to POST /api/grades, then request is rejected with 403 Forbidden
- Given a lecturer user, when they try to access another lecturer's courses, then request is rejected
- Given an unauthenticated request, when sent to protected endpoint, then system returns 401 Unauthorized
- Given a request without valid JWT, when sent, then system rejects it
- Given all permission violations, when they occur, then they are logged in audit trail

---

8. Priority Tags (MoSCoW Analysis)

Must Have (Critical for MVP)

- User authentication system with unique logins (FR1)
- Role-based access control (Lecturer/Student) (FR2)
- Comprehensive audit logging (FR3)
- RESTful API backend with Spring Boot (FR4)
- Course and student management (FR5, FR6)
- Attendance tracking APIs (FR7)
- Grade management APIs (FR8)
- Student self-service view (FR9)
- MariaDB database integration (FR11)
- Security requirements (NFR1)
- Performance requirements (NFR2)
- API testability with Postman (NFR9)

Should Have (Important but not critical)

- Reporting and CSV export (FR10)
- Advanced analytics dashboard
- Password reset functionality
- Email notifications
- Mobile app offline mode with sync
- Biometric authentication
- Multi-language support

Could Have (Desirable enhancements)

- Mobile push notifications
- Attendance geofencing (location-based check-in)
- Grade distribution charts
- Predictive analytics for at-risk students
- QR code-based attendance
- Integration with university LMS
- Parent portal access

Won't Have (Out of scope for this version)

- Web-based interface
- Video conferencing integration
- Automated grading algorithms
- Plagiarism detection
- Assignment submission system
- Discussion forums
- Course content management

---

9. Assumptions

Technical Assumptions

1. All users have Android devices with API 26+ and internet connectivity
2. University provides MariaDB server infrastructure
3. Spring Boot backend will run on university servers
4. Mobile app requires constant internet for API communication
5. JWT tokens are securely stored on mobile devices
6. University network allows HTTPS traffic on required ports
7. Postman is used for all API development and testing

Business Assumptions

1. Each lecturer and student receives unique login credentials

2. Student registration numbers are unique across university
3. Lecturers manage their own course enrollments
4. Attendance policies allow 24-hour modification window
5. Grade scale is 0-100 numerical system
6. Audit logs are reviewed periodically by administrators
7. System administrators have database access for backup/recovery

Security Assumptions

1. University provides secure network infrastructure
2. SSL certificates are properly configured
3. Database server has firewall protection
4. Users are responsible for password security
5. Device security (screen lock) is user's responsibility
6. University has data breach response procedures

User Assumptions

1. Users have basic smartphone operation skills
2. Lecturers will use system consistently each class session
3. Students will check their grades/attendance regularly
4. Users understand their role-specific permissions
5. Failed login attempts indicate potential security issues

---

10. Constraints

Technical Constraints

- **Platform:** Android mobile app (no iOS or web initially)
- **Backend:** Must use Spring Boot framework with Java
- **Database:** Must use MariaDB (not MySQL, PostgreSQL, or others)
- **Architecture:** Must implement RESTful API design
- **Testing:** Must support Postman for API testing
- **Authentication:** Must use JWT token-based authentication
- **No Offline Mode:** MVP requires internet connectivity

Infrastructure Constraints

- **Server Resources:** Limited to university-provided infrastructure
- **Network:** Dependent on university network reliability
- **Storage:** Database size limited by university server capacity
- **Bandwidth:** API payload size should be minimized

Security Constraints

- **Compliance:** Must comply with university data protection policies
- **Access:** Database access restricted to backend application
- **Authentication:** Cannot use third-party authentication (must be internal)
- **Audit:** All user actions must be logged for minimum 1 year

Development Constraints

- **Timeline:** Must complete within semester timeframe
- **Team:** Student development team with limited availability
- **Budget:** No commercial budget for third-party services
- **Tools:** Must use free/open-source tools (Postman, Spring Boot, MariaDB)

