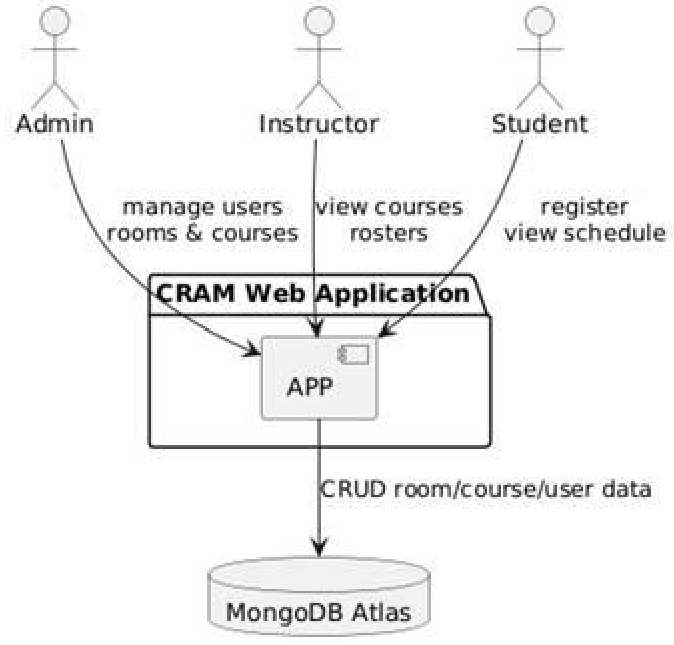
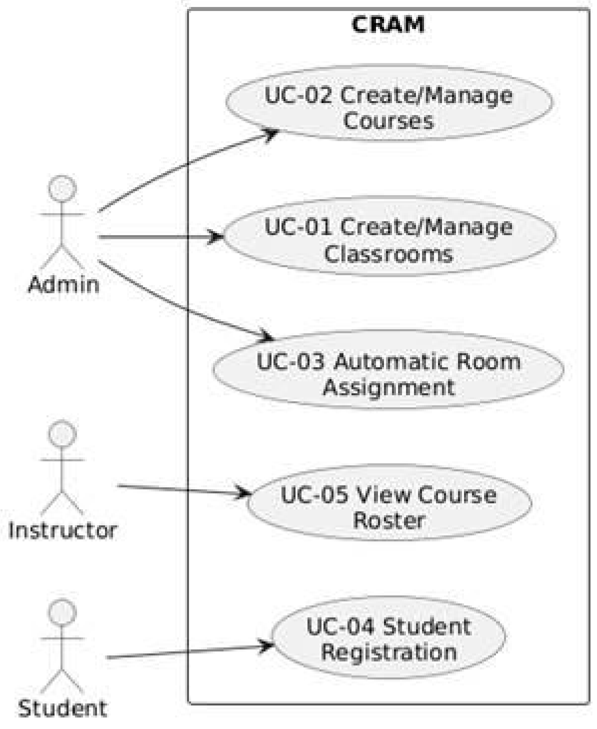
SD^2: Software Design Document

1. Project Overview

Class Room Assignment Manager (CRAM) is a web-based scheduling platform that streamlines the allocation of classrooms, courses, and enrollments for a university campus. Stakeholders include:
• Administrators – configure rooms, create courses, seed user accounts, and trigger automatic room assignment.
• Instructors – review the courses they teach and view enrolled rosters.
• Students – browse available sections, register or drop courses, and inspect their weekly schedule.

The system addresses the manual, error-prone process of matching courses to rooms with sufficient capacity and resources. By applying an automatic matching algorithm and a self-service portal, CRAM reduces administrative overhead and improves schedule accuracy.




