Course Selection System

1. Abstract

This project implements a Course Selection System using Java Spring Boot (back-end), MySQL (database), and vanilla HTML/CSS/JavaScript (front-end). It supports Student and Teacher roles, providing course creation, enrollment and dropping, time-conflict detection, prerequisite validation, and a per-user credit cap (≤ 21 credits). The front-end adopts a unified, responsive, and modern UI.

2. System Architecture

- Back-end: Spring Boot 3.x with layered modules: `entity`, `repository`, `service`, `controller`.
- Security: Spring Security form login; passwords stored via BCrypt.
- Database: MySQL with JPA/Hibernate ORM.
- Front-end: Vanilla HTML/CSS/JS with Fetch API for REST calls; responsive layout and consistent theming.

3. Database Design

- `users`: *id*, *username*, *password*, *role (STUDENT/TEACHER)*, *full_name*, *created_at*.
- `courses`: *id*, *code*, *name*, *description*, *credits*, *capacity*, *teacher_id*, *day_of_week*, *start_time*, *end_time*.
- `course_prerequisites`: many-to-many mapping between course and its prerequisites.
- `enrollments`: student enrollment records (unique constraint on *student_id* + *course_id*).

4. Feature Implementation

4.1 User & Security

- Registration: `POST /api/auth/register` (JSON), unique username validation, BCrypt password storage.
- Login: Spring Security form login, `/api/auth/login`.
- Current user: `GET /api/auth/me`.

4.2 Teacher Portal

- Course management: create, list teacher's own courses, delete.

- Prerequisite selection: send *prerequisiteIds* from fron-tend; map to prerequisite course set in back-end.
- Student list: `GET /api/teacher/courses/{id}/students`.

4.3 Student Portal

- Browse courses: `GET /api/student/courses`.
- Enroll/Drop: `POST /api/student/enroll/{courseId}`, `DELETE /api/student/enroll/{courseId}`.
- My enrollments: `GET /api/student/enrollments` and render a weekly schedule.

5. Business Rules & Validation

- Prerequisites: all required prerequisites must be satisfied prior to enrollment.
- Credit cap: current credits + new course credits ≤ 21; otherwise enrollment is rejected.
- Time conflicts: same day with overlapping time ranges is rejected.
- Capacity: if enrolled count reaches capacity, enrollment is rejected.
- Uniqueness: a student cannot enroll in the same course twice.

6. Exception Handling & Security

- Unified error responses: controllers return readable messages with appropriate status codes (400/401/403).
- Role-based access control: endpoints restricted by role (`/api/teacher/**`, `/api/student/**`).
- Password security: BCrypt hashing; plaintext is never stored or returned.

7. Frontend UI/UX

- Unified theme & components: cards, buttons, tables, modals, progress bar.
- Responsive layout: desktop-first with adjustments for tablets and phones.
- Interaction improvements: form validation, status toasts, conflict and limit warnings.
- Weekly schedule grid: hour-based blocks rendering enrolled courses.

8. Testing

- Unit tests: `EnrollmentServiceTest` covers credit cap and successful enroll path.
- Integration tests: manual verification through browser for enroll/drop/prerequisites/conflicts.

9. Deployment & Run

- Executable jar: `src/course-system.jar`.
- Startup script: `run/start.bat` (auto-opens `http://localhost:8080`).
- Configuration: `src/main/resources/application.properties`.

10. Data Initialization

- Default users & courses: `doc/Default_Users.txt`, `doc/Default_Courses.txt`.
- Runtime seeding: example teachers, students, and courses injected on first run, including typical prerequisite relations.

11. API Overview

Method	Endpoint	Description
POST	/api/auth/register	Register
POST	/api/auth/login	Login
GET	/api/auth/me	Current user
GET	/api/teacher/courses	List my courses (teacher)
GET	/api/teacher/all-courses	List all courses (for prerequisites)
POST	/api/teacher/courses	Create course (supports prerequisites)
DELETE	/api/teacher/courses/{id}	Delete course
GET	/api/teacher/courses/{id}/students	List enrolled students
GET	/api/student/courses	List available courses
GET	/api/student/enrollments	List my enrollments
POST	/api/student/enroll/{courseId}	Enroll
DELETE	/api/student/enroll/{courseId}	Drop

12. Limitations & Future Work

- Schedule granularity is hour-based; future work can refine time blocks and add locations/rooms.
- Prerequisite validation currently assumes enrollment history; integrating grades/semesters would improve correctness.
- Scalability & concurrency: high-concurrency enrollment may require DB locking or queuing to avoid oversubscription.
- Administration & auditing: consider adding admin role, operation logs, and approval workflows.