Business Constraints

- **Training:** Minimal training time available for users
- **Support:** Limited post-deployment support resources
- **Data Migration:** No existing digital data to migrate
- **Integration:** Cannot integrate with other university systems in MVP

---

11. Risks and Mitigation Strategies

Risk 1: Security Breach / Unauthorized Access

Probability: Medium | **Impact:** Critical
Mitigation:

- Implement strong authentication with password complexity requirements
- Use JWT with short expiration times
- Implement comprehensive audit logging
- Regular security testing and penetration testing
- Account lockout after failed login attempts
- Encrypt all data transmission with HTTPS
- Regular security awareness training for users
- Implement IP whitelisting if feasible

Risk 2: Database Performance Issues

Probability: Medium | **Impact:** High
Mitigation:

- Implement database indexing on frequently queried columns

- Use connection pooling (HikariCP)
- Implement query optimization
- Use pagination for large datasets
- Monitor database performance metrics
- Plan database scaling strategy
- Regular database maintenance and optimization

Risk 3: API Availability / Downtime

Probability: Medium | **Impact:** High
Mitigation:

- Implement proper error handling in mobile app
- Use connection timeouts and retry logic
- Monitor API uptime and performance
- Implement health check endpoints
- Have database backup and recovery procedures
- Document disaster recovery plan
- Consider implementing API rate limiting to prevent overload

Risk 4: Incomplete Audit Trail

Probability: Low | **Impact:** High
Mitigation:

- Design audit logging from start (not added later)
- Test audit logging thoroughly
- Make audit logs immutable in database
- Implement automated audit log monitoring
- Regular audit log review procedures
- Ensure adequate storage for log retention

Risk 5: Role Permission Misconfiguration

Probability: Medium | **Impact:** High
Mitigation:

- Implement role-based access control early in development
- Write comprehensive authorization tests
- Use Spring Security annotations consistently
- Regular security audits of permissions
- Document all permission rules clearly
- Test with different user roles extensively

Risk 6: Data Loss

Probability: Low | **Impact:** Critical
Mitigation:

- Implement automated daily database backups
- Test backup restoration procedures
- Use database transactions with rollback
- Implement foreign key constraints
- Regular backup integrity checks
- Document backup/recovery procedures

Risk 7: Poor Mobile App Performance

Probability: Medium | **Impact:** Medium
Mitigation:

- Implement data caching in mobile app
- Use pagination for lists
- Optimize API payload sizes
- Use background threading for network calls
- Implement loading indicators
- Test on various devices and network conditions

Risk 8: User Adoption Resistance

Probability: Medium | **Impact:** High
Mitigation:

- Design intuitive user interfaces
- Provide user training sessions
- Create user guides and video tutorials
- Gather user feedback early with prototypes
- Implement gradual rollout strategy
- Provide dedicated support during initial rollout
- Demonstrate time savings and benefits clearly

Risk 9: API Security Vulnerabilities

Probability: Medium | **Impact:** Critical
Mitigation:

- Use Spring Security framework
- Implement input validation on all endpoints
- Use parameterized queries (prevent SQL injection)
- Regular security code reviews

- Keep dependencies updated
- Use security scanning tools
- Test APIs with malicious inputs

