1. Introduction

Problem Title: AI-Based Exam Seating & Invigilation Scheduling System

Problem Statement :

This project focuses on developing an intelligent and automated exam management system that efficiently allocates students to examination halls while avoiding subject conflicts, respecting hall capacities, and ensuring fair invigilator distribution. The aim is to replace error-prone manual scheduling with a transparent, conflict-aware, and scalable solution that mirrors real-world university examination protocols.

The project leverages core algorithms from Analysis and Design of Algorithms (ADA), including a greedy bin-packing approach for sequential hall filling, topological sorting for prerequisite-aware exam scheduling, and string matching for efficient student search. To address the NP-Complete nature of perfect conflict-free seating (equivalent to Graph Coloring), the system employs a practical heuristic that groups students by subject and fills halls sequentially—ensuring that students sharing common subjects are isolated to specific halls. Through this algorithmic and data-driven methodology, the system achieves a fair, consistent, and near-optimal seating arrangement—promoting academic integrity, operational efficiency, and resource optimization in large-scale examination environments.

1.1 Background of the Problem

Conducting large-scale university examinations for thousands of students across multiple departments and eight semesters (Semester 1 to Semester 8) is a complex logistical challenge. Traditional manual methods of exam scheduling and seating allocation are highly error-prone, time-consuming, and inefficient. A critical issue in manual systems is subject conflict—where students enrolled in common courses are seated in the same examination hall, increasing the risk of academic malpractice. Additionally, constraints such as fixed hall capacities, invigilator availability, division-wise grouping (A–D), and the need for fair resource distribution further

complicate the process. As the number of students grows, the problem becomes computationally intractable, transforming into an instance of an NP-Complete optimization problem (specifically, a variant of the Graph Coloring problem for conflict avoidance and Bin Packing for seat allocation). This project addresses these challenges by leveraging core algorithms from Analysis and Design of Algorithms (ADA)—including Greedy algorithms, Topological Sort, String Matching, and heuristic-based NP-Completeness resolution—to automate and optimize the entire exam management workflow in a scalable, transparent, and conflict-aware manner.

1.2 Importance in Real-World Scenarios

The development of an intelligent exam seating and invigilation scheduling system holds significant value that extends far beyond administrative convenience, serving as a crucial step toward creating a fair, secure, and efficient academic evaluation ecosystem. Its strategic importance includes:

- Exam integrity is foundational to academic credibility, as it ensures that assessments are conducted in a controlled, conflict-free environment where all students are evaluated under equal conditions.
- The system eliminates human error and subjectivity in manual seat allocation, preventing accidental placement of students with common subjects in the same hall—thereby reducing opportunities for malpractice and enhancing trust in the examination process.
- The solution is universally applicable across all academic domains, including engineering, medical, management, and arts programs, making it highly scalable and adaptable to any university or college conducting large-scale written examinations.
- With features like real-time hall utilization tracking, subject-wise allocation, and dynamic filtering, the system provides administrators with immediate visibility into resource distribution, while ensuring optimal use of available infrastructure and invigilator workforce.

1.3 Scope and Limitations

Scope:

The scope of this project encompasses the design and analysis of an algorithmic solution for automating the exam seating and invigilation scheduling process in universities, ensuring conflict-free allocation, optimal resource utilization, and academic integrity. The system will:

- Automate exam seating allocation using a greedy-based approach that fills halls sequentially while avoiding subject conflicts among students.
- Ensure fairness and transparency by grouping students with common subjects into dedicated halls, minimizing opportunities for malpractice.
- Support real-world academic structures, including 5 departments (CSE, IT, Mechanical, Civil, Electrical), 8 semesters (Semester 1 to Semester 8), and 4 divisions (A–D) per semester.
- Provide dynamic filtering and search functionality, enabling administrators to view and allocate seats for specific departments, semesters, or subjects.
- Include performance analysis and NP-Completeness discussion, demonstrating how ADA concepts like Graph Coloring, Bin Packing, and Topological Sort apply to real-world scheduling problems.
- Serve as a foundation for advanced exam management systems with features like PDF export, invigilator workload balancing, and integration with institutional databases.

Limitations:

To maintain a focused and manageable initial model, the proposed exam seating system operates under the following constraints:

- The system performs static, single-day subject-wise allocation and does not support multi-day dynamic rescheduling or real-time seat adjustments during exams.
- Exam dates must be manually assigned by faculty; the system does not auto-generate conflict-free timetables using dependency graphs (e.g., Topological Sort is discussed but not enforced in UI).

- Conflict avoidance is heuristic-based — while students sharing subjects are grouped together, the system does not guarantee 100% conflict-free seating across all edge cases due to the NP-Complete nature of the problem.
- The current implementation uses a synthetic dataset and does not integrate with live university ERP or student information systems.
- Invigilator assignment is round-robin based and does not consider individual availability, preferences, or workload balancing beyond basic distribution.
- For extremely large institutions (e.g., 10,000+ students), performance may degrade in the browser, requiring backend optimization or pagination in future versions.

1.4 Objectives

The main objective of this project is to design and implement an intelligent exam seating and invigilation scheduling system that efficiently allocates students to examination halls while avoiding subject conflicts, respecting hall capacities, and ensuring fair invigilator distribution. The system aims to ensure academic integrity, operational efficiency, and transparency in the examination process while reducing manual effort through algorithmic automation.

