Mini Project Presentation

CAMPUS FLOW

PRESENTED BY:
Ayush Khaire BT22CSD028
Asutosh Sahoo BT22CSD062
Kishor Zatale BT22CSD010
Vasu Parashar BT22CSD021

Guided by: **Mrs. Sakshi Jaiswal**

Campus Flow

INTRODUCTION

CampusFlow is designed to streamline academic processes and reduce manual workload in our institute. By automating repetitive tasks and improving efficiency, our system ensures a seamless flow of operations, enhancing productivity for both faculty and students. With a focus on smart automation and data-driven decision-making, CampusFlow aims to create a more organized and hassle-free campus experience.

PROBLEMS

Problem 1: Inefficient Lecture Review

Students struggle with note-taking and revisiting long lectures, making revision difficult and time-consuming. Without structured summaries, understanding key concepts becomes a challenge.

Problem 2: Manual Exam Seating Hassles

Faculty spend hours manually arranging exam seating, ensuring no two students with the same paper sit together. The process is tedious, error-prone, and inefficient.

Campus Flow

SOLUTION

AUTOMATED LECTURE SUMMARIZATION

Our system records lectures, transcribes them, and generates structured summaries with key points, making revision quick and efficient for students.

EXAM SEAT PLANNING

We automate seat allocation using intelligent algorithms, ensuring a fair, error-free arrangement that saves faculty time and effort.

1. LectureSumm

3 MAIN COMPONENTS

1. **Frontend Client**: A React-based web application
2. **API Server**: A backend service handling requests
3. **Audio Processor**: A Python service using Gemini AI for processing lecture content

CORE FUNCTIONALITY

1. Record or upload lecture audio
2. Process audio into transcripts and summaries
3. Save lectures to a personal dashboard
4. Browse and search lecture content
5. Chat with an AI about lecture content

Campus Flow

HOME SERVICE ABOUT US CONTACT US

INTERFACE

1. Technical Architecture:

Frontend:

- UI Framework: Uses **shadcn/ui** components based on Radix UI primitives
- Styling: **Tailwind CSS** with custom themes
- Routing: **React Route**r for page navigation
- State Management: **React context** for auth and local state

Pages →

- **Dashboard.tsx**: Shows saved lectures
- **Admin.tsx**: Interface for recording/uploading lectures
- **Lecture.tsx**: Displays lecture content with summary
- **Course.tsx**: Course-specific lecture listings

Campus Flow

HOME SERVICE ABOUT US CONTACT US

INTERFACE

1. Technical Architecture:

Api Server:

A Node.js/Express backend that:

- Handles lecture uploads and storage
- Manages user authentication
- Communicates with the database (MongoDB)
- Routes requests to the audio processor

Campus Flow

INTERFACE

1. Technical Architecture:

Audio Processor:

Python service that:

- Processes audio files
- Uses Google's Gemini AI models for analysis
- Generates transcriptions, summaries, and extracts key topics
- Provides chat functionality based on lecture content

The main AI integration is in gemini_client.py, which:

- **Initializes Gemini AI models**
- **Generates structured markdown content from lecture text**
- **Powers the chat feature for answering questions about lectures**

