5. Requirement Summary: University Course Enrolment System - A Modernisation Project

1. Business Scenario:

The "Nexus University" is a large, public university with a rapidly growing student body. Their existing course enrolment system, "LegacyEnroll," is a monolithic application that has been in use for over 20 years. It's difficult to maintain, slow, and lacks modern features like real-time updates, mobile access, and a user-friendly interface. The university's IT department has decided to launch a modernisation project to replace this system with a new, flexible, and scalable platform.

2. The New System – "NexusEnroll":

The new system, "NexusEnroll," must address the following key requirements:

- **Student Self-Service:** Students should be able to browse course catalogues, add/drop classes, view their schedule, and check their academic progress.
- **Faculty Management:** Faculty members need to see their class rosters, submit grades, and request course changes.
- **Administrator Control:** University administrators require tools to manage course offerings, student records, and generate reports on enrolment trends and faculty workload.
- **Scalability:** The system must be able to handle a high volume of simultaneous users, especially during peak enrolment periods.
- **Modern Accessibility:** The system should be accessible via **both** web browsers and mobile applications.

3. Detailed Functional Description of "NexusEnroll"

1. Student Module

The student module is the primary interface for the student body. Its core functions are:

- **Course Catalogue Browse:**
    - Students can search for courses by department, course number, keyword, and instructor.
    - The system must display real-time information for each course, including:
        - Course name and description
        - Instructor name
        - Available seats vs. total capacity
        - Schedule (days, times, location)
        - Prerequisites
    - **Use Case:** A student wants to browse all computer science courses for the upcoming semester that a specific professor teaches.
- **Registration and Enrolment:**
    - Students can add or drop courses from their enrolment list.
    - The system must validate all enrolment requests.
        - **Validation Rules:**
            - Prerequisite checks: The student must have completed all prerequisites for a course.
            - Capacity checks: The course cannot be full.
            - Time conflict checks: The requested course cannot overlap with an already-enrolled course.
    - **Use Case:** A student attempts to enrol for a course. The system must first check if the student has the necessary prerequisites. If they do, they must check if there's available capacity. If both conditions are met, the enrolment is confirmed, and the student's schedule and academic record are updated.

- **Personal Schedule Management:**
    - Students can view their current and past semester schedules.
    - The system should dynamically build and display a calendar-like view of their classes.
- **Academic Progress Tracking:**
    - Students can view a list of completed courses and the grades received.
    - The system should show which courses are still required for their degree program.

2. Faculty Module

The faculty module provides tools for instructors to manage their classes.

- **Class Roster Viewing:**
    - Instructors can view a real-time list of all students currently enrolled in their courses.
    - The roster should include student names, IDs, and contact information.
- **Grade Submission:**
    - Instructors can enter and submit final grades for each student in their courses at the end of the semester.
    - The system must have a process for grade approval (e.g., a "Pending" state before a final "Submitted" state).
    - **Use Case:** A professor submits a batch of grades for a class. The system should process these grades and update the student's academic records. If an error occurs during the process (e.g., an invalid grade is submitted), the system must handle it gracefully and allow the professor to correct it without losing other submitted grades.
- **Course Information Management:**
    - Instructors can submit requests to update course descriptions, add prerequisites, or change course capacity (these requests must be approved by an administrator).

3. Administrator Module

The administrator module is the central control panel for the entire system, with a focus on system-wide management and reporting.

- **Course & Program Management:**
    - Administrators can create, edit, and delete courses.
    - They can define and manage degree programs, including the required courses and credits.
- **Student & Faculty Management:**
    - Administrators can add, edit, and deactivate student and faculty accounts.
    - They can manually override enrolment rules (e.g., force-add a student into a full class).
- **Reporting & Analytics:**
    - The system must generate various reports:
        - Enrolment statistics by department and semester.
        - Faculty workload reports.
        - Course popularity trends.
    - **Use Case:** An administrator needs to generate a report on all courses in the Business school that are currently at over 90% capacity. The system should retrieve this data and present it in a clean, organised format (e.g., a table or spreadsheet).

4. System-Wide Requirements

These requirements apply across all modules and user types.

- **Notification System:**
    - The system must have a notification (e.g. email) mechanism. For example:
        - Students should be notified when a course they are waitlisted for becomes available.
        - Advisors should be notified when one of their advisees drops a critical course.

        - Administrators should be notified of any system-wide errors.

    - **Use Case:** A student drops a course. The notification system must automatically alert any waitlisted students that a spot has opened up. This process should be automated and decoupled from the core enrolment logic.

- **Transaction Management:**

    - All enrolment operations (adding/dropping a course) must be treated as transactions. This means either the entire operation succeeds (e.g., the student is enrolled, the class capacity is updated, and the schedule is modified) or the entire operation fails, leaving no partial state changes.

- **User Interface:**

    - The front-end should be a single-page application (SPA) that communicates with the back-end via APIs. This allows for a responsive user experience.

    - The same back-end services must be usable by both the web application and a future mobile application.

You don't understand anything until you learn it more than one way - Marvin Minsky

[END]