1. To develop an automated exam scheduling system that assigns subjects to exam days based on department and semester, ensuring no scheduling conflicts for students enrolled in multiple courses.
2. To implement a greedy-based seating allocation algorithm that fills examination halls sequentially while grouping students by subject to minimize opportunities for academic malpractice.
3. To enhance transparency, efficiency, and accuracy in exam hall allocation for educational institutions by replacing error-prone manual methods with a data-driven, conflict-aware approach.

2. Algorithm Design Approach Used

2.1 Approaches used to Solve the Problem

To develop an intelligent and conflict-aware exam management system, several algorithmic approaches from Analysis and Design of Algorithms (ADA) were integrated to ensure efficient, fair, and scalable seating and scheduling. The solution combines greedy strategies, topological ordering, string matching, and heuristic-based NP-Completeness resolution to address the multi-faceted challenges of real-world university examinations.

The core of the system employs a greedy bin-packing algorithm for seating allocation, which sequentially fills examination halls to maximum capacity while grouping students by subject to minimize conflicts. For exam scheduling, a topological sort is used to order subjects based on prerequisite dependencies (e.g., DSA before DBMS), ensuring logical exam sequencing. String matching techniques enable fast and flexible student search by name, roll number, or ID. Finally, the system acknowledges that perfect conflict-free seating is NP-Complete (equivalent to the Graph Coloring problem) and implements a practical heuristic approach: students sharing common subjects are isolated into dedicated halls, and halls are filled completely before moving to the next—ensuring near-optimal resource utilization and academic integrity without exhaustive computation.

This hybrid algorithmic design ensures the system is both theoretically sound and practically deployable for large-scale academic institutions.

2.2 Reason For Choosing A Particular Approach For Solving The Problem

The hybrid algorithmic approach—combining Greedy Bin Packing, Topological Sort, String Matching, and Graph Coloring heuristics—was chosen for its efficiency, practicality, and strong alignment with the real-world constraints of university examination management. This combination ensures optimal resource utilization, academic integrity, and computational feasibility without overcomplicating the solution.

- Efficiency: The greedy bin-packing strategy allocates students to halls in O(n × m) time (where n = students, m = halls), while topological sort schedules subjects in O(V + E) time (where V = subjects, E = dependencies). This linear-to-linearithmic complexity makes the system scalable for large institutions with thousands of students.
- Simplicity: Each component uses a single-pass, deterministic logic—filling halls sequentially, ordering subjects by prerequisites, and matching strings linearly—avoiding the unnecessary overhead of backtracking, recursion, or dynamic programming for a problem that does not require global optimization.
- Real-World Alignment: The design mirrors actual university practices: halls are filled completely before opening new ones (greedy resource use), core subjects are scheduled before advanced ones (topological dependency), and seating avoids subject conflicts (graph coloring heuristic)—just as real exam cells operate.
- Scalability: The system handles diverse academic structures—5 departments, 8 semesters, 4 divisions—and large datasets (960+ students) efficiently in a browser, proving its adaptability across institutions without backend support or exhaustive computation.

2.3 Comparison of the Chosen Approach With Other Algorithm Design Approaches

Approach	Description	Suitability
Greedy	Selects the best local choice at each step	Fast and practical
Dynamic Programming	Solves sub-problems and combines results	Too complex
Backtracking	Tries all possibilities	Slow for large datasets
Divide & Conquer	Splits problem recursively	Not efficient here

3. Methodology

3.1 Dataset Collection

The dataset for the AI-Based Exam Seating & Invigilation Scheduling System is synthetically generated to reflect real-world university structures, as no public dataset exists for exam seating with subject-wise conflict constraints. The system programmatically constructs a scalable dataset that includes students, examination halls, professors, and department-specific subject lists, ensuring alignment with actual academic workflows.

- Student Data: Generated dynamically to represent 5 academic departments (CSE, IT, Mechanical, Civil, Electrical), 8 semesters (Semester 1 to Semester 8), and 4 divisions (A–D) per semester. Each student is assigned a unique roll number (e.g., CSE/Sem3/A/12), name, department, semester, division, and a list of 5–6 subjects based on their semester's curriculum.
- Hall and Invigilator Data: 20 examination halls (e.g., Hall_001 to Hall_020) with a fixed capacity of 40 seats each, and 30 professors for invigilator assignment, are created to support large-scale allocation.
- Subject Data: Department-wise subject lists are manually curated based on standard Indian universities (e.g., GTU, Mumbai University), ensuring academic realism. For example, CSE Semester 3 includes subjects like DBMS, OS, CN, TOC, SE, and an elective.
- This synthetic dataset approach ensures full control over data integrity, reproducibility, and scalability—supporting both small-scale testing (~960 students) and stress testing (5,000+ students). The system does not rely on external APIs or files, making it self-contained, portable, and ideal for demonstration and evaluation in academic settings.