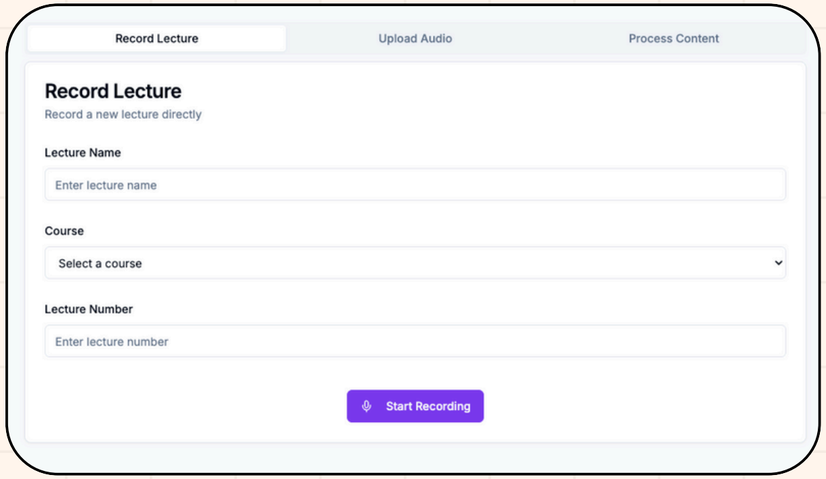
Campus Flow

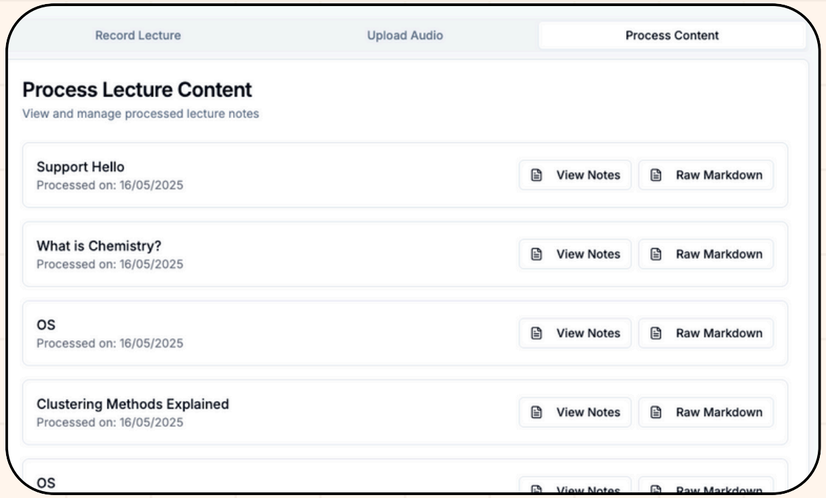
HOME SERVICE ABOUT US CONTACT US

INTERFACE

2. Data Flow:

- User records or uploads audio via frontend
- API server receives the file and stores it
- Audio processor transcribes and analyzes the content
- Processed content is stored in MongoDB
- Frontend displays the processed lecture with summaries