CRAM
UC-02 Create/Manage Courses
UC-01 Create/Manage Classrooms
UC-03 Automatic Room Assignment
UC-05 View Course Roster
UC-04 Student Registration
Admin
Instructor
Student

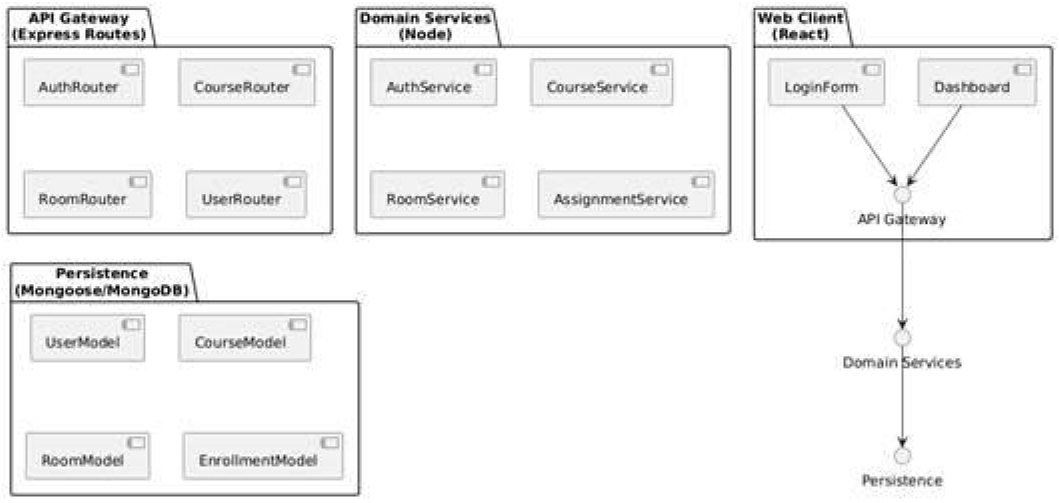
See the context and use-case diagrams above for the general model of external actors and primary system use-cases.

2. Architectural Overview

Two candidate architectures were evaluated:
1. Traditional LAMP stack: PHP/Apache serving server-rendered pages with a MySQL database.
2. MERN stack (MongoDB, Express, React, Node.js) – a RESTful service layer backed by MongoDB and a single-page React client.

The MERN approach was selected for its clear separation of concerns, rich React ecosystem, and built-in horizontal scalability afforded by stateless REST endpoints.

2.1 Subsystem Architecture



Web Client (React) – renders login, dashboards, and CRUD forms; communicates exclusively via the API Gateway.
API Gateway (Express Routers) – AuthRouter, CourseRouter, RoomRouter, and UserRouter expose REST endpoints, perform validation, and delegate to domain services.
Domain Services (Node) – CourseService, RoomService, AssignmentService, and AuthService hold business rules such as capacity checks and the automatic room-matching algorithm.
Persistence (Mongoose/MongoDB Atlas) – Mongoose models (User, Course, Room, Enrollment) map objects to BSON documents.
The design follows layered / service-oriented architectural styles, isolating presentation, application logic, and data access. Loose coupling between packages facilitates independent testing and future replacement (e.g., GraphQL gateway).

2.2 Deployment Architecture

Deployment uses three nodes connected over HTTPS/TCP:
• Client browser – executes the React bundle served at https://cram.example.edu (default localhost:3000 for development).
• Application server – Node/Express process listening on port 5000; containerised via Docker and reverse-proxied by Nginx.
• MongoDB Atlas cluster – hosted database, accessed with SRV connection string and TLS.

The client communicates with the application server via JSON/REST. Application server communicates with MongoDB using the MongoDB wire protocol through the mongoose ODM.

2.3 Persistent Data Storage

Persistent storage is handled by MongoDB. Key collections and their salient fields are:

Collection	Fields
users	_id:ObjectId, email, passwordHash, role ∈ {ADMIN,INSTRUCTOR,STUDENT}, firstName, lastName
rooms	_id, building, number, capacity, resources:[string]
courses	_id, code, name, capacity, instructor:User._id, room:Room._id, schedule:{day,time}
enrollments	_id, student:User._id, course:Course._id, status ∈ {ENROLLED, WAITLISTED}