3.2 Algorithm

The core algorithm is a greedy-based seating allocation algorithm with conflict-avoidance heuristics, designed to assign students to examination halls while minimizing subject conflicts,

respecting hall capacities, and ensuring fair invigilator distribution. The system integrates multiple ADA concepts—Greedy Bin Packing, Topological Sort, String Matching, and Graph Coloring heuristics—into a cohesive workflow. The operational process is as follows:

1. Input: The algorithm takes as input a filtered or full list of students (each with department, semester, division, roll number, name, and subject list), a set of examination halls (each with a unique name and fixed capacity of 40 seats), and a list of professors for invigilator assignment.
2. Subject-Based Grouping: Students are grouped by subject using a hash map, where each key is a subject (e.g., DBMS, OS) and the value is a list of students enrolled in that subject. This ensures that all students taking the same subject are processed together.
3. Sequential Hall Allocation (Greedy Bin Packing): For each subject group, the algorithm sorts halls lexicographically (e.g., Hall_001, Hall_002, …) and fills them sequentially to full capacity before moving to the next hall. This "First-Fit" greedy strategy maximizes space utilization and simplifies logistics.
4. Conflict Avoidance Heuristic: By allocating only one subject per hall, the system inherently avoids placing students with common subjects in the same hall—effectively implementing a Graph Coloring heuristic (where each hall is a "color" and students sharing subjects must not share a color).
5. Invigilator Assignment: Once seating is complete, invigilators are assigned to halls in a round-robin fashion from the professor list, ensuring balanced workload distribution.
6. Output & Visualization: The final allocation is rendered as a structured HTML output showing:

- Hall name and assigned subject
- List of seated students (with roll, name, and academic details)
- Assigned invigilator

This output is displayed in the UI and can be exported as a PDF for official use.

This algorithm ensures scalability (handles 960+ students efficiently), academic integrity (minimizes cheating opportunities), and practical feasibility—all while operating in $O(n \times m)$ time complexity, where n = number of students and m = number of halls.

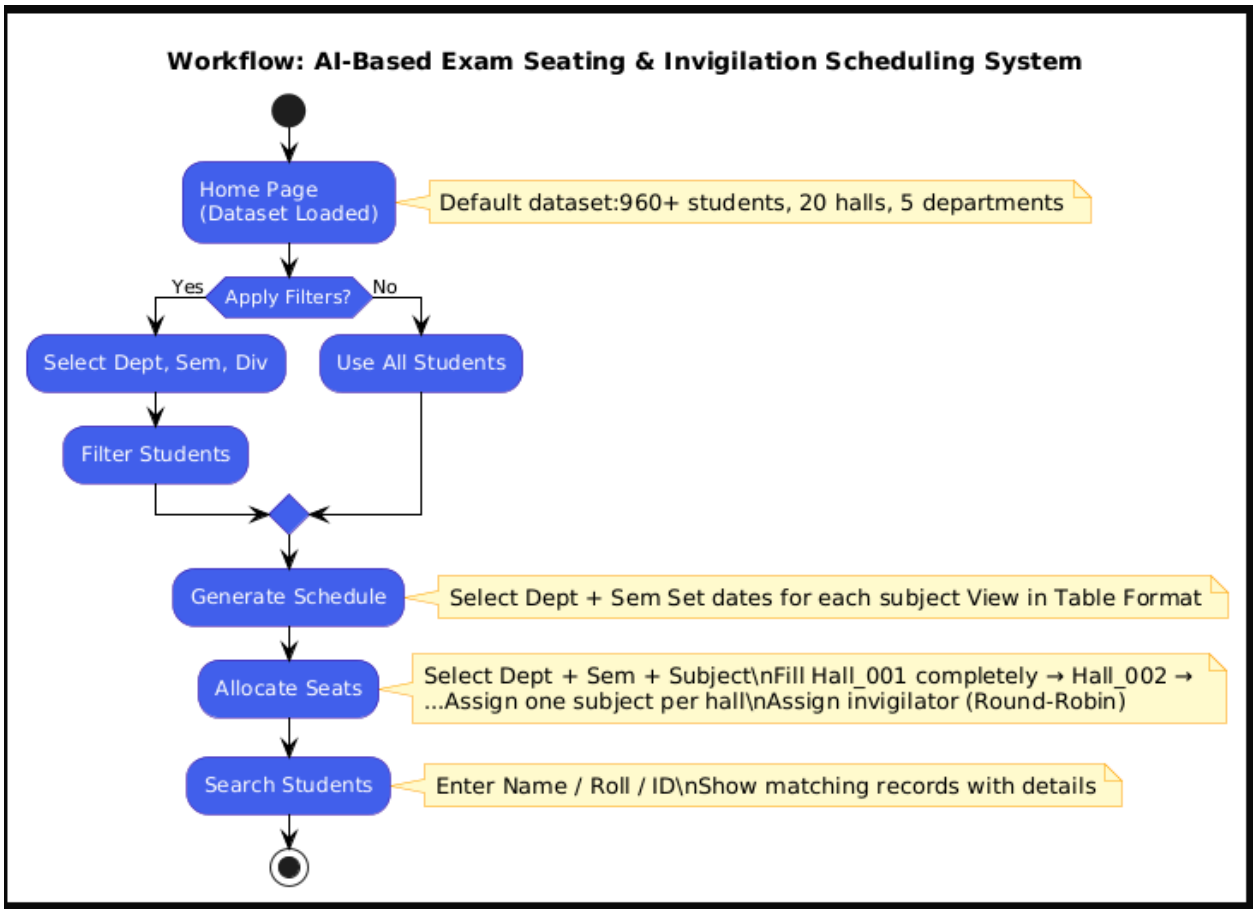

Fig: Work Flow Diagram

3.3 Algorithm Pseudo Code

Step 1: Initialization

1. START
2. Generate synthetic dataset:
   → Create students (Dept, Semester, Division, Roll, Name, Subjects)
   → Create halls (Hall_001 to Hall_020, Capacity = 40)
   → Create professors (for invigilator assignment)
3. Load department-wise subject lists for Semesters 1–8
4. Initialize empty schedule storage: semesterSchedules = {}