13. Directory Structure

- `src`: source code and resources.
- `run`: runnable scripts.
- `doc`: documentation and guides.

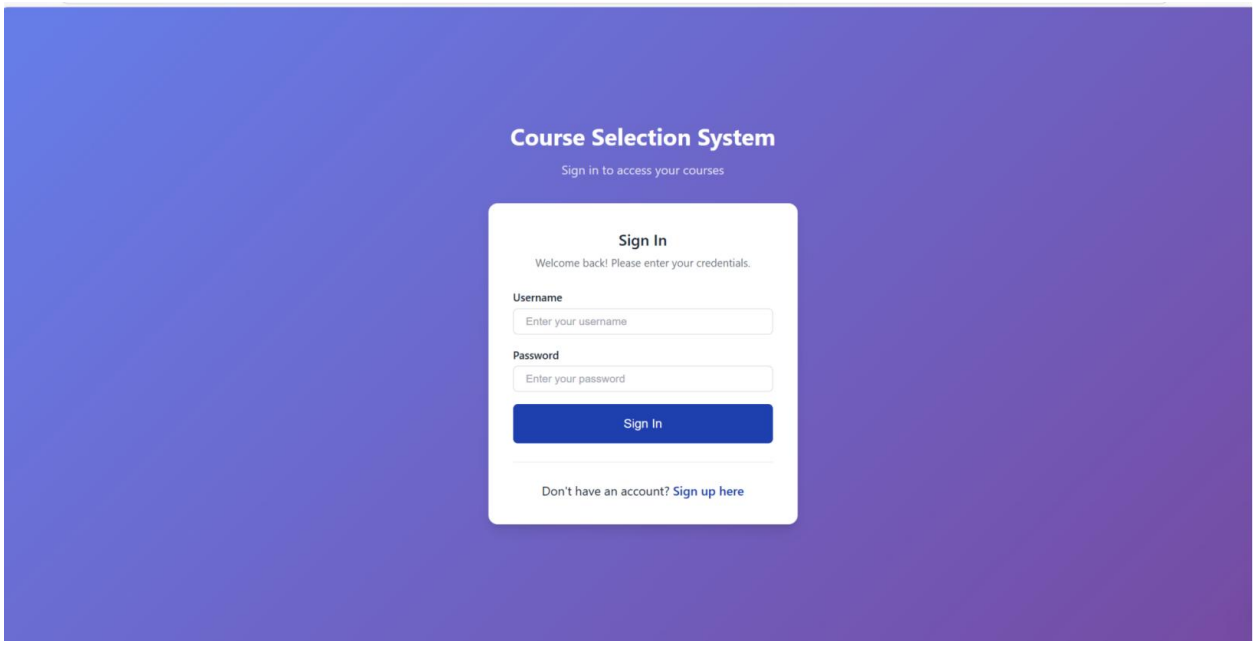
14. Usage Flow

1. Start the system and open the login page.
2. Teacher creates courses and sets prerequisites.
3. Student browses and enrolls; the system validates prerequisites, credits, and conflicts.
4. Student checks weekly schedule and drops or adjusts courses as needed.