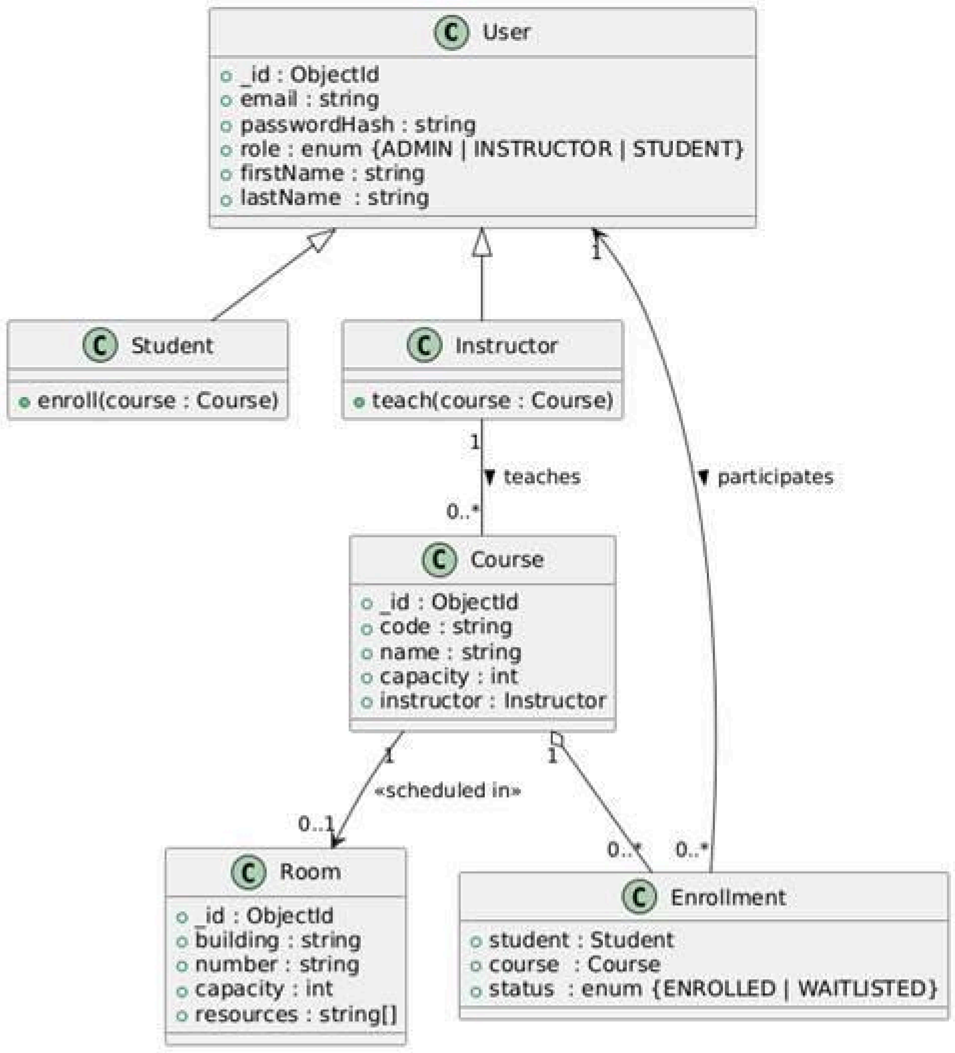
Mongoose schemas enforce validation and provide helper methods (e.g., virtuals for roster lookup).

2.4 Global Control Flow

CRAM is event-driven:
• Frontend React components dispatch Axios requests upon user interaction.
• Express routes are executed asynchronously within Node's single-threaded event loop.
• No periodic real-time deadlines exist, although the AssignmentService can be triggered manually or by a scheduled CRON job in the future.
• Concurrency inside Node relies on the non-blocking I/O model; Mongo queries are executed in parallel on the database cluster.

