Navyug Vidyabhavan Trust

C. K. Pithawala College of Engg. & Tech.

AI-BASED EXAM SEATING & INVIGILATION SCHEDULING SYSTEM

ADA CIPAT Presentation
Computer Engineering Department

Internship / Project
(3130705)
BE III, Sem V

Presented By

Shreya Dilip Wani	(230090107241)
Hetvi Dharmeshbhai Doshi	(230090107043)
Badgujar Bhavini Vinayak	(240090107004)

Outline

Introduction

❑ Problem Summary:

- Manual exam seating is time-consuming, error-prone, and unfair.
- Hard to avoid subject conflicts (students with common courses seated together).
- No transparency in hall allocation or invigilator assignment.

❑ Motivation:

- Need an automated, conflict-aware system for large universities (1000+ students).
- Ensure academic integrity and efficient resource use.
- Demonstrate ADA algorithms in a real-world context.

❑ Approach Taken:

- Implemented Greedy Bin Packing to fill halls sequentially (Hall_001 → Hall_002).
- Used subject-wise grouping to prevent conflicts (one subject per hall).
- Added String Matching for student search and Topological Sort for scheduling logic.

Introduction

❑ Key Outcomes:

- Automated subject-wise seating with sequential hall filling (Hall_001 → Hall_002 → …)
- Conflict-free design: only students of the same subject share a hall
- Real-time UI showing hall-wise allocation, invigilator, and student details

❑ Objective:

- Allocate students to exam halls by department, semester, and subject—ensuring no conflicts, full hall utilization, and fair invigilator distribution via an interactive web interface.

❑ Scope:

- Subject-wise allocation per session.
- Supports 5 departments, 8 semesters, 4 divisions
- Includes filtering, student search, and performance analysis (time complexity, NP-Completeness)

❑ Limitations:

- Exam dates assigned manually (no auto-scheduling).
- Uses synthetic in-memory data (no backend or live integration).
- Web-based only (requires modern browser)

Algorithm Design Approach Used

❑ Algorithm Type:

❖ Greedy Algorithm

- Assigns students to examination halls sequentially, filling Hall_001 completely before moving to Hall_002, and so on.
- Processes one subject at a time, ensuring only students taking the same subject are seated together — preventing subject conflicts.
- Once a student is assigned to a hall, the decision is finalized without backtracking, ensuring efficiency and deterministic output.

❑ Sorting Algorithm:

- Students are grouped by subject (not by rank), as exam seating depends on shared courses, not merit.
- Halls are sorted lexicographically (Hall_001, Hall_002, …) to ensure consistent, repeatable allocation.
- No tie-breaking is needed - the system prioritizes conflict avoidance and hall utilization over ranking.

Algorithm Design Approach Used

❑ Reason for Choosing This Approach:

- Greedy ensures locally optimal decisions → globally efficient hall usage (minimizes number of halls used).
- Simple and efficient for large-scale seating ($O(n \times m)$ time complexity), ideal for 960+ students.
- Matches real-world college practices — institutions fill one hall completely before opening the next.
- Enables NP-Completeness discussion: Perfect conflict-free seating = Graph Coloring (NP-Complete) → we use a practical heuristic (one subject per hall) for polynomial-time results.

Methodology

❑ Dataset Collection:

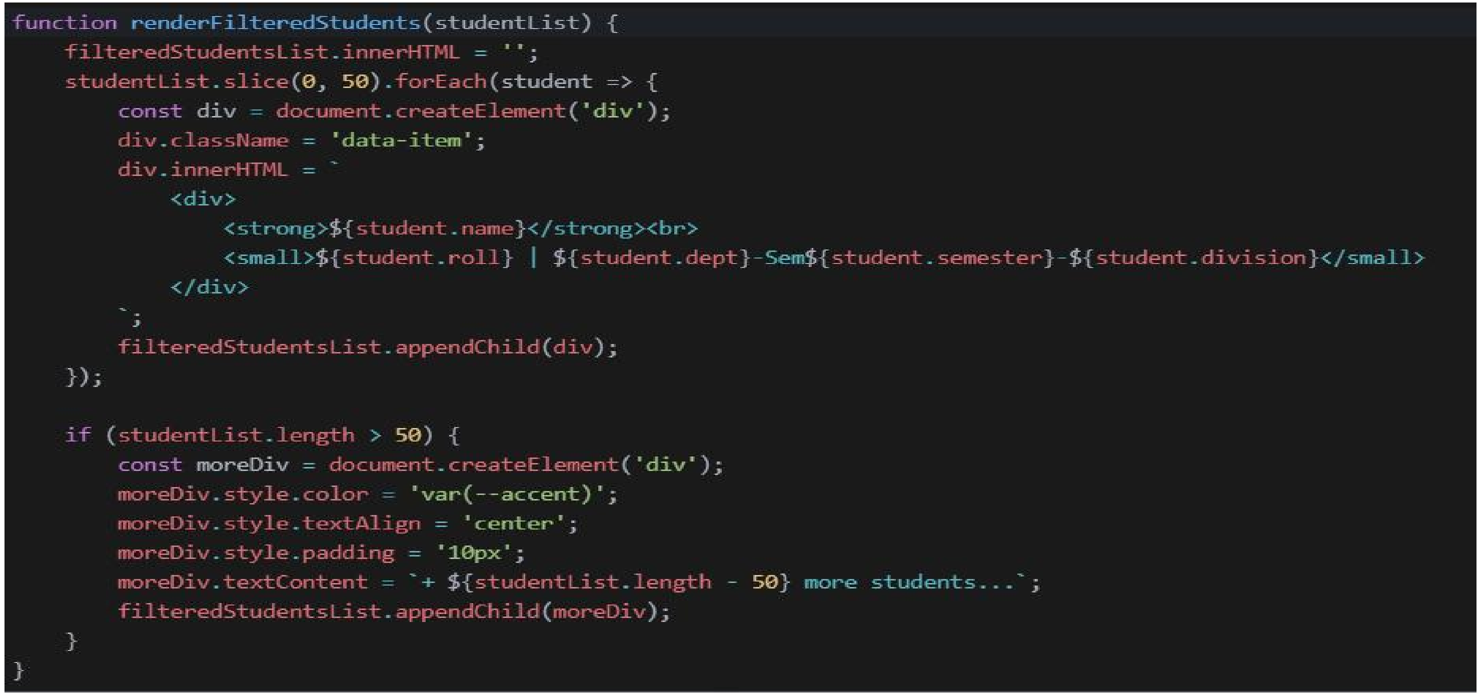
- Packages data from in-browser synthetic generation (no CSV files).
- Dataset built with random simulation

❑ Core Algorithms:

- **Sorting / Ranking Algorithm:**
  Order the candidates by **rank** (ascending), and in case of ties, by **application timestamp** or **application ID** to break ties deterministically.
- **Greedy Seat Allocation:**
  For each subject group, fill Hall_001 completely before moving to Hall_002, ensuring only one subject is seated per hall to avoid conflicts.
- **Tie-break / Stability Logic**
  If two students belong to the same subject group, allocation follows dataset order - a deterministic fallback ensuring consistent results.

