Team Introduction – Course Registration System Project

- Team Members:
  Zeenat (24G-BCS036)
  Barira Fatima (24G-BCS003)
  Mariya Kamran (24G-BCS008)
- Course: Data Structures
- Instructor: *Sir Twaha Ahmed Minai*
- Project Overview:
  – A real-world simulation of a university course registration system
  – Implementing prerequisite validation, conflict checking, seat allocation, waitlists, and section balancing
  – Built entirely using core data structures learned in this course

Made with GAMMA

Domain Study: How Universities Register Students

- Students log into an online portal with existing records (ID, semester, completed courses)
- System checks multiple conditions before allowing registration:

Prerequisite validation	Seat availability in selected section
Timetable conflict detection	Waitlist management (FIFO)

- If seats exist → student is enrolled
- If full → student is placed in a waitlist automatically
- System ensures fairness & avoids manual errors

Made with GAMMA

Domain Study: Prerequisites, Waitlists, Section Capacity, Conflicts

Prerequisites

- A course must be completed before taking an advanced one
- Example: "OOP → prerequisite for DSA"
- System uses DAG traversal to check prerequisite chains

Waitlists (Queues – FIFO)

- When section capacity is full, new students go to waitlist
- First student in queue gets the next available seat

Section Capacity

- Fixed seats per section (e.g., “DSA-A: 40 seats”)

Timetable Conflict Checking

- Detects overlapping class timings
- Example: “Clash with CS200, Monday 10–11 AM”
- Conflict blocks registration

Made with GAMMA

Functional Requirements

- [ ] View all available courses and their sections
- [ ] Register only if prerequisites are satisfied
- [ ] Show error if student attempts a course without completing required prerequisites
- [ ] Check timetable for overlapping slots before registration
- [ ] Store all students, courses, prerequisites, timings, and seat data
- [ ] Prevent duplicate registration for the same course
- [ ] Add student to waitlist if section is full
- [ ] Auto-promote waitlisted student when a seat opens
- [ ] Support section balancing when one section is overloaded
- [ ] Display clear error reasons (conflict, full section, missing prereq)
- [ ] Generate reports:
  - Students enrolled in a course
  - Students waitlisted
  - Available seats in each section

Made with GAMMA

High-Level Architecture Overview

System is divided into four main modules:

- CourseGraph Module (DAG Prerequisite Management)
- Registrar Module (Registration Logic, Prerequisite & Conflict Checking)
- SectionManager Module (Seat Allocation, Waitlist, Balancing)
- Reports Module (Enrollment, Waitlist, Fail Reasons)

Each module communicates using hash maps, arrays, queues, and tree/heap structures to ensure fast lookup and efficient processing.

Made with GAMMA

Architecture: CourseGraph, Registrar, SectionManager

CourseGraph Module (DAG)

- Stores all course prerequisites in a directed acyclic graph
- Supports BFS/DFS traversal
- Validates prerequisite chains efficiently (O(V+E))

Registrar Module

- Interacts with CourseGraph for prerequisite validation
- Uses hash maps for:
  - studentID → enrolledCourses
  - courseCode → sections
- Performs timetable conflict detection using array-based weekly grids
- Controls the full registration process & failure handling

SectionManager Module

- Manages sections using hash maps for O(1) access
- Seat allocation handled using tree/heap structures for load balancing
- Maintains FIFO waitlists using Queue
- Auto-promotes students from waitlist to open seats

Made with GAMMA

Chosen Data Structures

Directed Graph / DAG

- Stores prerequisites
- Enables DFS/BFS prerequisite validation
- Complexity: O(V + E)

Hash Maps

- Very fast lookup: student records, course-section mapping

Queues (FIFO)

- Used strictly for waitlists
- Ensures fairness (first-come, first-served)

Trees / Heaps

- Used for section balancing and priority-based seat allocation

Arrays

- Table-grid representation for conflict detection

Algorithms

- DFS/BFS for prerequisite checks
- Heap operations for load balancing
- O(1) hash table access for registration records

Made with GAMMA

Project Plan: Overview, Objectives, Scope

1

Project Overview

Build a DS-based academic registration system handling prerequisites, timetable clashes, seat allocation, waitlists, and section balancing

2

Project Objectives

- Implement DAG + DFS/BFS for prerequisite validation
- Real-time timetable conflict detection
- Fast hashing for seat and registration info
- FIFO queue-based waitlist management
- Trees/heaps for load balancing
- Generate reports: seat availability, enrollments, waitlists

3

Project Scope (Included)

- Student registration & drop
- Prerequisite checking
- Conflict detection
- FIFO waitlists
- Section balancing with trees/heaps
- Reporting
- CLI-based interface

Excluded: Payments, frontend UI, instructor management, transcripts, dashboards

Made with GAMMA

Work Breakdown Structure

01

Milestone 1 — System Design & DS Implementation

- Build DAG, adjacency lists, hash maps
- CRUD operations for student/section records

02

Milestone 2 — Queue, Tree/Heap & Array Implementation

- FIFO queue for waitlists
- Min-heap for section load balancing
- 2D Timetable array with clash detection

03

Milestone 3 — Module Implementation

- CourseGraph: DFS/BFS, prerequisite chain checking
- Registrar: hashing, conflict checking, registration logic
- SectionManager: seat allocation, waitlist promotion, balancing

04

Milestone 4 — Integration

- Connect all modules
- Validate workflow end-to-end

05

Milestone 5 — Testing & Optimization

- Unit, integration, stress testing
- Complexity & memory optimization

06

Milestone 6 — Documentation & Final Submission

- Manuals, UML, reports, final presentation

Made with GAMMA

Risks & Mitigation

Technical Risks

- Circular prerequisites → solve with topological sort
- Hash collisions → use good hash functions
- Heap balancing issues → unit tests & validated operations
- Race conditions → transaction-like seat allocation

Project Management Risks

- Integration delays → start early, weekly checks
- Scope creep → stick to requirements
- Last-minute bugs → freeze features in Week 9

Logic Risks

- Incorrect prerequisite validation → thorough test matrix
- Waitlist promotion errors → strict FIFO enforcement
- Conflict detection inaccuracies → standardized time-slot grid

Made with GAMMA