SMART CAMPUS MANAGEMENT SYSTEM

1. Objectives

1.1 Build a Smart Campus Management System using Java to handle students, faculty, courses, and timetables.
1.2 Integrate modules with proper input validation, custom exceptions, and JDBC- based database operations.
1.3 Ensure clean, modular, and scalable code suitable for real-world deployment.
1.4 Delivered a console-based, fully functional application that demonstrates professional software development practices.

2. Sprint Details

i. Sprint Number – 1
ii. Sprint Pod Name – **Smart Campus Management System**
iii. Pod Members –

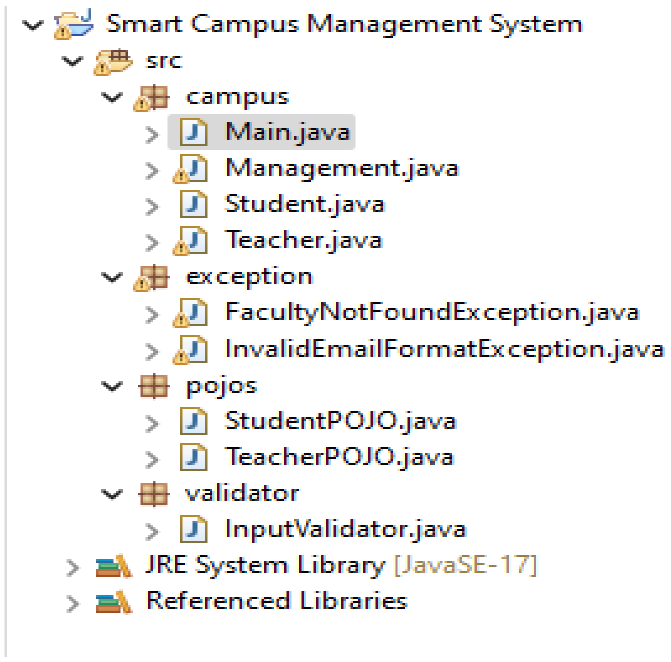
- Kanala Venkata Lakshmi Prasanna
- Kaveesh Bhat
- Kishor Kumar Parida
- Kunal Kanti Saha
- Mayank Anand
- Mugesh B
- Neha Mohanta
- Jobin Shery Mathew

iv. Submission Date – 19-06-2025

3. Deliverables

i. Integrated Project Code -
https://github.com/mayankanand2701/Smart-Campus-Management-System.git

ii. Project Documentation
i. Project Architecture



```
Smart Campus Management System
  src
    campus
      Main.java
      Management.java
      Student.java
      Teacher.java
    exception
      FacultyNotFoundException.java
      InvalidEmailFormatException.java
    pojos
      StudentPOJO.java
      TeacherPOJO.java
    validator
      InputValidator.java
  JRE System Library [JavaSE-17]
  Referenced Libraries
```

ii.Modules:

- Course Management
- Faculty Management
- Student Management
- Faculty Timetable Scheduling and Concurrent Course Allotment.

iii. Major Functionalities

- CRUD for Students, Faculty, and Courses.
- Faculty schedule management with slot conflict and room availability checks.
- Weekly timetable display in tabular format.
- Complete input validation (name, email, phone, DOB, etc.)

iv. What was Achieved

- Fully integrated campus management system.
- Proper module separation and clean code design.
- Realistic validations and error-checking.
- Successfully handled complex faculty-timetable logic with slot/room conflict resolution.

v. Key Challenges

- Designing slot conflict logic for timetable entries.
- Validating foreign key relationships (e.g., faculty-course links).
- Maintaining clean database connection handling and ensuring error handling didn't break user flow.

vi. Final Reflection and Learning

- Learned best practices for modular Java design.
- Improved skills in exception handling and JDBC.
- Understood the importance of user input validation in backend systems.

vii. Retrospective Notes

1. **What went well**
    - Modular Design & Single-Responsibility Methods
    - Each package (e.g., campus, POJOs, validator, exception) has a clear purpose, which kept code easy to navigate and test.
    - Refactoring into small, cohesive methods minimized duplication and simplified future enhancements.
    - Custom Exception Handling
    - Domain-specific exceptions (FacultyNotFoundException, InvalidEmailFormatException) provided precise feedback, improving user trust and troubleshooting efficiency.
    - Centralized error messages reduced clutter and made localization or UI integration straightforward.
    - Robust Input Validation
    - InputValidator caught most invalid entries at the source, preventing corrupt data in the database.
    - Regex-based checks for email, phone, and names significantly lowered runtime errors and rework.
    - Timetable Conflict Logic
    - Slot and room-conflict detection ensured realistic scheduling.
    - Immediate visual feedback (formatted timetable) boosted usability for faculty and management.
2. **Areas to Improve**
    - Automated Testing Coverage
    - Introduce JUnit or TestNG suites to cover core CRUD operations and validation logic.
    - Mock database layers with an in-memory DB (e.g., H2) to enable CI pipelines.
    - Logging & Monitoring
    - Implement a logging framework (Log4j2 or SLF4J) to capture info, warning, and error events.
    - Add log rotation and configurable log levels for production readiness.
    - User Interface Evolution
    - Transition from a console UI to a Swing/JavaFX GUI or a lightweight web frontend (Spring Boot + Thymeleaf/React).

- Provide role-based dashboards for students, faculty, and administrators.
- Security Enhancements
- Parameterize DB credentials via environment variables or a secure vault.

3. **Actionable Learnings for Future Projects**
    - Design modular architecture from the start.
    - Plan exception handling early and apply consistently.
    - Finalize database schema before development.
    - Prioritize basic testing and automation from early sprints.