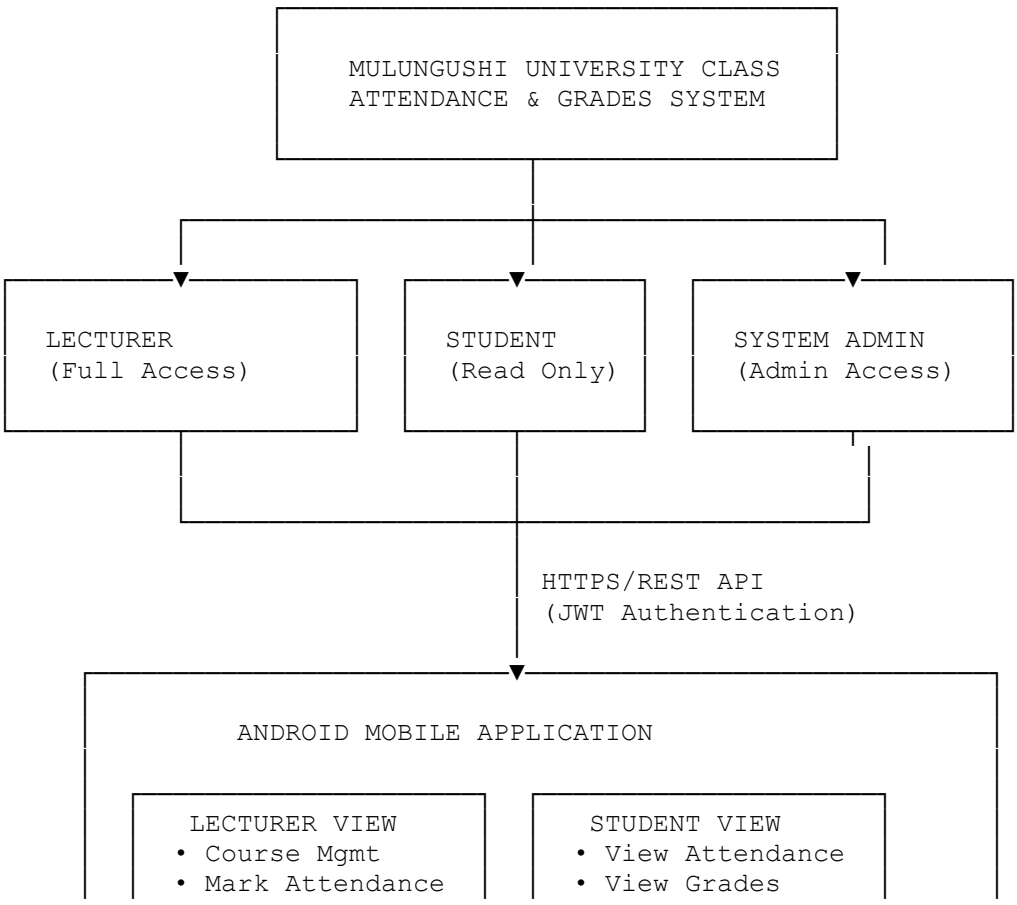
Risk 10: Inadequate Testing