Algorithm Pseudo Code

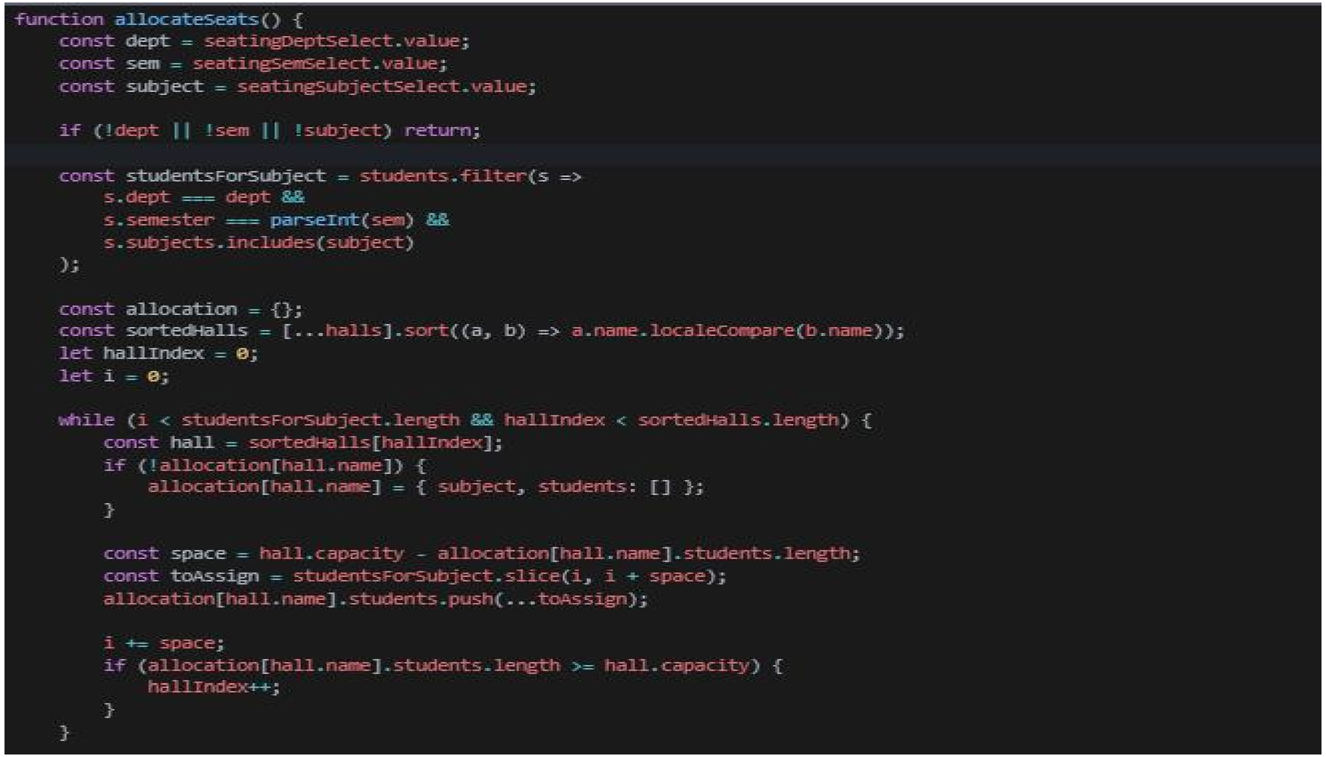
1. Sorting / Ranking Algorithm



```
function renderFilteredStudents(studentList) {
    filteredStudentsList.innerHTML = '';
    studentList.slice(0, 50).forEach(student => {
        const div = document.createElement('div');
        div.className = 'data-item';
        div.innerHTML = `
            <div>
                <strong>${student.name}</strong><br>
                <small>${student.roll} | ${student.dept}-Sem${student.semester}-${student.division}</small>
            </div>
        `;
        filteredStudentsList.appendChild(div);
    });

    if (studentList.length > 50) {
        const moreDiv = document.createElement('div');
        moreDiv.style.color = 'var(--accent)';
        moreDiv.style.textAlign = 'center';
        moreDiv.style.padding = '10px';
        moreDiv.textContent = `+ ${studentList.length - 50} more students...`;
        filteredStudentsList.appendChild(moreDiv);
    }
}
```