3. Detailed System Design

3.1 Static View



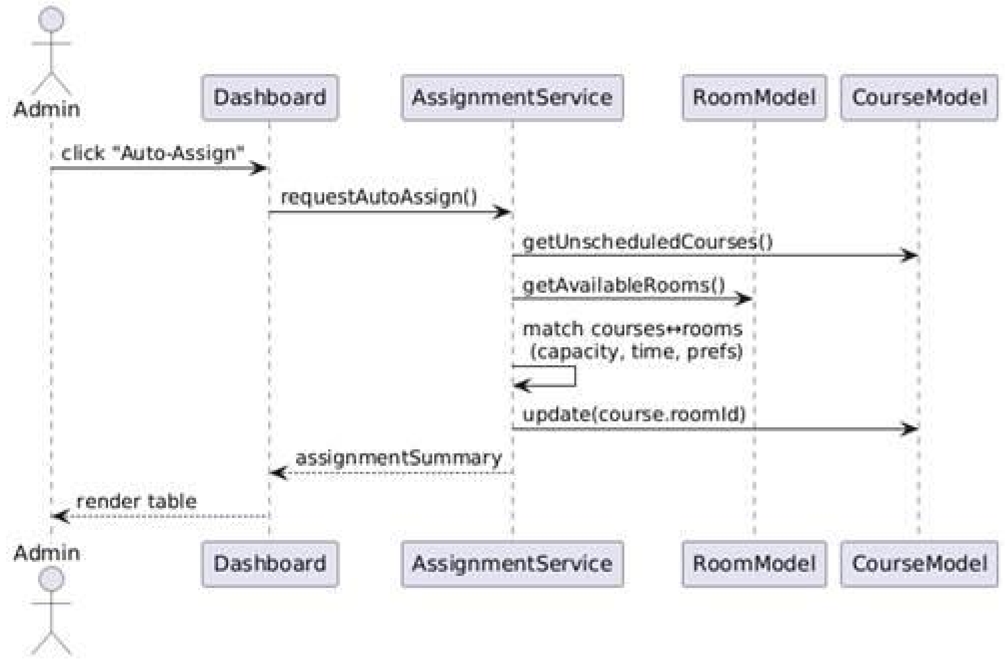
The class diagram depicts the primary domain entities. User is a base abstraction specialised by Student and Instructor. Course aggregates enrollment counts, is taught by exactly one instructor, and may be scheduled in at most one Room. Enrollment captures the many-to-many relationship between students and courses while storing the current status (ENROLLED or WAITLISTED).

Alternative designs considered a join table inside courses for enrolled IDs, but the dedicated Enrollment collection was chosen to simplify querying wait-lists and future grade storage.

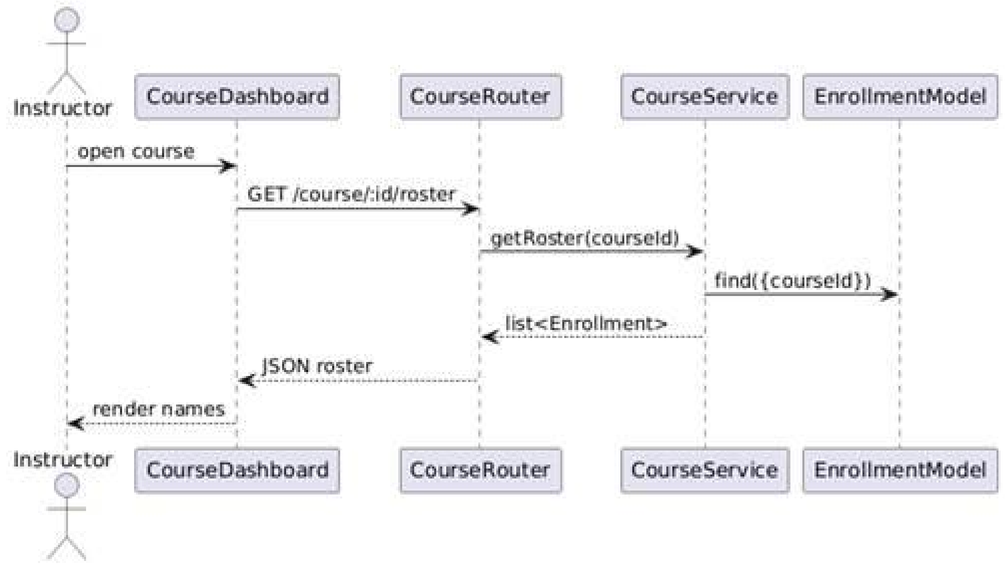
Patterns employed:
- Service Layer – isolates business logic from controllers.
- Repository (via Mongoose models) – encapsulates data-access logic.
- Factory – User registration endpoint instantiates specialised user objects based on role.