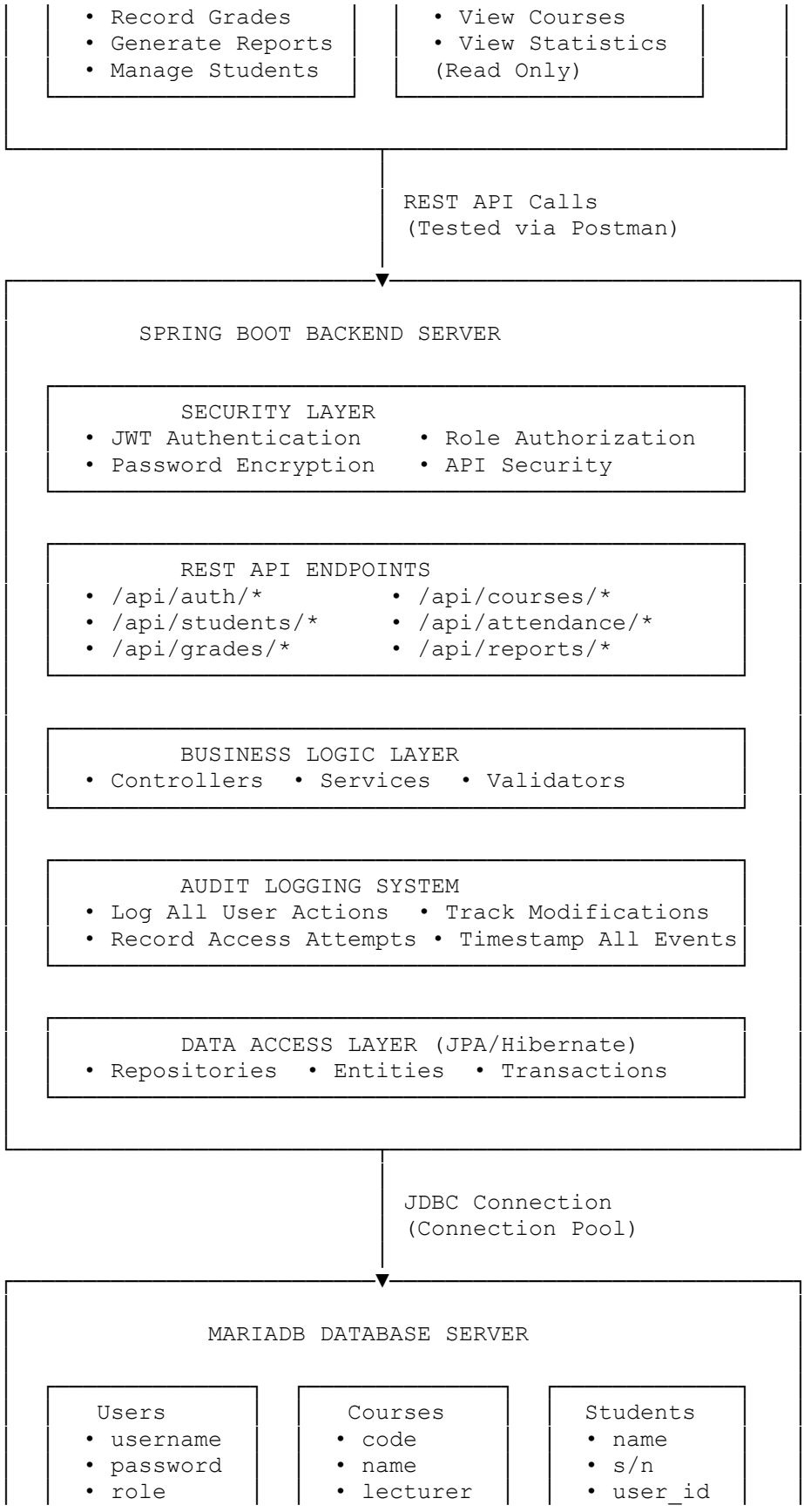
Probability: Medium | **Impact:** High
Mitigation:

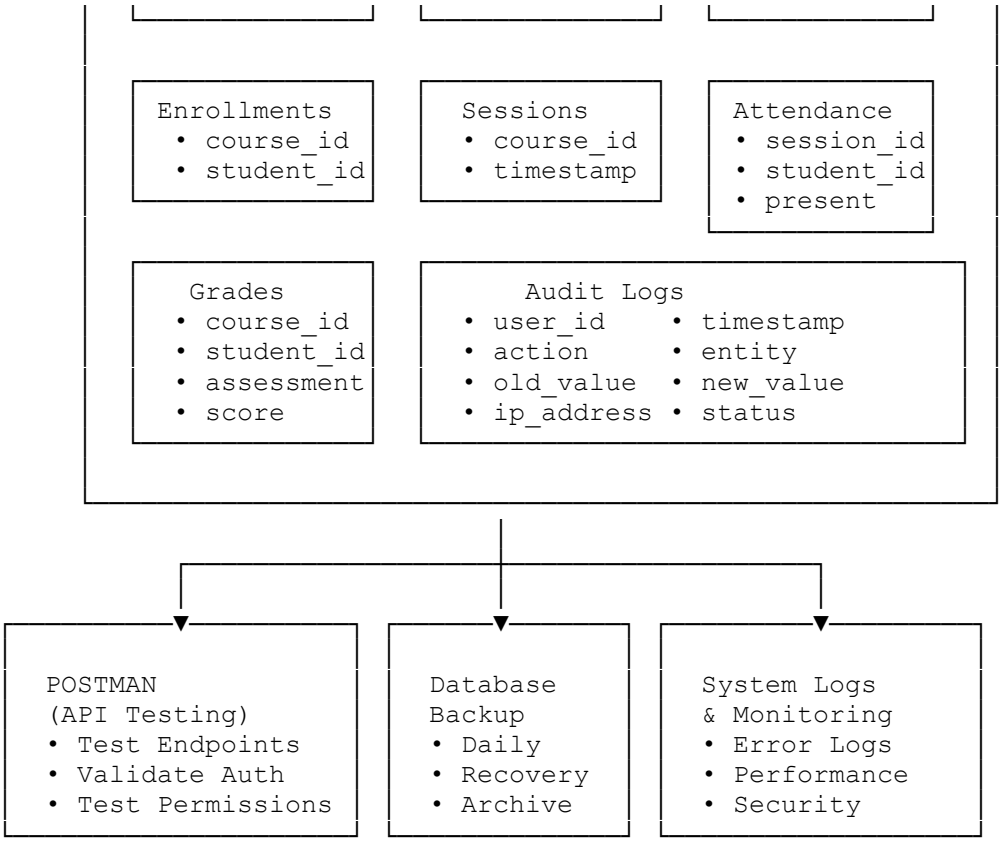
- Create comprehensive Postman test collection
- Write unit tests for all business logic
- Implement integration tests
- Test all user roles and permissions
- Conduct user acceptance testing
- Test error scenarios and edge cases
- Automated regression testing

---

12. Context Diagram




• Record Grades
• Generate Reports
• Manage Students
• View Courses
• View Statistics
(Read Only)
REST API Calls
(Tested via Postman)
SPRING BOOT BACKEND SERVER
SECURITY LAYER
• JWT Authentication
• Password Encryption
• Role Authorization
• API Security
REST API ENDPOINTS
• /api/auth/*
• /api/students/*
• /api/grades/*
• /api/courses/*
• /api/attendance/*
• /api/reports/*
BUSINESS LOGIC LAYER
• Controllers
• Services
• Validators
AUDIT LOGGING SYSTEM
• Log All User Actions
• Track Modifications
• Record Access Attempts
• Timestamp All Events
DATA ACCESS LAYER (JPA/Hibernate)
• Repositories
• Entities
• Transactions
JDBC Connection
(Connection Pool)
MARIADB DATABASE SERVER
Users
• username
• password
• role
Courses
• code
• name
• lecturer
Students
• name
• s/n
• user_id



External Systems and Actors:

- **Lecturers:** Full access to manage courses, attendance, grades, and students
- **Students:** Limited access to view personal attendance and grades only
- **System Administrators:** Manage user accounts, review audit logs, system configuration
- **Spring Boot Backend:** Central application server handling business logic and security
- **MariaDB Database:** Persistent storage for all application data
- **Postman:** Independent API testing tool for development and validation
- **Database Backup System:** Automated backup and recovery processes
- **Monitoring Tools:** System logging and performance monitoring

---

13. Initial Domain Glossary

Authentication: The process of verifying user identity through username and password credentials before granting system access.

Authorization: The process of determining whether an authenticated user has permission to perform a specific action based on their role.

JWT (JSON Web Token): A secure token format used for authentication that contains user claims and is signed to prevent tampering.

Role: A classification of user type (LECTURER or STUDENT) that determines system permissions and accessible features.

Audit Log: An immutable record of system events including user actions, timestamps, IP addresses, and data modifications.

API (Application Programming Interface): A set of RESTful endpoints that allow the mobile app to communicate with the backend server.

Spring Boot: A Java framework used to build the backend application with built-in features for REST APIs, security, and database access.