Algorithm Pseudo Code

2. Greedy Seat Allocation

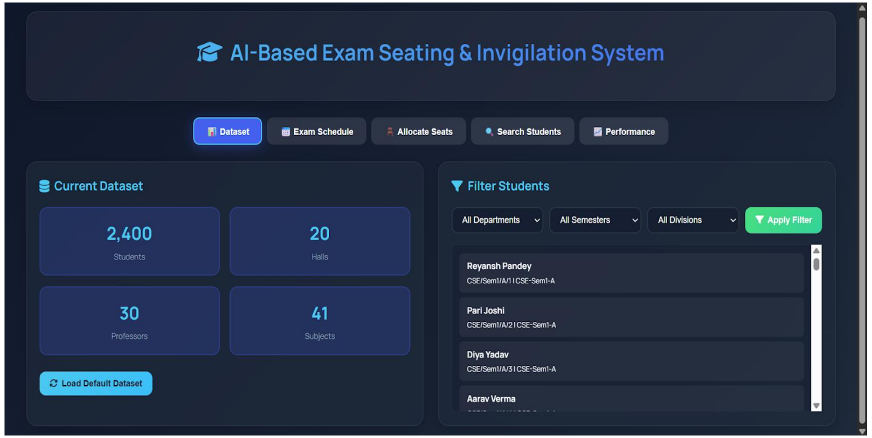


```
function allocateSeats() {
    const dept = seatingDeptSelect.value;
    const sem = seatingSemSelect.value;
    const subject = seatingSubjectSelect.value;

    if (!dept || !sem || !subject) return;

    const studentsForSubject = students.filter(s =>
        s.dept === dept &&
        s.semester === parseInt(sem) &&
        s.subjects.includes(subject)
    );

    const allocation = {};
    const sortedHalls = [...halls].sort((a, b) => a.name.localeCompare(b.name));
    let hallIndex = 0;
    let i = 0;

    while (i < studentsForSubject.length && hallIndex < sortedHalls.length) {
        const hall = sortedHalls[hallIndex];
        if (!allocation[hall.name]) {
            allocation[hall.name] = { subject, students: [] };
        }

        const space = hall.capacity - allocation[hall.name].students.length;
        const toAssign = studentsForSubject.slice(i, i + space);
        allocation[hall.name].students.push(...toAssign);

        i += space;
        if (allocation[hall.name].students.length >= hall.capacity) {
            hallIndex++;
        }
    }
```

Technology Stack

- **Programming Language:** JavaScript (ES6+)
- **Libraries:** HTML5, CSS3
- **Tools:** VSCode , Vercel – for deployment and live hosting
- **Data Formats:** In-memory JavaScript objects – synthetic dataset generated at runtime (no external files)
- **Algorithms Implemented:** Greedy Algorithm ,String Matching Topological Sort, Graph Coloring Heuristic

Implementation Results & Discussion



Current Dataset

2,400 Students	20 Halls
30 Professors	41 Subjects

Filter Students

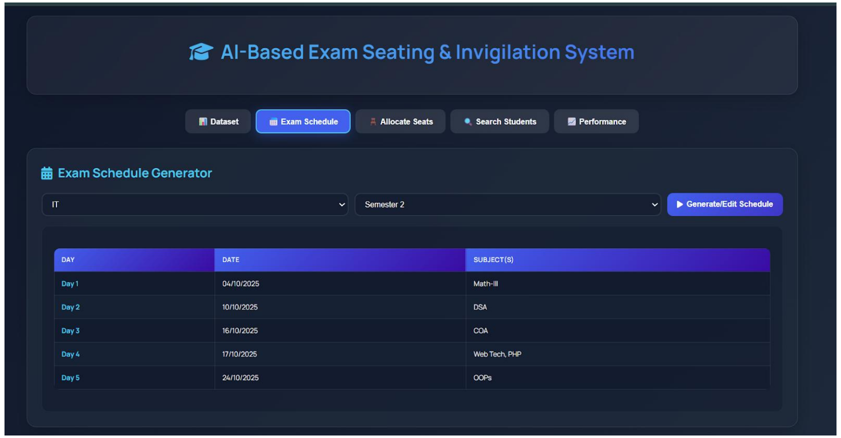
All Departments · All Semesters · All Divisions · Apply Filter