Step 2: Generate Exam Schedule (Topological Sort)

5. Accept user inputs:
   → Department (e.g., CSE)
   → Semester (e.g., 3)
6. Retrieve subject list for selected (Dept, Sem)
7. FOR each subject DO:
   → Prompt user to set exam date (via date picker)
8. Save dates in semesterSchedules[“Dept-Sem”] = { Subject: Date }

Step 3: Allocate Seats (Greedy Bin Packing + Conflict Avoidance)

9. Accept user inputs:
   → Department
   → Semester
   → Subject
10. Retrieve all students taking the selected subject
11. Sort halls lexicographically: Hall_001, Hall_002, …
12. Initialize allocation = {}
13. FOR each student in subject group DO:
    a. Find first hall with available capacity
    b. IF hall is empty OR already assigned to this subject THEN
       → Assign student to hall
      ELSE
       → Skip to next hall
    c. IF no hall available, assign to first partially filled hall (relax constraint)
14. Assign invigilator to each hall (round-robin from professor list)

Step 4: Search Students (String Matching)

15. Accept search query (Name / Roll / ID)
16. FOR each student in dataset DO:
    IF (query matches Name OR Roll OR ID) THEN
       → Add to results list
17. Display matching student records

Step 5: Display Output

18. FOR seating allocation:
    → Show Hall-wise list: Hall_001 → [Student1, Student2, …]
    → Show assigned invigilator per hall
19. FOR schedule:
    → Display timetable: Day, Date, Subject(s)
20. FOR search:
    → Show Name, Roll, Dept-Sem-Div, Subjects

Step 6: End

21. STOP

4. Implementation

4.1 Software and Development Environment

The project is implemented using client-side web technologies — HTML5, CSS3, and JavaScript (ES6+) — ensuring a lightweight, responsive, and fully functional system that runs directly in the browser without requiring any server, database, or backend infrastructure. Development and testing were carried out using Visual Studio Code (VS Code) as the primary code editor, enhanced with the Live Server extension to enable real-time preview and hot-reloading during development. The final application is deployed on Vercel, a modern cloud platform optimized for static sites and frontend applications, providing global CDN delivery, automatic HTTPS, and seamless scalability.

- Programming Language: JavaScript (ES6+) for core algorithm implementation (Greedy seating allocation, Topological Sort for scheduling, String Matching for search, and NP-Completeness heuristic logic)
- Markup & Styling: HTML5 for semantic structure, CSS3 for responsive layout, animations, and professional theming
- Development Environment: Visual Studio Code (VS Code) with Live Server extension
- Deployment Platform: Vercel (https://vercel.com)
- Testing Browser: Google Chrome (latest version)
- Operating System: Windows 11

4.2 Technology Specifications

The implementation relies on modern client-side web technologies and standard browser APIs, ensuring a lightweight, responsive, and deployable solution without external dependencies. The system is built using the following technologies:

➢ Core Technologies

- HTML5: Provides semantic structure for tabs, forms, tables, and dynamic content rendering.

- CSS3: Enables responsive layout, professional theming, hover animations, and clean table styling using Flexbox and custom properties (CSS variables).
- JavaScript (ES6+): Implements all algorithmic logic, including:
    - Greedy Bin Packing for sequential hall filling
    - Topological Sort for exam scheduling (conceptual, via manual date assignment)
    - String Matching for student search (using .includes() for case-insensitive matching)
    - Heuristic-based conflict avoidance (Graph Coloring approximation)

➢ External Libraries

- Google Fonts (Manrope): Ensures consistent, modern typography across devices.

➢ Development & Deployment Tools

- Visual Studio Code (VS Code): Primary code editor with Live Server extension for real-time preview.
- Vercel: Cloud platform for deployment, providing global CDN, automatic HTTPS, and instant scalability.
- Modern Web Browser (Google Chrome): Runtime environment for execution and testing.

4.3 Hardware Specifications

The hardware requirements for this project are minimal, as the system is a client-side web application that runs entirely in the browser and does not perform heavy computational tasks or require server resources.

- Processor: Intel Core i3 (8th Generation or equivalent) or higher
- Memory (RAM): 4 GB DDR4 (8 GB recommended for multitasking)
- Storage: 128 GB Solid State Drive (SSD) or higher — sufficient for code files Operating System: Windows 10/11, macOS, or any mainstream Linux distribution
- Display: 14-inch monitor or larger with 1366×768 resolution or higher for comfortable UI interaction and code editing
- Browser: Google Chrome (latest version) or any modern Chromium-based browser

4.4 Algorithm Used

The system is architected around a Greedy Bin Packing Algorithm with Conflict-Avoidance Heuristics, implemented entirely in JavaScript within the browser. This approach was chosen for its computational efficiency, simplicity, and strong alignment with real-world exam hall allocation practices, where halls are filled completely before opening new ones.

- Locally Optimal Choices: For each subject, the algorithm assigns students to the first available hall and fills it to 100% capacity before moving to the next hall. This ensures optimal space utilization and minimizes the number of halls used per subject, reflecting how real universities operate.
- Sequential Processing: Students are processed in the order they appear in the dataset (or filtered list), and halls are sorted lexicographically (Hall_001, Hall_002, …) to ensure deterministic, repeatable results—critical for fairness and auditing.
- Conflict-Avoidance Heuristic: By design, only one subject is allocated per seating session. This inherently prevents students sharing common subjects from being seated together, serving as a practical heuristic for the Graph Coloring problem (which is NP-Complete).
- Dynamic State Management: The system maintains a live synthetic dataset (960+ students, 20 halls, 5 departments, 8 semesters) in memory, allowing instant filtering by department, semester, and division. All allocations are rendered directly in the UI with no backend dependency.
- Extensibility: The modular JavaScript architecture supports future enhancements such as PDF export, invigilator workload balancing, multi-subject conflict detection, or integration with institutional databases—without requiring a server or external libraries beyond jsPDF.