3.2 Dynamic View

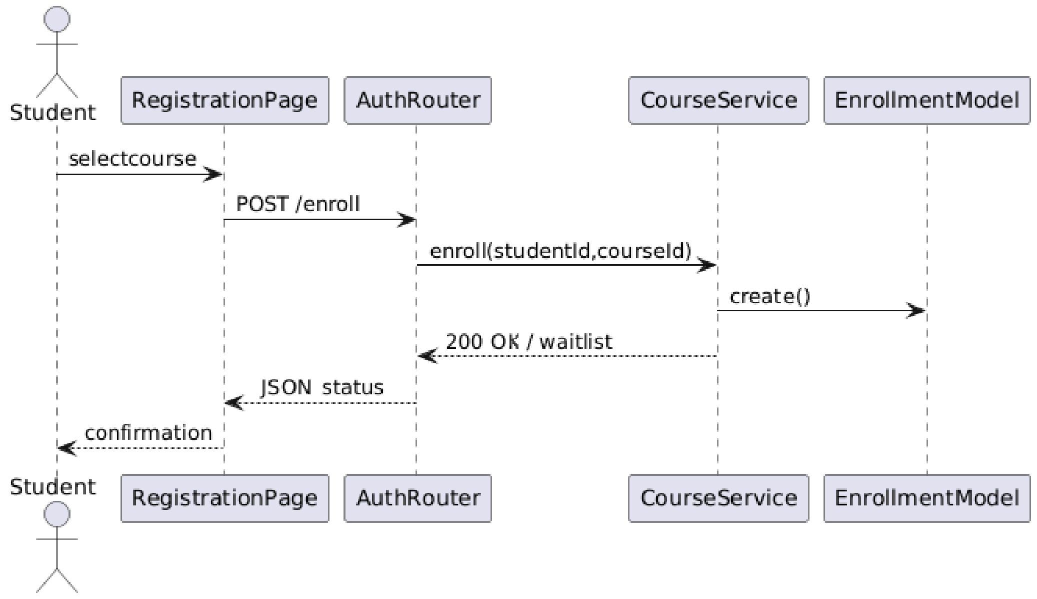
Admin:



Instructor:



Student:



The sequence diagrams illustrate three representative scenarios:
 1. Automatic Room Assignment – an admin command invokes AssignmentService, which retrieves unscheduled courses and available rooms, matches by capacity and preferences, and persists the selections.
 2. Instructor Views Roster – an instructor dashboard request traverses CourseRouter → CourseService → EnrollmentModel, returning a JSON roster that the client renders.
 3. Student Registration – the registration page submits an /enroll POST; CourseService validates conflicts before an EnrollmentModel.create() call writes a new document and returns status 200 or WAITLISTED.

4. Verification & Validation Assets

Test Plan Highlights:
Unit Tests – Test suites for services verifying edge-cases (e.g., over-capacity, duplicate enrollment).
Integration Tests – Supertest harness spins up an in-memory Mongo instance to exercise REST endpoints.
E2E Tests – Cypress scripts cover core user journeys: login, create room, create course, auto-assign, student enroll.

Sprints & Reviews:

Sprint	Goals	Outcome
1	Project setup, auth scaffolding	Registration/login working; CI on GitHub Actions
2	CRUD for Rooms & Courses	Express routes + React forms merged
3	Automatic room assignment algorithm	Initial algorithm done; edge cases logged
4	Instructor roster & student enrollment	Roster complete; enrollment pending final conflict checks