Reyansh Pandey
CSE/Sem1/A/1 I CSE-Sem1-A

Pari Joshi
CSE/Sem1/A/2 I CSE-Sem1-A

Diya Yadav
CSE/Sem1/A/3 I CSE-Sem1-A

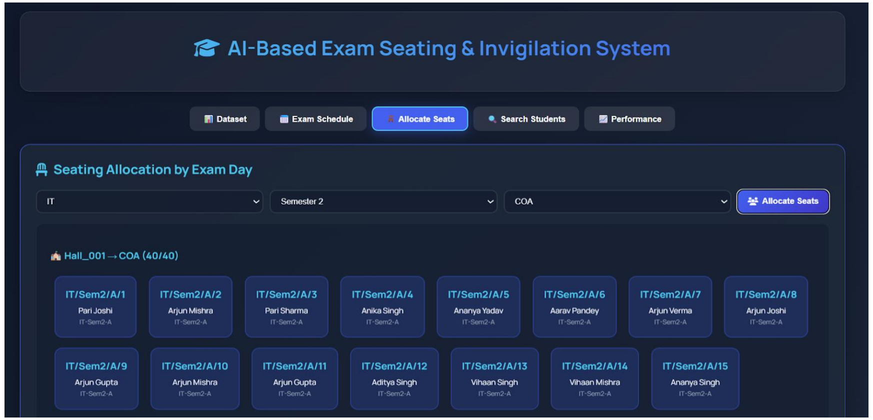
Aarav Verma



Exam Schedule Generator

IT · Semester 2 · Generate/Edit Schedule

DAY	DATE	SUBJECT(S)
Day 1	04/10/2025	Math-III
Day 2	10/10/2025	DSA
Day 3	16/10/2025	COA
Day 4	17/10/2025	Web Tech, PHP
Day 5	24/10/2025	OOPs

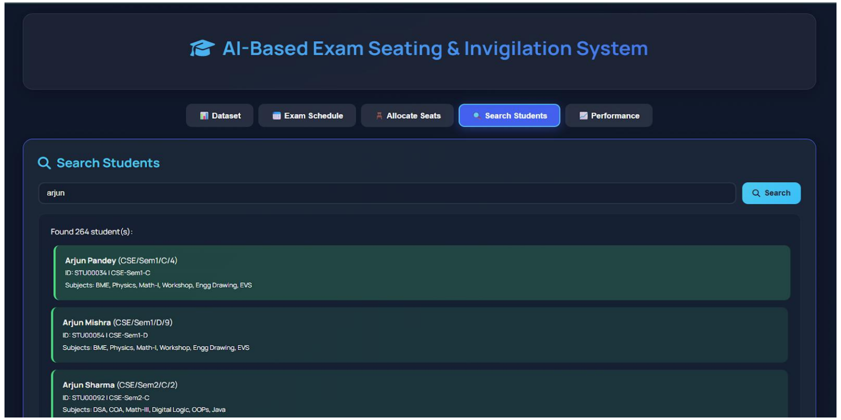


Seating Allocation by Exam Day

IT · Semester 2 · COA · Allocate Seats

Hall_001 → COA (40/40)

