PDF REPORT

NAME

Muhammad Masood Hussain (SP23-BSE-119).

Module:

- Auto Attendence (Muhammad Masood Hussain).

Chapter 1

ANALYSIS

1. Primary Actor:
   - Student: The main user who interacts with the system to mark attendance automatically.
2. Use Cases:
   - Check Proximity for Auto Attendance:
     - Likely uses geofencing or Bluetooth beacons to ensure the student is physically present (e.g., in a classroom) before triggering face recognition.
     - Ensures fairness by preventing proxy attendance.
   - Mark Auto Attendance:
     - Core functionality where the system identifies the student via face recognition and records attendance.
   - Record Attendance Date and Time:
     - Logs timestamps automatically to track punctuality and prevent manual errors.
   - Select Course for Attendance:
     - Allows students to choose the relevant course/session for which attendance is being marked (may integrate with a timetable system).
3. Implied Technology:
   - Face Recognition: Though not explicitly stated, "auto attendance" combined with "proximity check" suggests facial recognition is the primary method for identity verification.
   - Geolocation/Beacons: Proximity checks may use GPS, Wi-Fi, or Bluetooth to confirm the student's location.

Strengths of the Design

- Automation: Reduces manual effort for both students and faculty.
- Anti-Fraud Measures: Proximity checks + face recognition minimize proxy attendance.
- Audit Trail: Timestamp recording ensures transparency.

Potential Improvements

1. Add Secondary Actors:
   - Admin/Faculty: To manage courses, resolve discrepancies, or override errors.
   - System Database: To show where attendance data is stored.
2. Clarify Face Recognition:
   - Add a use case like "Verify Identity via Face Recognition" to make the technology explicit.
3. Error Handling:
   - Include use cases like "Handle Recognition Failure" (e.g., fallback to manual entry).
4. Integration:
   - Show connections to external systems (e.g., Timetable Module, Student Database).

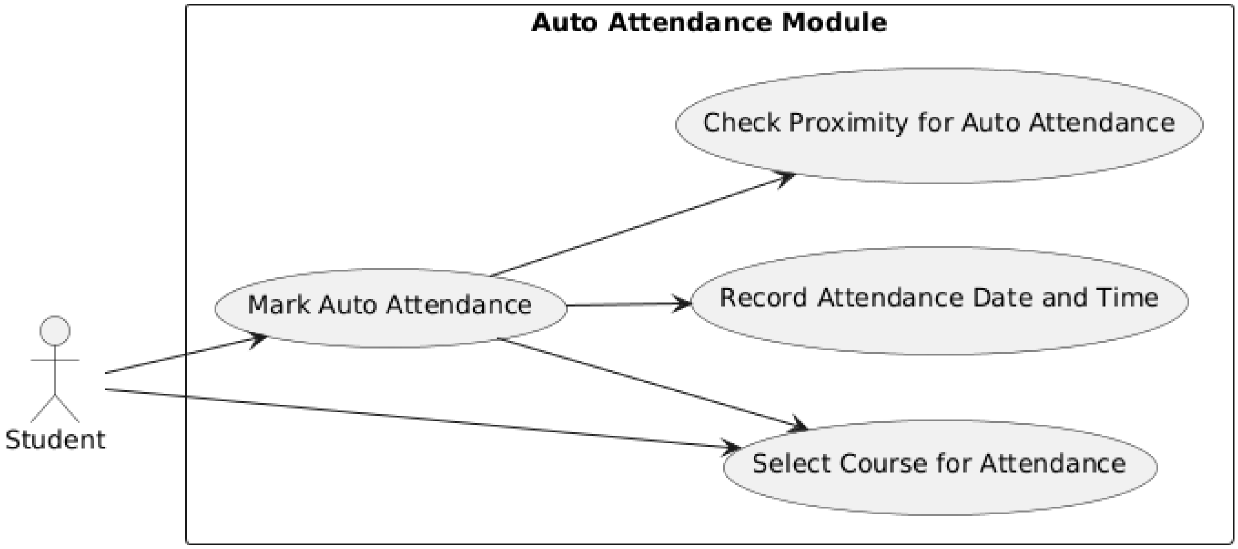
Workflow Summary

1. Student selects a course.
2. System verifies proximity (location).
3. Face recognition confirms identity.
4. Attendance is marked with a timestamp.

This design effectively addresses core attendance automation needs but could benefit from detailing error handling and system integrations. If face recognition is central, labeling it explicitly would improve clarity.