Campus Flow

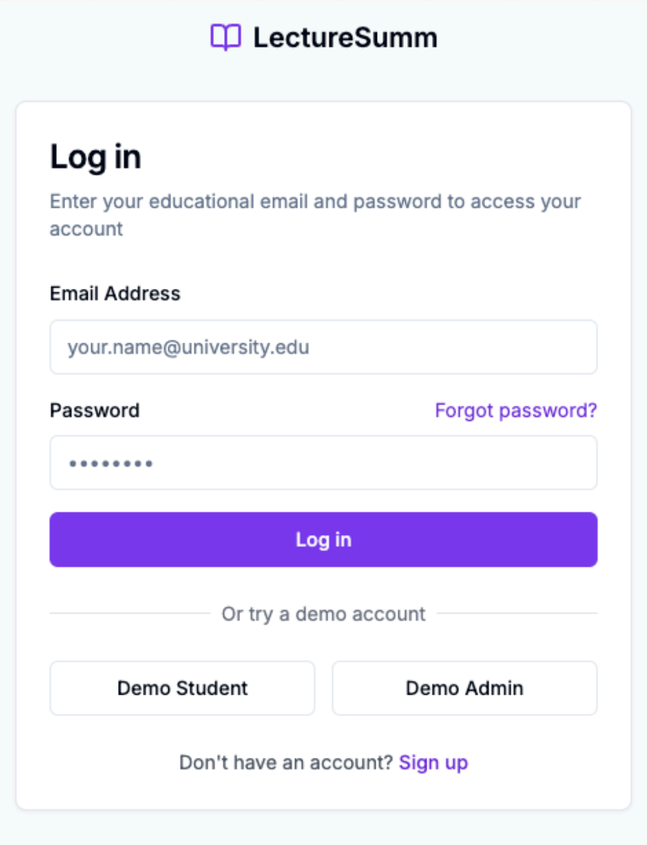
HOME SERVICE ABOUT US CONTACT US

INTERFACE

3. User Roles:

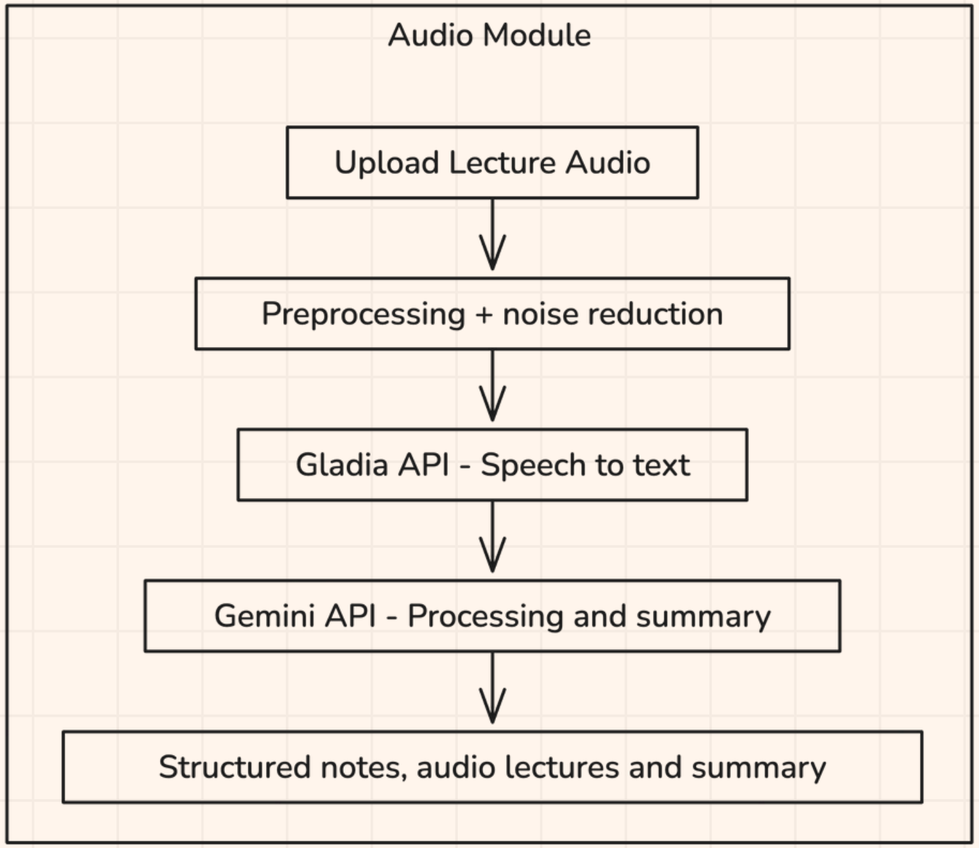
- **Regular Student**
  Can view and save lectures to dashboard.

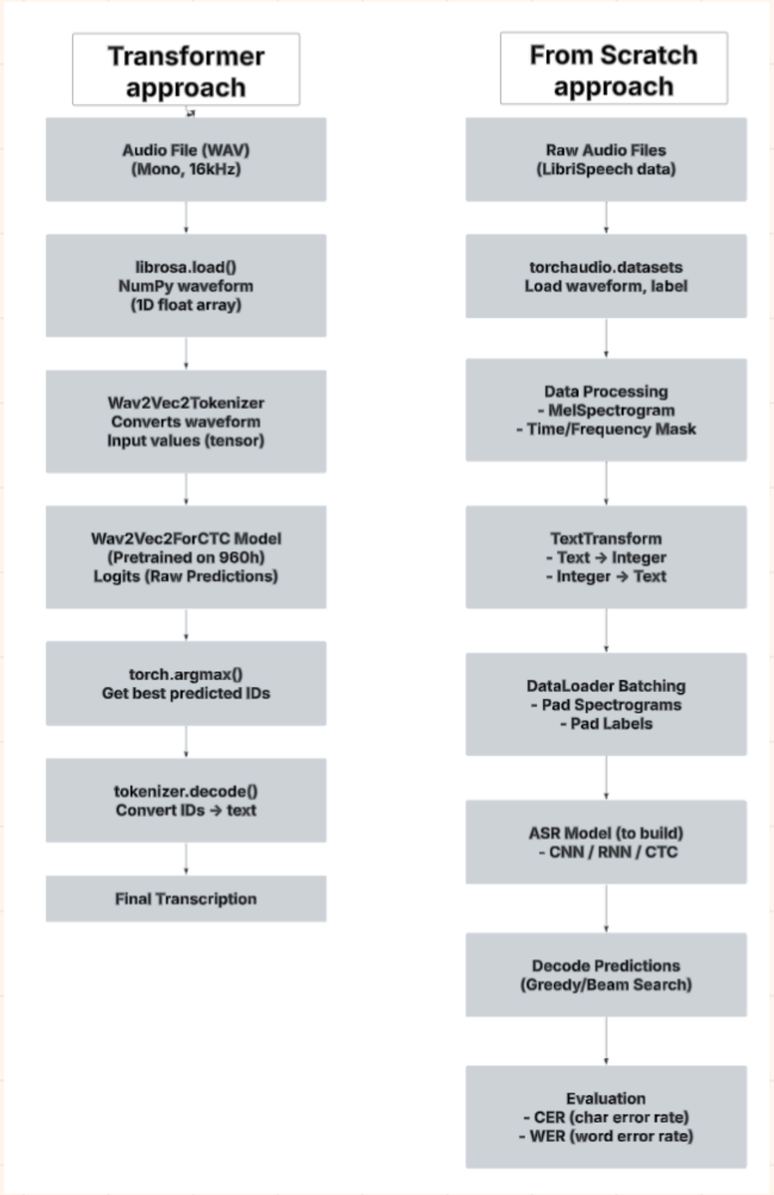
- **Admin Student**
  Can upload, record and process lectures