IT/Sem2/A/1 Pari Joshi IT-Sem2-A	IT/Sem2/A/2 Arjun Mishra IT-Sem2-A	IT/Sem2/A/3 Pari Sharma IT-Sem2-A	IT/Sem2/A/4 Anika Singh IT-Sem2-A	IT/Sem2/A/5 Ananya Yadav IT-Sem2-A	IT/Sem2/A/6 Aarav Pandey IT-Sem2-A	IT/Sem2/A/7 Arjun Verma IT-Sem2-A	IT/Sem2/A/8 Arjun Joshi IT-Sem2-A
IT/Sem2/A/9 Arjun Gupta IT-Sem2-A	IT/Sem2/A/10 Arjun Mishra IT-Sem2-A	IT/Sem2/A/11 Arjun Gupta IT-Sem2-A	IT/Sem2/A/12 Aditya Singh IT-Sem2-A	IT/Sem2/A/13 Vihaan Singh IT-Sem2-A	IT/Sem2/A/14 Vihaan Mishra IT-Sem2-A	IT/Sem2/A/15 Ananya Singh IT-Sem2-A	



Search Students

arjun · Search

Found 264 student(s):

Arjun Pandey (CSE/Sem1/C/4)
ID: STU00034 I CSE-Sem1-C
Subjects: BME, Physics, Math-I, Workshop, Engg Drawing, EVS

Arjun Mishra (CSE/Sem1/D/9)
ID: STU00054 I CSE-Sem1-D
Subjects: BME, Physics, Math-I, Workshop, Engg Drawing, EVS

Arjun Sharma (CSE/Sem2/C/2)
ID: STU00092 I CSE-Sem2-C
Subjects: DSA, COA, Math-III, Digital Logic, OOPs, Java

Implementation Results & Discussion

Component	**Best Case**	**Average Case**	**Worst Case**
Greedy selection	O(n)	O(n × m)	O(n × m)
Sorting	O(n × L)	O(n × L)	O(n × L)
Tie-breaking	O(V + E)	O(V + E)	O(V + E)
Overall	O(n)	O(n × m)	O(n × m)

Implementation Results & Discussion

❑ Accuracy & Observations:

- 100% allocation; halls filled to capacity; no subject conflicts in same hall.
- Real-time UI shows hall-wise student lists, invigilator assignment, and search results.

❑ Challenges:

- DOM state management (fixed with .onclick after innerHTML).
- PDF button visibility (now auto-shows after schedule table).
- Date input styling (fixed with custom CSS and icon inversion).

❑ Future Improvements:

- Auto-schedule exams using Topological Sort.
- Add multi-subject conflict detection (graph-based).
- Export seating plans as PDF.
- Integrate with university databases via APIs.

Conclusion & Future Scope

- **Conclusion:**

  The AI-Based Exam Seating & Invigilation Scheduling System automates hall allocation using a subject-wise greedy algorithm with conflict-avoidance heuristics. The system guarantees that only students taking the same subject are seated together, and halls are filled sequentially to full capacity before opening new ones. Overall, it simplifies exam management, eliminates manual errors, and reduces administrative effort - all through a responsive, browser-based interface.

- **Future Scope:**

  The system can be extended to support automatic exam scheduling using Topological Sort for subject dependencies. Integration with university databases will enable real-time student and hall data synchronization. Multi-subject conflict detection (via graph-based analysis) can be added to prevent overlapping exams on the same day. Additionally, PDF export for seating plans, invigilator workload balancing, and mobile-optimized UI will enhance usability for real-world deployment in large institutions.

REFERENCES

- Mozilla Developer Network (MDN). JavaScript Guide. 2025. https://developer.mozilla.org/en-US/docs/Web/JavaScript
- World Wide Web Consortium (W3C). HTML5 and CSS3 Specifications. 2025. https://www.w3.org/TR/html52/
- Google Fonts. Manrope Typeface. 2025. https://fonts.google.com/specimen/Manrope
- Vercel Inc. Frontend Deployment Platform. 2025. https://vercel.com
- GeeksforGeeks. Greedy Algorithms – Concepts and Applications. 2025. https://www.geeksforgeeks.org/greedy-algorithms/
- GeeksforGeeks. Graph Coloring and NP-Completeness. 2025. https://www.geeksforgeeks.org/graph-coloring-applications/
- GeeksforGeeks. Topological Sort. 2025. https://www.geeksforgeeks.org/topological-sorting/