Chapter 2

USE CASE DAIGRAM


Auto Attendance Module
Check Proximity for Auto Attendance
Mark Auto Attendance
Record Attendance Date and Time
Select Course for Attendance
Student

1. Use Case: Check Proximity for Auto Attendance

Actor: Student, System
Preconditions:

- Student has installed the attendance app with location permissions enabled.
- Classroom geofence/Bluetooth beacons are configured.

Steps:

1. Student opens the app to mark attendance.
2. System activates GPS/Bluetooth to detect proximity to the classroom.
3. **If within range**:
   - App displays: "Proximity verified. Proceed to face recognition."
   - Triggers *Mark Auto Attendance* use case.
4. **If out of range**:
   - App shows error: "You must be in the classroom to mark attendance."

    - Attendance option is disabled.

Postconditions:

- Proximity status is logged (success/fail).

---

2. Use Case: Mark Auto Attendance via Face Recognition

Actor: Student
Preconditions:

- Proximity check is successful.
- Student's face is registered in the database.

Steps:

1. System activates the camera for face capture.
2. Student aligns face within the on-screen frame.
3. System compares face with registered images:
    - **On match**:
        - Attendance is marked.
        - Confirmation: "Attendance recorded for [Course Name]."
    - **On failure (3 attempts)**:
        - Redirects to *Handle Recognition Failure* use case.

Postconditions:

- Attendance status and timestamp are saved.

---

3. Use Case: Record Attendance Date and Time

Actor: System (automated)
Preconditions:

- Attendance is successfully marked.

Steps:

1. System fetches current date/time from the server (to prevent device spoofing).

2. Logs entry in the database with:
    - Student ID.
    - Course ID.
    - Timestamp (e.g., "2025-05-03 09:15:00").
3. Flags late entries (e.g., timestamp > 5 minutes after class start).

Postconditions:

- Data is immutable and auditable for faculty.

---

4. Use Case: Select Course for Attendance

Actor: Student
Preconditions:

- Student is enrolled in ≥1 active course.

Steps:

1. App displays a list of ongoing courses (pulled from the timetable).
2. Student selects the correct course (e.g., "CS101 - 9:00 AM").
3. System validates enrollment:
    - **If valid**: Proceeds to *Check Proximity*.
    - **If invalid**: Error: "You are not enrolled in this course."

Postconditions:

- Selected course is locked for attendance marking.

---

5. Use Case: Handle Recognition Failure

Actor: Student, Admin
Preconditions:

- Face recognition fails after 3 attempts.

Steps:

1. App suggests: "Try better lighting or remove obstructions (glasses/mask)."

2. **If retries fail**:
    - Generates a temporary OTP sent to the student’s email.
    - Student enters OTP to manually mark attendance.
3. **If OTP expires**:
    - Notifies admin to resolve manually (e.g., via admin dashboard).

Postconditions:

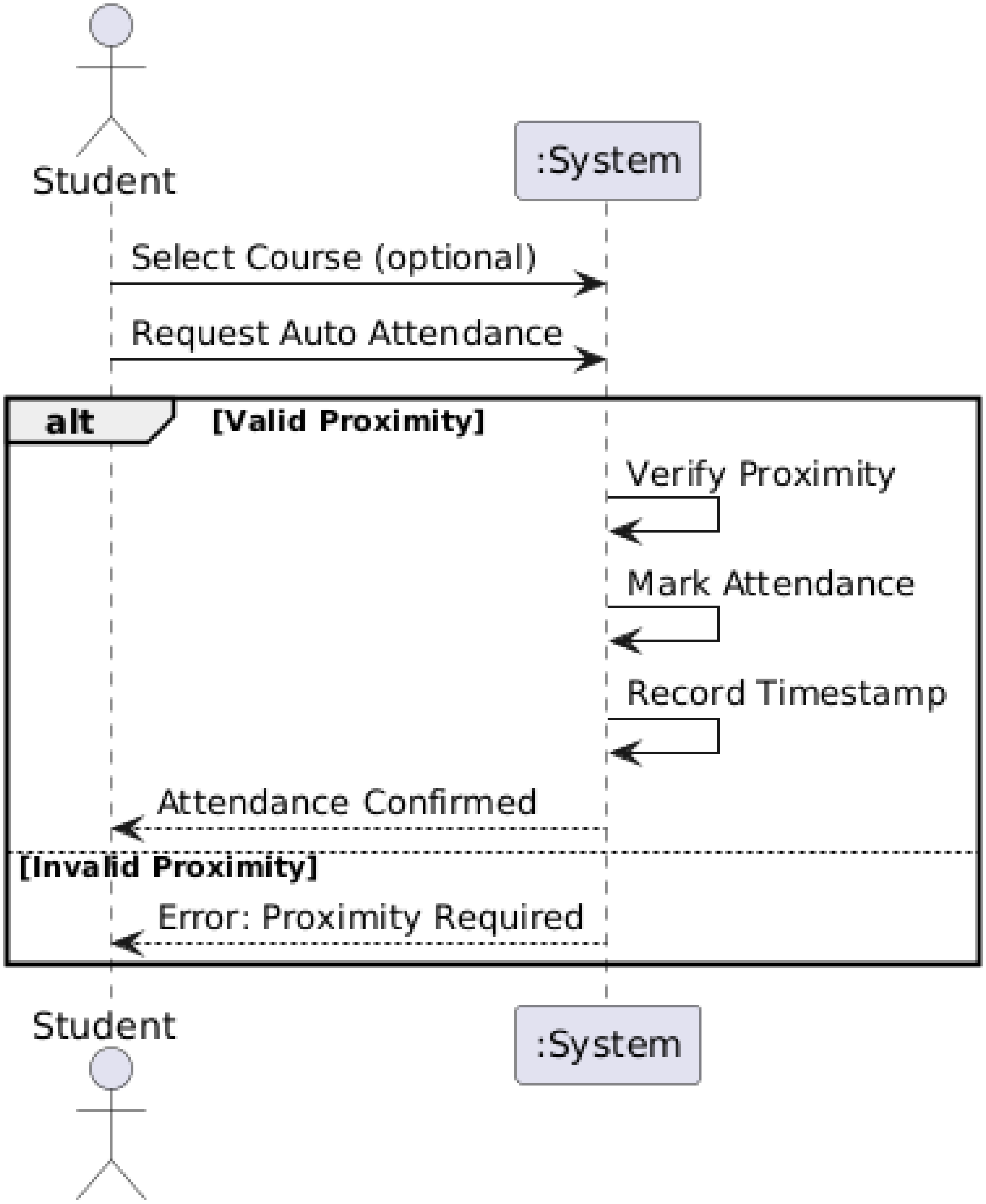
- Attendance is either marked or flagged for review.

Key System Features Implied:

- **Geofencing/Bluetooth**: Ensures physical presence.
- **Face Recognition**: Anti-spoofing measures (e.g., liveness detection).
- **Fallback Mechanisms**: OTP/manual override for edge cases.
- **Integration**: Syncs with course timetables and student databases.