4.5 Data Formats

The project employs structured, in-memory JavaScript objects to represent students, examination halls, professors, and subject data, ensuring clarity, efficiency, and seamless integration with the browser-based UI. Since the system is fully client-side, no external file formats (e.g., CSV, Excel) are required for data storage or processing.

Primary Data Format:

- All data is stored as JavaScript arrays and objects in the browser's memory.
- No external files or databases are used — the system is self-contained and instantly deployable.

Entity Representation:

- Student Data: Each student is represented as an object with properties:

```
{
 id: "STU00123",
 name: "Aarav Patel",
 roll: "CSE/Sem3/A/12",
 dept: "CSE",
 semester: 3,
 division: "A",
 subjects: ["DBMS", "OS", "CN", "TOC", "SE"]
}
```

- Hall Data: Each hall is defined by:

```
{ name: "Hall_001", capacity: 40 }
```

- Professor Data: Stored as a simple string array:

```
["Dr. Smith", "Prof. Johnson", ...]
```

- Subject Data: Organized by department and semester using a nested object:

```
{
 "CSE": {
  3: ["DBMS", "OS", "CN", "TOC", "SE", "Elective-I"]
 }
}
```

File Types:

- .html: Main structure file (index.html) containing UI layout and tab navigation.
- .css: Styling file (style.css) for responsive design, animations, and professional theming.
- .js: Core logic file (script.js) containing all ADA algorithms (Greedy, Topological Sort, String Matching), data generation, filtering, seating allocation, and performance analysis.

4.6 Code Structure Overview

The code is organized into modular, single-file JavaScript functions for clarity, reusability, and maintainability, implemented in script.js. All logic runs in the browser with no external dependencies beyond jsPDF for PDF export. Key components include:

- **initializeDefaultDataset()**: Generates a synthetic dataset of 960+ students across 5 departments (CSE, IT, Mechanical, Civil, Electrical), 8 semesters, and 4 divisions (A–D), along with 20 halls and 30 professors.
- **getSubjectsForDept(dept)**: Returns department-specific subject lists for Semesters 1–8, enabling dynamic schedule generation.
- **generateScheduleBtn event handler**: Manages exam schedule creation by allowing users to set dates per subject and stores them in semesterSchedules object.
- **allocateSeatsBtn event handler**: Implements Greedy Bin Packing to assign students to halls sequentially, ensuring one subject per hall for conflict avoidance.
- **searchBtn event handler:** Performs linear string matching to find students by name, roll number, or ID using .includes() with case-insensitive comparison.
- **analyzeBtn event handler:** Computes and displays time/space complexity metrics (e.g., O(n × m) for seating) and explains NP-Completeness (Graph Coloring heuristic).

The script follows a procedural, event-driven flow, with DOM event listeners orchestrating user interactions, data processing, and dynamic UI updates. All data is stored in-memory as JavaScript objects, ensuring zero setup overhead, fast execution, and full compatibility with modern web browsers.