MariaDB: An open-source relational database management system used for persistent data storage.

REST (Representational State Transfer): An architectural style for APIs using HTTP methods (GET, POST, PUT, DELETE) for operations.

Endpoint: A specific URL path in the API that performs a particular function (e.g., /api/courses).

HTTP Status Code: A standardized response code indicating request success or failure (200, 401, 403, 404, 500).

Session: A single instance of attendance tracking for a course, identified by course ID and timestamp.

Attendance: A record indicating whether a student was present or absent for a specific session.

Course: An academic subject identified by a unique code and name, managed by a lecturer.

Student: A user with limited system access who can view personal attendance and grades.

Lecturer: A user with full system privileges who can manage courses, attendance, and grades.

Enrollment: The relationship between a student and a course indicating active participation.

Grade/Score: A numerical assessment (0-100) of student performance on an assessment.

Assessment: A type of evaluation (e.g., Test 1, Assignment 2, Final Exam, Project).

Registration Number: A unique identifier assigned to each student (e.g., 2021001).

Course Code: A unique alphanumeric identifier for a course (e.g., ICT381).

CRUD: Create, Read, Update, Delete - the four basic database operations.

ORM (Object-Relational Mapping): A technique (JPA/Hibernate) that maps Java objects to database tables.

Repository: A data access layer component that handles database operations for a specific entity.

Service Layer: Business logic layer that processes data between controllers and repositories.

Controller: API endpoint handler that receives requests and returns responses.

Entity: A Java class that maps to a database table (e.g., Course, Student, Grade).

Foreign Key: A database constraint that maintains referential integrity between related tables.

Transaction: A sequence of database operations that execute as a single unit (all succeed or all rollback).

Connection Pool: A cache of database connections to improve performance and manage resources.

BCrypt: A password hashing algorithm used to securely store passwords in the database.

CORS (Cross-Origin Resource Sharing): Security configuration allowing mobile app to access backend APIs.

Postman Collection: A group of saved API requests used for testing and documentation.

CSV (Comma-Separated Values): A file format used for exporting attendance and grade data.

Pagination: Dividing large datasets into smaller pages for better performance.

Rate Limiting: Restricting the number of API requests a user can make within a time period.

IP Address: A unique identifier for a device on the network, logged for security tracking.

Session Timeout: The period of inactivity after which a user is automatically logged out (30 minutes).

Account Lockout: Automatic account suspension after multiple failed login attempts (5 attempts).

---

14. Draft Backlog for Week 2

PART 1: Backend Foundation

Focus: Database, authentication, core API structure

STEP 1: Database Setup and Configuration

1. **Task 1.1:** Install and configure MariaDB server
    - Install MariaDB 10.5+
    - Create database and user accounts
    - Configure connection parameters
2. **Task 1.2:** Design database schema
    - Create ERD with all entities and relationships
    - Define primary keys, foreign keys, indexes
    - Document table structures
3. **Task 1.3:** Create database tables via Spring Boot
    - Create JPA entity classes (User, Course, Student, etc.)
    - Configure Hibernate to generate schema
    - Implement foreign key constraints
4. **Task 1.4:** Create audit log table
    - Design audit logs table structure
    - Create triggers or application-level logging
    - Test log insertion

STEP 2: Spring Boot Project Setup

1. **Task 2.1:** Initialize Spring Boot project
    - Create project with Spring Initializer
    - Add dependencies (Web, JPA, Security, MariaDB)
    - Configure application properties
2. **Task 2.2:** Configure database connection
    - Set up HikariCP connection pool
    - Configure JPA/Hibernate settings
    - Test database connectivity
3. **Task 2.3:** Implement layered architecture
    - Create package structure (controller, service, repository)
    - Set up dependency injection
    - Create base classes and interfaces
4. **Task 2.4:** Configure Spring Security
    - Add Spring Security dependencies
    - Configure security filter chain
    - Set up CORS configuration