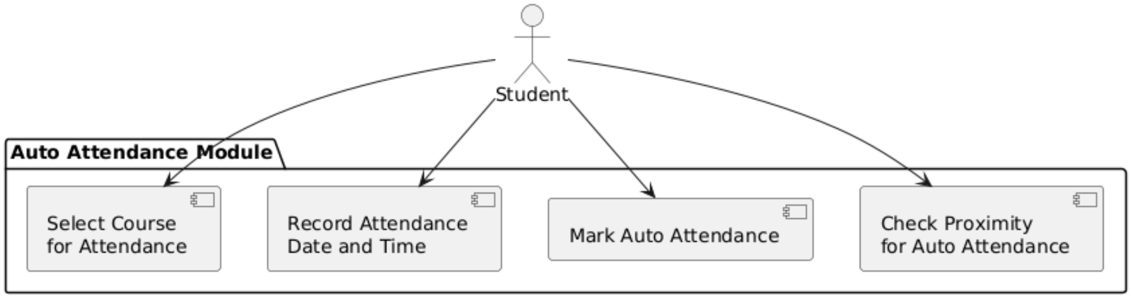
Chapter 3

SSDs



Chapter 4

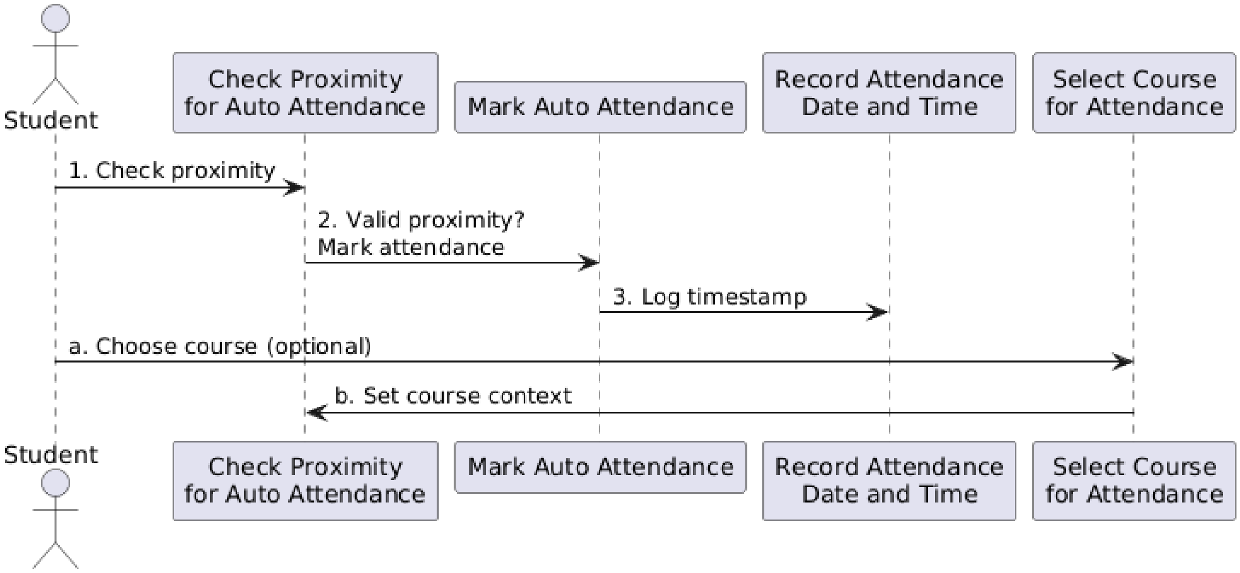
Package diagram



Student
Auto Attendance Module
Select Course for Attendance
Record Attendance Date and Time
Mark Auto Attendance
Check Proximity for Auto Attendance