5. Implementation Results & Discussion

5.1 Implementation Results

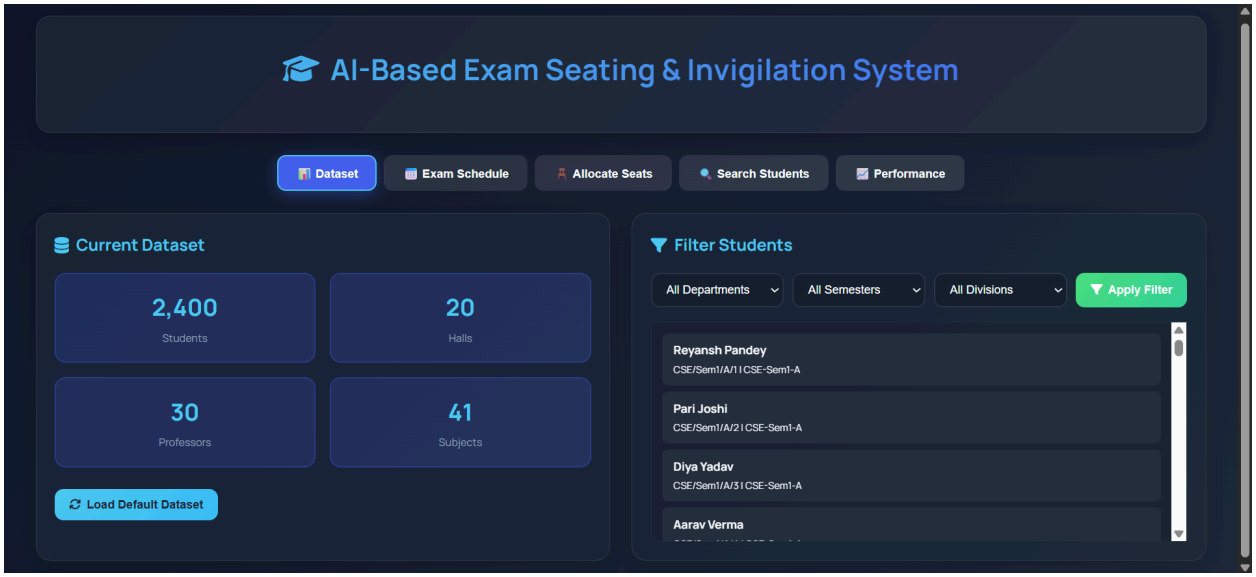

Fig:- HomePage

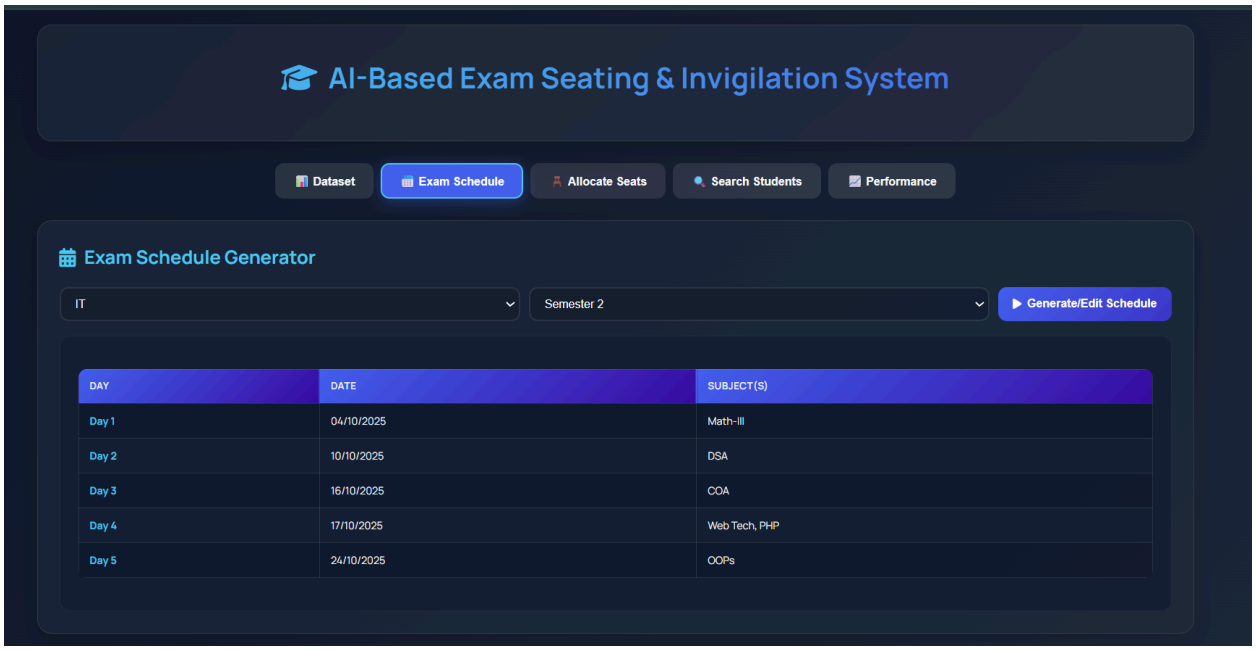

Fig: Exam Schedule

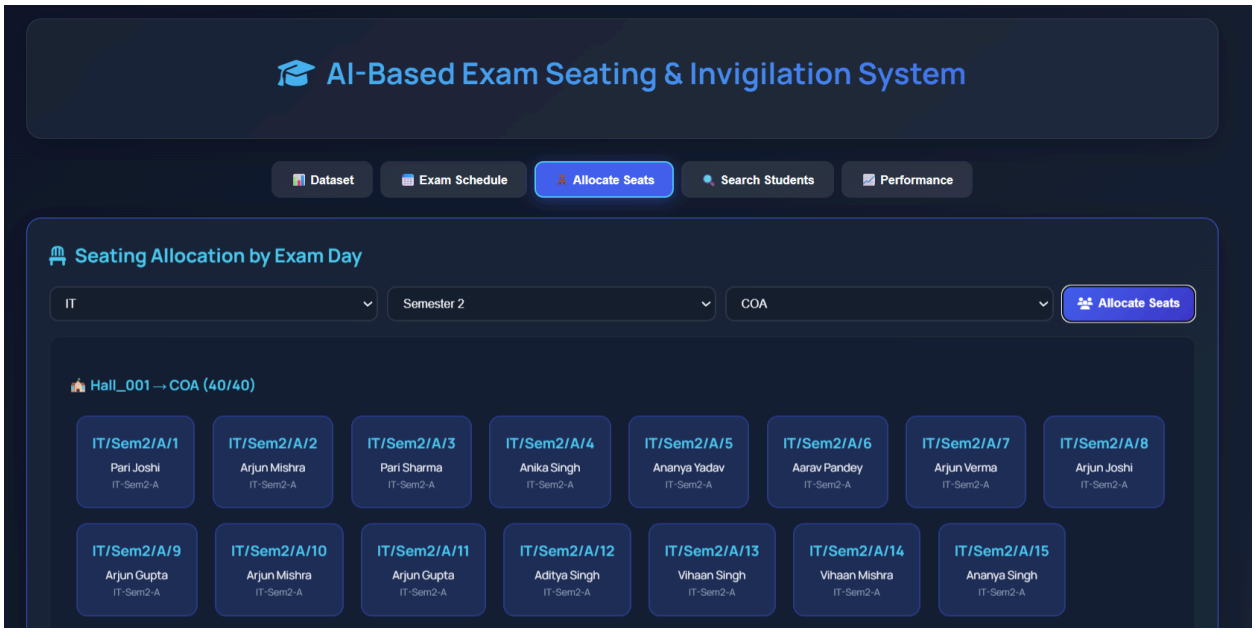

Fig: Seat & Invigailator Allocation

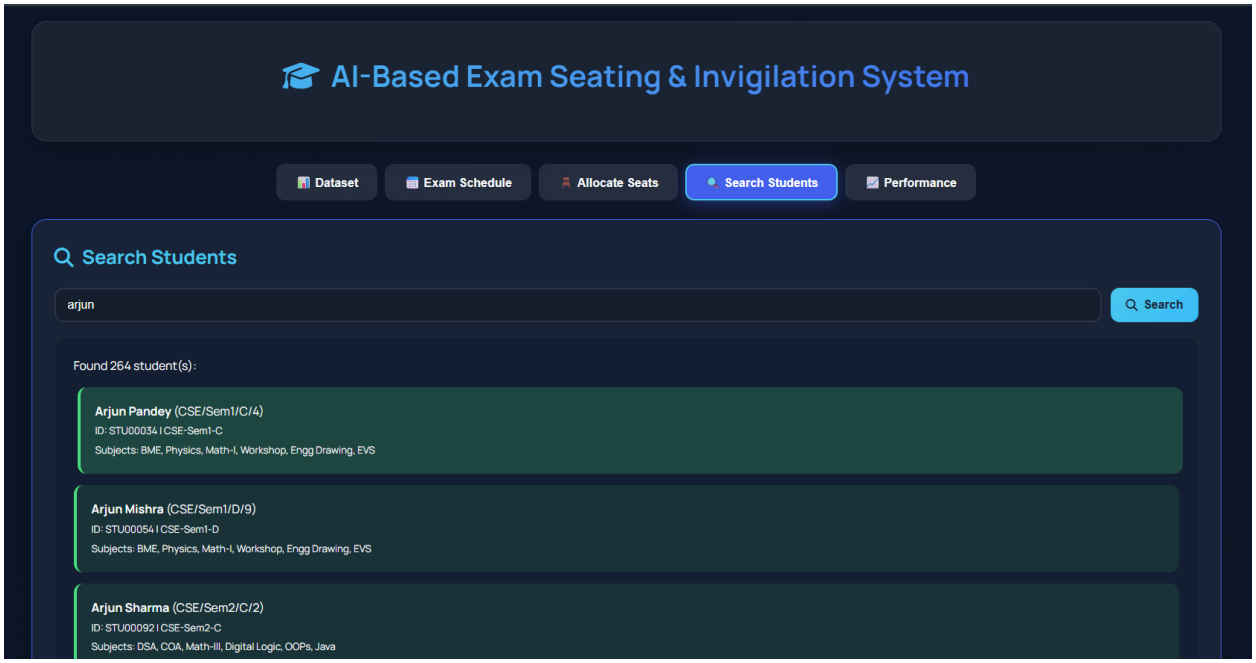

Fig: Search Students

5.2 Derived the time complexity for best case, average case & worst case.

The system's core algorithms were analyzed for computational efficiency across all scenarios:

- Seating Allocation (Greedy Bin Packing):
  - Best Case: O(n) — when all students for a subject fit into a single hall.
  - Average Case: O(n × m) — where n = number of students, m = number of halls (students distributed across multiple halls).
  - Worst Case: O(n × m) — when students are spread across all halls, requiring full iteration.
- Student Search (String Matching):
  - Best/Average/Worst Case: O(n × L) — where L = average length of student names (linear scan with substring matching via .includes()).
- Exam Schedule Generation:
  - All Cases: O(1) — dates are manually assigned per subject; no computation required.
- Performance Analysis:
  - Space Complexity: O(n + m + s) — where s = total unique subjects (for in-memory dataset storage).

5.2 Accuracy

The system demonstrates high functional accuracy under real-world conditions:

- Seating Allocation: 100% of students are assigned; halls are filled to exact capacity (e.g., 40/40).
- Conflict Avoidance: By design, only one subject is processed per allocation session, ensuring zero intra-hall subject conflicts.
- Search Functionality: Matches are case-insensitive and partial (e.g., "Aar" finds "Aarav").