STEP 3: Authentication System

Priority: Must Have:

1. **Task 3.1:** Create User entity and repository
    - Define User entity with roles
    - Create UserRepository interface
    - Implement custom queries
2. **Task 3.2:** Implement password encryption
    - Configure BCrypt password encoder
    - Create password hashing utilities
    - Test password verification
3. **Task 3.3:** Implement JWT authentication
    - Add JWT library dependency
    - Create JWT utility class (generate, validate)
    - Implement JWT filter
4. **Task 3.4:** Create authentication endpoints
    - POST /api/auth/login
    - POST /api/auth/register
    - POST /api/auth/logout
    - Test with Postman
5. **Task 3.5:** Implement role-based authorization
    - Configure method security annotations
    - Create role checks (LECTURER, STUDENT)
    - Test permission enforcement

STEP 4: Audit Logging Implementation

Priority: Must Have:

1. **Task 4.1:** Create audit log entity and repository
    - Define AuditLog entity
    - Create AuditLogRepository
2. **Task 4.2:** Implement logging aspect
    - Create AOP aspect for method logging
    - Capture user context and request details
    - Log before/after values for modifications
3. **Task 4.3:** Create audit log query APIs
    - GET /api/audit/logs (admin only)
    - Implement filtering by user, date, action
    - Test with Postman

PART 2: Core API Development

Focus: Course, student, and enrollment management

STEP 5: Course Management APIs

Priority: Must Have:

1. **Task 5.1:** Create Course entity and repository
    - Define Course entity with lecturer relationship
    - Create CourseRepository
2. **Task 5.2:** Implement course service layer
    - Create CourseService with business logic
    - Implement CRUD operations
    - Add validation rule
3. **Task 5.3:** Create course REST endpoints
    - POST /api/courses (create)
    - GET /api/courses (list all for lecturer)
    - GET /api/courses/{id} (get one)
    - PUT /api/courses/{id} (update)
    - DELETE /api/courses/{id} (delete)
4. **Task 5.4:** Test course APIs with Postman
    - Create Postman collection
    - Test all CRUD operations
    - Test authorization rules

STEP 6: Student Management APIs

Priority: Must Have:

1. **Task 6.1:** Create Student entity and repository
    - Define Student entity with user relationship
    - Create StudentRepository
2. **Task 6.2:** Implement student service layer
    - Create StudentService
    - Implement student account creation
    - Generate login credentials
3. **Task 6.3:** Create student REST endpoints
    - POST /api/students (create)
    - GET /api/students (list all)
    - PUT /api/students/{id} (update)
    - DELETE /api/students/{id} (delete)
4. **Task 6.4:** Test student APIs with Postman
    - Test student CRUD operations
    - Verify unique registration numbers
    - Test lecturer-only access

STEP 7: Enrollment Management APIs

Priority: Must Have:

1. **Task 7.1:** Create Enrollment entity and repository
    - Define Enrollment entity (course-student junction)
    - Create EnrollmentRepository with custom queries
2. **Task 7.2:** Implement enrollment service
    - Create EnrollmentService
    - Implement enroll/unenroll logic
    - Prevent duplicate enrollments
3. **Task 7.3:** Create enrollment REST endpoints
    - POST /api/enrollments (enroll student)
    - DELETE /api/enrollments/{id} (unenroll)
    - GET /api/courses/{id}/students (get enrolled)
4. **Task 7.4:** Test enrollment APIs
    - Test enrollment operations
    - Test duplicate prevention
    - Verify course-student relationships