Chapter 5

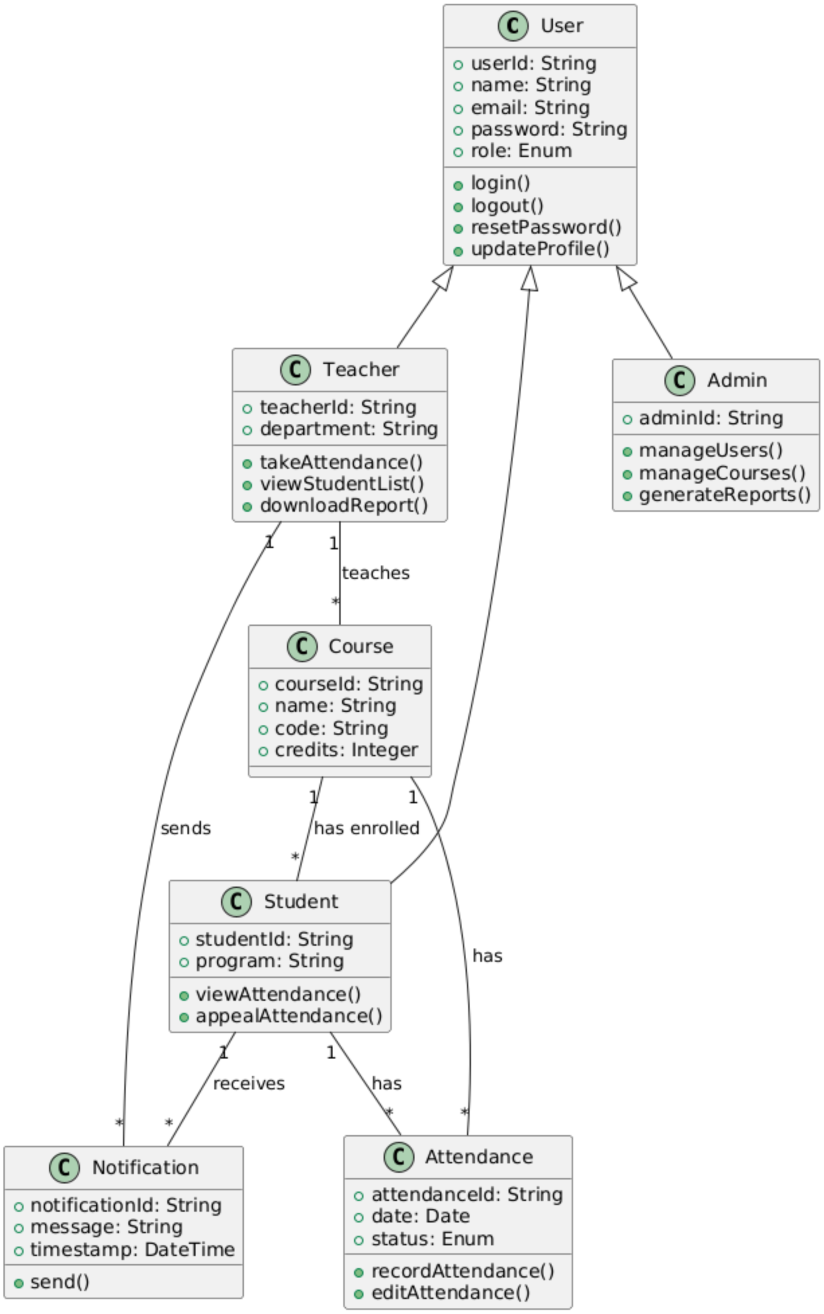
Collaboration Diagrams



Student
Check Proximity for Auto Attendance
Mark Auto Attendance
Record Attendance Date and Time
Select Course for Attendance
1. Check proximity
2. Valid proximity? Mark attendance
3. Log timestamp
a. Choose course (optional)
b. Set course context
Student
Check Proximity for Auto Attendance
Mark Auto Attendance
Record Attendance Date and Time
Select Course for Attendance

Chapter 6

Class Diagram



Chapter 7

Java Coding Standards for Auto Attendance Module

1. Naming Conventions

- Classes: Use PascalCase with descriptive names (e.g., AttendanceManager, FaceRecognitionService).
- Methods/Functions: Use camelCase with action-oriented names (e.g., verifyStudentPresence(), logAttendance()).
- Variables: Use camelCase (e.g., studentId, attendanceTimestamp). Avoid single-letter names.
- Constants: Use UPPER_SNAKE_CASE (e.g., MAX_RETRY_ATTEMPTS).
- Packages: Use lowercase, hierarchical naming (e.g., com.attendance.proximity).

---

2. Project Structure

- Modular Design:
  - model/: Contains POJOs (e.g., Student.java, Course.java).
  - service/: Business logic (e.g., AttendanceService.java).
  - util/: Helper classes (e.g., DateUtils.java).
  - exception/: Custom exceptions (e.g., InvalidAttendanceException.java).
- Separation of Concerns: Ensure classes follow the Single Responsibility Principle (SRP).

---

3. Error Handling

- Custom Exceptions: Define exceptions for specific scenarios (e.g., FaceRecognitionFailureException).
- Graceful Degradation: Provide fallback mechanisms (e.g., OTP verification if face recognition fails).
- Logging: Log errors with context (e.g., logger.error("Face recognition failed for student: " + studentId)).

---

4. Documentation

- JavaDoc: Document all public methods/classes with @param, @return, and @throws.
- Inline Comments: Explain complex logic or non-obvious decisions. Avoid redundant comments.
- README: Include setup instructions, dependencies, and API usage in the project root.

---

5. Security Practices

- Data Privacy: Encrypt sensitive data (e.g., face recognition templates).
- Input Validation: Validate all inputs (e.g., reject null/empty studentId).
- Authentication: Ensure only authorized users can override attendance (e.g., admin roles).

---

6. Logging & Monitoring

- Log Levels: Use INFO for operations, WARN for recoverable issues, and ERROR for failures.
- Structured Logs: Include timestamps, session IDs, and actionable details.
- Audit Trails: Log attendance actions (e.g., "Student marked present at [timestamp]").

---

7. Testing Standards

- Unit Tests: Cover all business logic (e.g., AttendanceServiceTest.java).
- Integration Tests: Test interactions between modules (e.g., face recognition + database).
- Edge Cases: Test failures (e.g., invalid face scans, network outages).

---

8. Code Formatting

- Indentation: Use 4 spaces (no tabs).
- Braces: Always include {}, even for single-line blocks.
- Line Length: Limit to 80–100 characters.
- Imports: Organize alphabetically, avoid wildcards (*).

---

9. Version Control

- Branching: Follow Git Flow (e.g., feature/face-recognition).
- Commit Messages: Use semantic prefixes (e.g., feat:, fix:, docs:).
- `.gitignore**: Exclude binaries, logs, and IDE files.

---

10. Performance & Optimization

- Database: Use batch inserts for bulk attendance records.
- Caching: Cache frequently accessed data (e.g., student face templates).
- Concurrency: Handle concurrent attendance marking (e.g., synchronized methods).

Tools to Enforce Standards

- Checkstyle: Enforce naming conventions and formatting.
- SonarQube: Monitor code quality and technical debt.
- JUnit: Automate testing.
- SpotBugs: Detect potential bugs.

Why These Standards?

- Readability: Clear names and structure make code easier to debug.
- Maintainability: Modular design allows scalable updates.
- Reliability: Error handling and testing reduce runtime failures.
- Security: Protects sensitive attendance data.