Campus Flow

FLOWCHART





2. EXAM SEAT ALLOCATION AUTOMATION SYSTEM

Automating exam seat allocation enhances efficiency and accuracy in educational institutions. A comprehensive solution involves developing a software system that assigns seats to students based on predefined criteria, optimizing space utilization, and minimizing manual intervention.

Campus Flow

HOME SERVICE ABOUT US CONTACT US

INTERFACE

1. Technical Architecture:

Frontend:

- **React** with **TypeScript** for building a component-based UI
- **TailwindCSS** for responsive styling
- **Lucide React** for icons
- **React Router** for navigation
- **React Toastify** for notifications
- **React Dropzone** for file uploads

Campus Flow

INTERFACE

1. Technical Architecture:

Backend:

- **Node.js** with **Express.js** for RESTful API development
- **MongoDB** with **Mongoose ODM** for data modeling and persistence
- **XLSX.js** for Excel file generation and parsing
- **Multer** for file upload handling

Campus Flow

HOME SERVICE ABOUT US CONTACT US

INTERFACE

2. Core Features

Exam Management

- Create and configure exams with metadata
- Import student lists from Excel files
- Support for multiple courses per exam
- Automatic student sorting by roll number

Classroom Management

- Define classroom layouts with rows and columns
- Mark unavailable seats for special arrangements
- Group classrooms by building and floor
- Calculate classroom capacity automatically

Campus Flow

HOME SERVICE ABOUT US CONTACT US

INTERFACE

2. Core Features

Seating Plan Generation

- Advanced anti-cheating algorithm
- Optimized student distribution across classrooms
- Support for multiple courses in a single exam
- Real-time preview of seating arrangements

Export and Publishing

- Export to Excel with color-coding by course
- Professional formatting for easy printing
- Status tracking (draft, finalized, published)
- Master sheet with summary information

Campus Flow

HOME SERVICE ABOUT US CONTACT US

INTERFACE

3. Data Models

Exam Model

- Basic exam information (name, date)
- Collection of courses with metadata
- Students grouped by course

Classroom Model

- Physical details (name, building, floor)
- Layout information (rows, columns)
- Unavailable seats tracking

Seating Plan Model

- References to exam and classrooms
- Seat matrix with student assignments
- Statistics for reporting
- Status tracking

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

Column-Based Alternating Course Assignment

- Alternates courses between adjacent columns
- Creates a checkerboard pattern to prevent cheating
- Process classrooms sequentially with optimized distribution

Key Optimizations

- Course Pairing: Calculates optimal course combinations to minimize leftover seats
- Forward Checking: Looks ahead to select courses that best fill available seats
- Dynamic Course Rotation: Rotates courses when one runs out of students

Algorithm Flow

1. Sort courses by size (largest first)
2. Calculate optimal course pairings
3. Process classrooms column by column
4. Apply checkerboard pattern with alternating courses
5. Dynamically adjust course selection based on remaining students