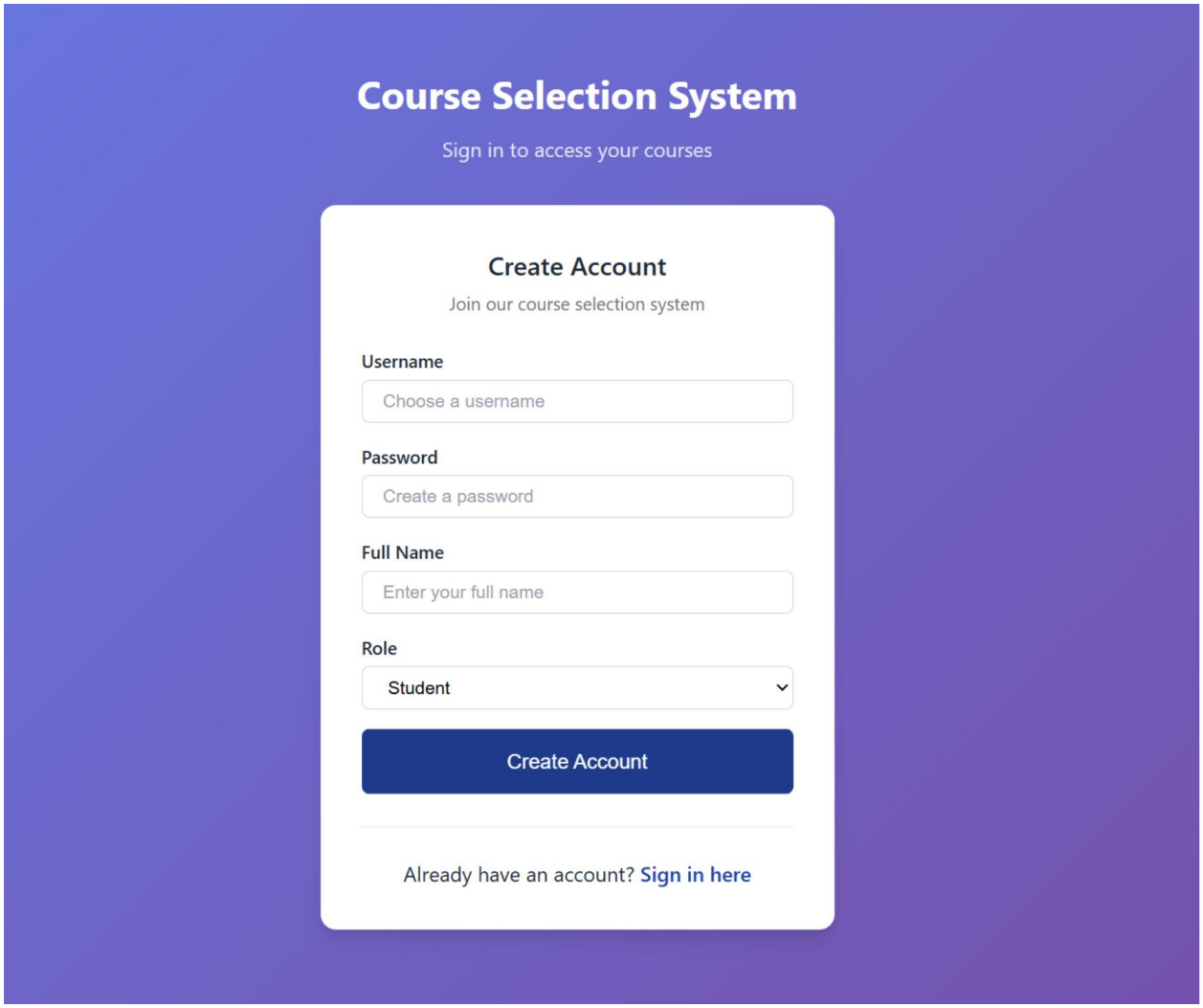
15. Screenshots (images/ directory)

Place screenshots under project root `images/` and the report will reference them using relative paths:

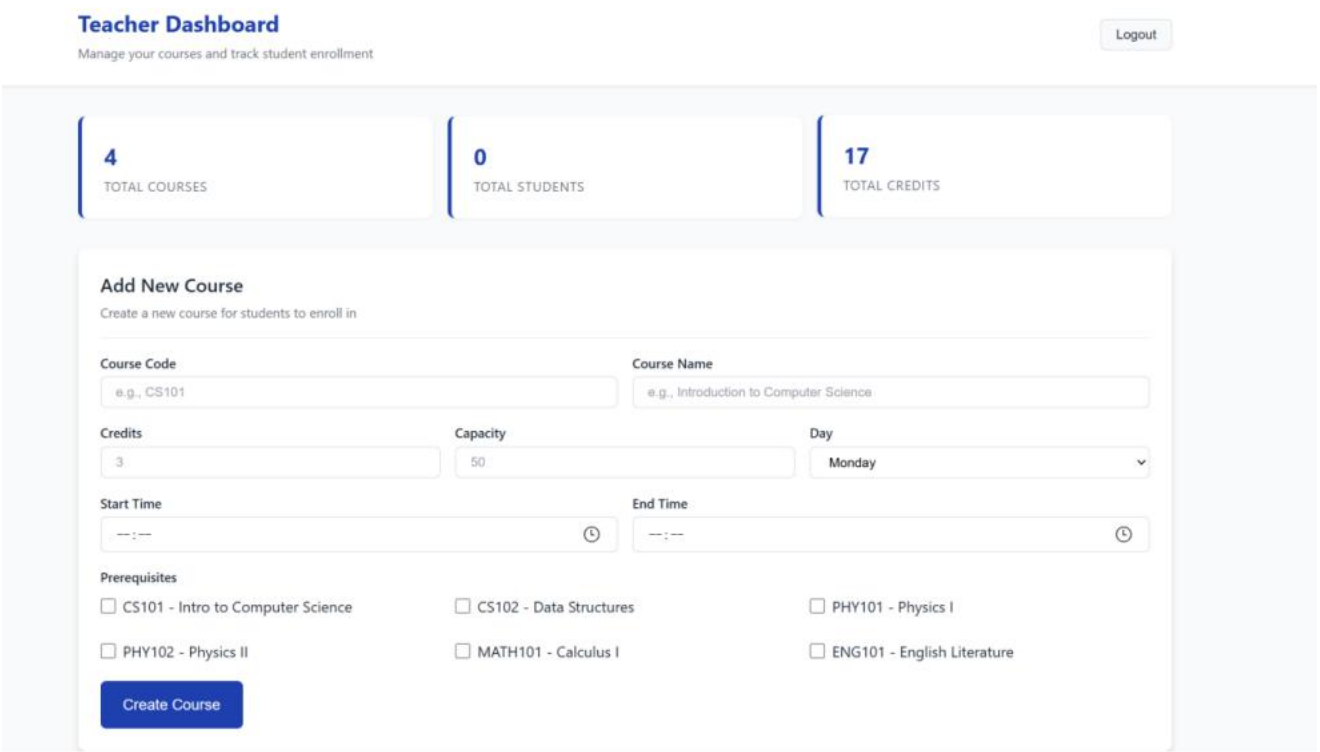
- Login page:



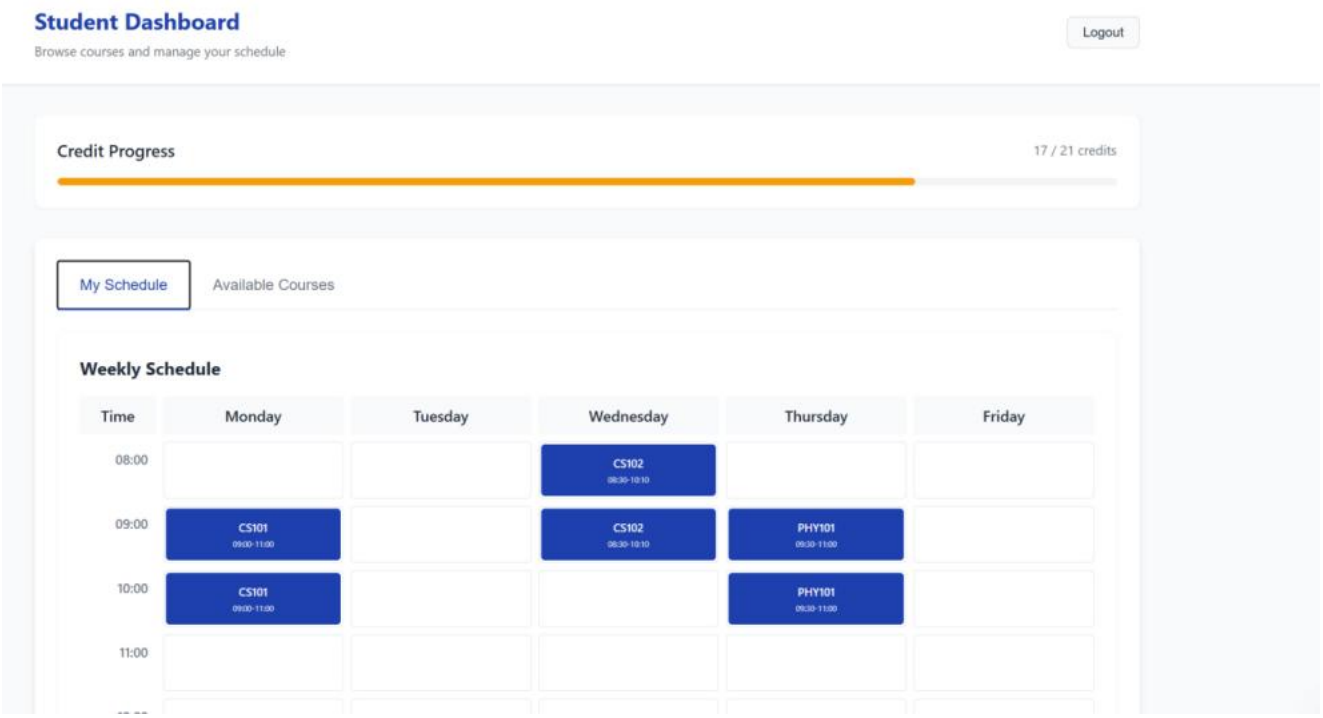
- Register page



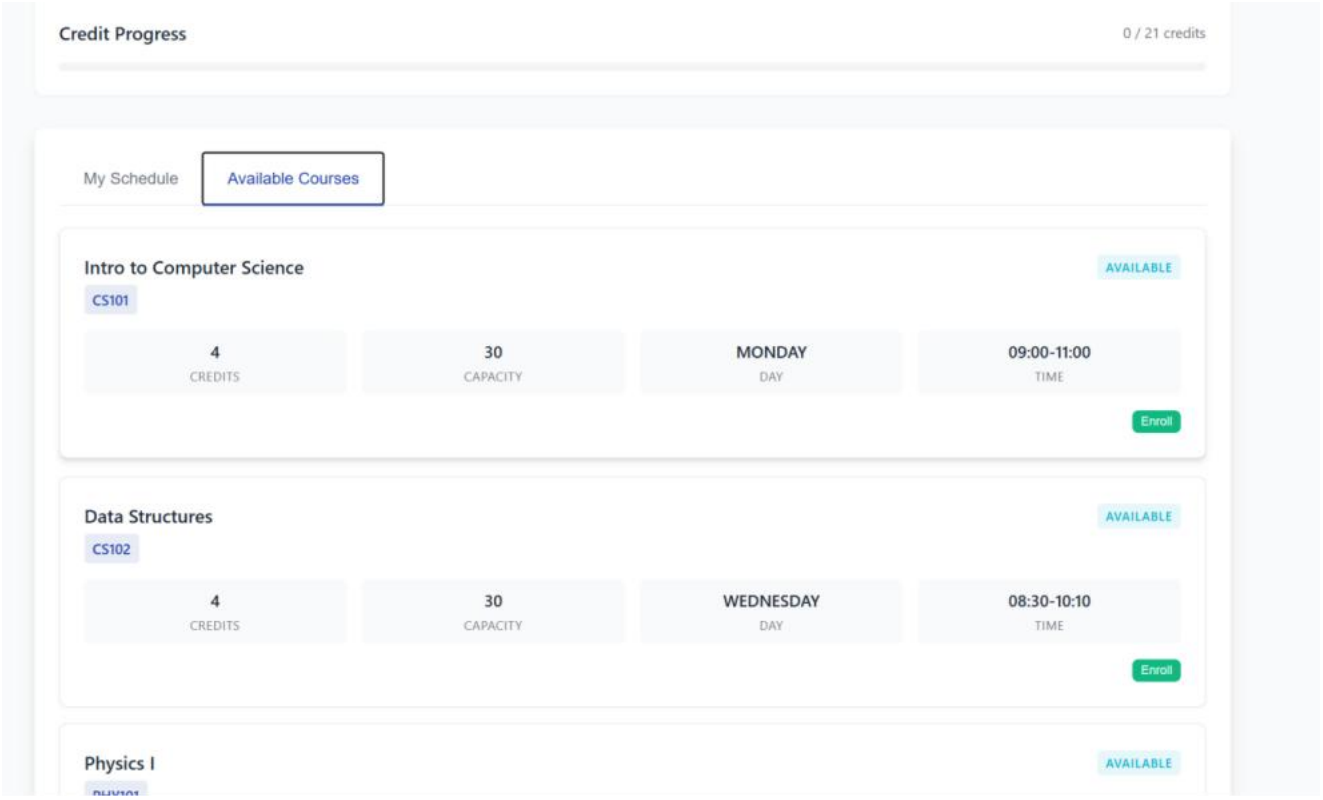
- Teacher dashboard:



- Student dashboard:



- Available courses:


Credit Progress
0 / 21 credits
My Schedule
Available Courses
Intro to Computer Science
AVAILABLE
CS101
4
CREDITS
30
CAPACITY
MONDAY
DAY
09:00-11:00
TIME
Enroll
Data Structures
AVAILABLE
CS102
4
CREDITS
30
CAPACITY
WEDNESDAY
DAY
08:30-10:10
TIME
Enroll
Physics I
AVAILABLE