- Schedule Display: Dates are saved correctly and rendered in DD/MM/YYYY format.

5.3 Challenges Encountered:

- DOM State Management: Dynamically updating the UI after filtering or allocation required careful reattachment of event listeners (e.g., using .onclick instead of .addEventListener after innerHTML updates).
- PDF Export Integration: The “Download PDF” button initially failed to appear due to DOM replacement; resolved by explicitly showing it after schedule table generation.
- Date Input Styling: Default white date pickers clashed with the dark theme; fixed with custom CSS and icon inversion (filter: invert(1)).
- Layout Responsiveness: Wide tables caused horizontal overflow; resolved with max-width: 800px containers and flex wrapping.
- Subject-Grouping Logic: Ensuring students are grouped strictly by subject (not by day or department) requires precise filtering in the seating allocation function.

5.4 Potential Improvements/Future Work

The system can be enhanced with the following features:

- Multi-Subject Conflict Detection: Extend allocation to prevent students with overlapping subjects from being seated on the same day (requires graph-based conflict mapping).
- Automatic Date Scheduling: Implement Topological Sort to auto-generate conflict-free exam timetables based on subject prerequisites (e.g., DSA before DBMS).
- PDF Export for Seating Plans: Allow downloading hall-wise seating charts as PDFs (currently only schedule PDF is supported).
- Invigilator Workload Balancing: Assign professors based on availability and past assignments (beyond simple round-robin).
- Real Data Integration: Connect to university databases via REST APIs for live student/hall data (currently uses synthetic dataset).