PART 3: Attendance and Grades

Focus: Attendance tracking and grade management

STEP 8: Attendance Management APIs

1. **Task 8.1:** Create Session and Attendance entities
    - Define Session entity
    - Define Attendance entity
    - Create repositories
2. **Task 8.2:** Implement attendance service layer
    - Create AttendanceService
    - Implement session creation logic
    - Implement bulk attendance marking
    - Calculate attendance percentages
3. **Task 8.3:** Create attendance REST endpoints
    - POST /api/sessions (create session)
    - POST /api/attendance (mark attendance)
    - GET /api/courses/{id}/attendance (get records)
    - GET /api/courses/{id}/attendance-stats (get percentages)
4. **Task 8.4:** Test attendance APIs with Postman
    - Test session creation
    - Test bulk attendance marking
    - Verify calculations
    - Test lecturer-only access

STEP 9: Grade Management APIs

Priority: Must Have:

1. **Task 9.1:** Create Grade entity and repository
    - Define Grade entity
    - Create GradeRepository with aggregation queries
2. **Task 9.2:** Implement grade service layer
    - Create GradeService
    - Implement grade CRUD operations
    - Calculate averages (per student, per course)
3. **Task 9.3:** Create grade REST endpoints
    - POST /api/grades (add grade)
    - PUT /api/grades/{id} (update grade)
    - DELETE /api/grades/{id} (delete grade)
    - GET /api/courses/{id}/grades (get course grades)
    - GET /api/grades/averages (get statistics)
4. **Task 9.4:** Test grade APIs with Postman
    - Test grade CRUD operations
    - Verify validation (0-100 range)
    - Test average calculations

PART 4: Student View and Mobile App

Focus: Student self-service APIs and mobile app development

STEP 10: Student Self-Service APIs

Priority: Must Have | **Estimated Effort:** 8 hours

1. **Task 10.1:** Create student-specific endpoints
    - GET /api/students/me/profile (get own profile)
    - GET /api/students/me/courses (get enrolled courses)
    - GET /api/students/me/attendance (get own attendance)
    - GET /api/students/me/grades (get own grades)
2. **Task 10.2:** Implement access restrictions
    - Ensure students only access own data
    - Return 403 for unauthorized access
    - Test permission enforcement
3. **Task 10.3:** Implement statistics calculations
    - Calculate attendance percentage per course
    - Calculate grade averages per course
    - Return summary statistics
4. **Task 10.4:** Test student APIs with Postman

- Test as student user
- Verify access restrictions
- Test data accuracy

STEP 11: Mobile App - Authentication

Priority: Must Have:

1. **Task 11.1:** Set up Android project
   - Create new Android project
   - Add dependencies (Retrofit, Room, Material)
   - Configure build.gradle
2. **Task 11.2:** Implement API client
   - Create Retrofit service interfaces
   - Configure OkHttp client
   - Implement JWT token interceptor
3. **Task 11.3:** Create login screen
   - Design login UI
   - Implement login API call
   - Store JWT token securely
   - Handle authentication errors
4. **Task 11.4:** Implement role-based navigation
   - Detect user role from token
   - Navigate to appropriate view (Lecturer/Student)

STEP 12: Mobile App - Lecturer View

Priority: Must Have:

1. **Task 12.1:** Create course list screen
   - Display courses from API
   - Implement add/edit/delete
   - Show enrollment statistics
2. **Task 12.2:** Create attendance marking screen
   - Display enrolled students
   - Implement toggle switches
   - Save attendance via API
3. **Task 12.3:** Create gradebook screen
   - Display students and grades
   - Implement add/edit grades
   - Show calculated averages
4. **Task 12.4:** Create student management screen
   - Display all students
   - Implement add/edit/delete
   - Manage course enrollments

STEP 13: Mobile App - Student View

Priority: Must Have:

1. **Task 13.1:** Create student dashboard
    - Display enrolled courses
    - Show overall statistics
    - Summary cards for attendance/grades
2. **Task 13.2:** Create attendance view screen
    - Display attendance by course
    - Show attendance percentage
    - List all attendance records with dates
3. **Task 13.3:** Create grades view screen
    - Display grades by course
    - Show grade averages
    - List all assessments and scores
4. **Task 13.4:** Implement read-only interface
    - Remove all edit/delete options
    - Implement pull-to-refresh