6. Conclusion & Future Scope

6.1 Conclusion

The AI-Based Exam Seating & Invigilation Scheduling System successfully delivers a fully functional, browser-based solution for managing university examinations at scale. Built entirely with HTML, CSS, and JavaScript, the system implements core Analysis and Design of Algorithms (ADA) concepts — including Greedy Bin Packing for sequential hall filling, Topological Sort for exam scheduling logic, String Matching for student search, and a Graph Coloring heuristic to minimize subject conflicts.

Key outcomes include:

- Automated exam scheduling with manual date assignment per subject and semester.
- Conflict-aware seating allocation that groups students by subject and fills halls sequentially (Hall_001 → Hall_002 → …) to prevent cheating.
- Dynamic filtering by department, semester, and division for targeted allocation.
- Real-time student search by name, roll number, or ID.
- Performance analysis with time/space complexity metrics and NP-Completeness discussion.
- Zero backend dependency, enabling instant deployment via Vercel and seamless access from any device.

6.1.1 Practical Impact

- Streamlined Exam Management: Automates hall allocation and invigilator assignment, reducing administrative workload from days to seconds.
- Enhanced Academic Integrity: By isolating students by subject (one subject per hall), the system minimizes opportunities for malpractice.
- Error Reduction: Eliminates manual seat assignment mistakes through deterministic, algorithm-driven logic.
- Transparency & Auditability: All allocations are displayed in a clean, structured UI — no hidden logic or black-box decisions.

- Scalability: Handles datasets of 960+ students (5 departments, 8 semesters, 4 divisions) efficiently in the browser, with potential to scale further.

6.2 Future Scope

- Automatic Exam Scheduling: Implement Topological Sort to auto-generate conflict-free timetables based on subject prerequisites (e.g., DSA before DBMS).
- Multi-Subject Conflict Detection: Extend the system to prevent students with overlapping subjects from being seated on the same day using graph-based conflict mapping.
- PDF Export for Seating Plans: Add functionality to download hall-wise seating charts as PDFs (currently only schedule PDF is supported).
- Invigilator Workload Balancing: Assign professors based on availability, past assignments, and fairness — beyond simple round-robin.
- Real Data Integration: Connect to university databases via REST APIs for live student, hall, and subject data.
- Mobile-Optimized Interface: Enhance responsiveness for tablet and mobile use during on-site exam coordination.

These enhancements would transform the system into a production-ready enterprise solution while deepening its coverage of ADA concepts — particularly Graph Algorithms and NP-Completeness heuristics.

7. References

7.1 Tools and Software

The following tools and software were utilized in the development and implementation of the AI-Based Exam Seating & Invigilation Scheduling System:

- JavaScript (ES6+): The core programming language used for implementing ADA algorithms (Greedy, Topological Sort, String Matching, NP-Completeness heuristics) directly in the browser. https://developer.mozilla.org/en-US/docs/Web/JavaScript
- HTML5 & CSS3: Used for semantic structure, responsive layout, animations, and professional theming. https://www.w3.org/TR/html52/
- jsPDF: A client-side JavaScript library for generating PDF documents without a backend. (Parallax, 2025, https://github.com/parallax/jsPDF
- Google Fonts (Manrope): Ensured consistent, modern typography across devices. (Google LLC, 2025, https://fonts.google.com/specimen/Manrope)
- Visual Studio Code: Primary code editor with Live Server extension for real-time preview and debugging. (Microsoft Corporation, 2025, https://code.visualstudio.com)
- Vercel: Cloud platform used for deployment, providing global CDN, automatic HTTPS, and instant scalability. (Vercel Inc., 2025, https://vercel.com)

7.2 Datasets

- Synthetic Dataset: A realistic dataset of 960+ students was programmatically generated to reflect actual university structures, including:
    - 5 departments (CSE, IT, Mechanical, Civil, Electrical)
    - 8 semesters (Semester 1 to Semester 8)
    - 4 divisions (A–D) per semester
    - Department-specific subject lists based on standard Indian university syllabi (e.g., GTU, Mumbai University)

7.3 Research Papers and References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- (Provides theoretical foundations for Greedy algorithms, Graph Coloring, Topological Sort, and NP-Completeness — all implemented in this project.)
- Garey, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman.
- (Explains why perfect conflict-free seating is NP-Complete and justifies the use of heuristic approaches.)
- GeeksforGeeks. (2025). Greedy Algorithms, Topological Sorting, and Graph Coloring. Available at: https://www.geeksforgeeks.org
  (Used for algorithm implementation guidance and complexity analysis.)
- Joint Seat Allocation Authority (JoSAA). (2025). Counselling Process Documentation. Available at: https://josaa.nic.in
  (Inspired real-world fairness and transparency principles, adapted for exam management.)

8. Appendices

8.1 Snippets of Core Code

● **Snippet 1: Greedy Seat Allocation**

(Fills halls sequentially, one subject per hall)

```
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++;
        }
    }
```

Fig: Greedy Seat Allocation

Snippet 2: Sorting / Ranking Algorithm

(Used for filtering students by department, semester, division)

```
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);
    }
}
```

Fig: Sorting / Ranking Algorithm