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

1. Preparation Phase

```
// Sort courses by size (largest first)
courseCodes.sort((a, b) => courseStudents[b].length - courseStudents[a].length);

// Find optimal pairing of courses
const optimizedPairs = optimizeCourseDistribution(courseCodes, courseStudents);

let currentCourseIndex = 0;
let nextCourseIndex = optimizedPairs[0] || 1;
```

- **Sort Courses**: We first arrange courses in descending order by student count (largest courses first)
- **Find Optimal Pairs**: We calculate which courses work best together to minimize leftover seats
- **Set Initial Courses**: We select two courses to start with (course 0 and its best pair)

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

2. Seat Allocation Strategy

```
// Process each classroom
for (const classroom of classrooms) {
  // Create empty seat matrix
  const seatMatrix = Array(classroom.rows).fill().map(() => Array(classroom.columns).fill(...));

  // Mark unavailable seats
  classroom.unavailableSeats.forEach(seat => { ... });

  // Fill seats column by column
  for (let col = 0; col < classroom.columns; col++) {
    // Even columns use currentCourseIndex, odd use nextCourseIndex
    const courseCode = courseCodes[col % 2 === 0 ? currentCourseIndex : nextCourseIndex];
    // ...
  }
}
```

- **Create Matrix:** Initialize an empty seat grid with rows and columns
- **Mark Unavailable:** Block any seats marked as unavailable
- **Column-by-Column:** Fill one column at a time
- **Alternating** Courses: Even columns get Course A, odd columns get Course B

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

3. Forward Checking

```
// Calculate available seats in this column
let availableSeatsInColumn = 0;
for (let row = 0; row < classroom.rows; row++) {
  if (!seatMatrix[row][col].isOccupied) availableSeatsInColumn++;
}

// Check if we need a better course
if (students.length < availableSeatsInColumn && courseCodes.length > 2) {
  const optimalCourseIdx = findOptimalCourse(courseStudents, courseCodes,
                                             availableSeatsInColumn,
                                             col % 2 === 0 ? currentCourseIndex : nextCourseIndex);

  // Update course if better match found
}
```

Before filling each column, we look ahead:

- **Count Available Seats**: We check how many seats are available in the column
- **Compare with Student Count**: If the current course doesn't have enough students
- **Find Better Course**: We look for a course that better matches the available seats
- **Update Assignment**: We may switch to a different course for better utilization

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

4. Course Rotation

```
// Move to next course when current course exhausted
if (col % 2 === 0) {
  currentCourseIndex = (currentCourseIndex + 1) % courseCodes.length;
} else {
  nextCourseIndex = (nextCourseIndex + 1) % courseCodes.length;
  if (nextCourseIndex === currentCourseIndex) {
    nextCourseIndex = (nextCourseIndex + 1) % courseCodes.length;
  }
}
```

When a course runs out of students:

- **Cycle Through Courses:** When one course is exhausted, we move to the next course
- **Keep Alternating:** We ensure the two current courses are always different
- **Modulo Operation:** We use modulo to cycle back to the first course when needed

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

5. Key Helper Functions

```
function optimizeCourseDistribution(courseCodes, courseStudents) {
  // For each course, find its best pairing
  for (let i = 0; i < courseCodes.length; i++) {
    // Try each possible pairing
    for (let j = 0; j < courseCodes.length; j++) {
      // Calculate leftover students and balance
      // Choose pairing with minimum leftover and best balance
    }
  }
}
```

A. Optimal Course Distribution

- **Purpose:** Find which courses work best together in adjacent columns
- **Metric 1:** Minimize leftover students (students who won't fit in columns)
- **Metric 2:** Maximize balance between course sizes
- **Output:** For each course index, return the index of its best pair

ALGORITHM

The current seating plan algorithm uses a **greedy column-alternating approach** with **forward checking** and **optimized course pairing**.

5. Key Helper Functions

```
function findOptimalCourse(courseStudents, courseCodes, availableSeats, currentIdx) {
  // Find course whose student count best matches available seats
  for (let i = 0; i < courseCodes.length; i++) {
    // Calculate fit - how close student count is to available seats
    const difference = Math.abs(studentCount - availableSeats);
    // Update if better fit found
  }
}
```

B. Finding Optimal Course

- **Purpose:** Find the best course to fill a specific column
- **Metric:** How closely the number of students matches available seats
- **Output:** Index of the course that best fits the available space

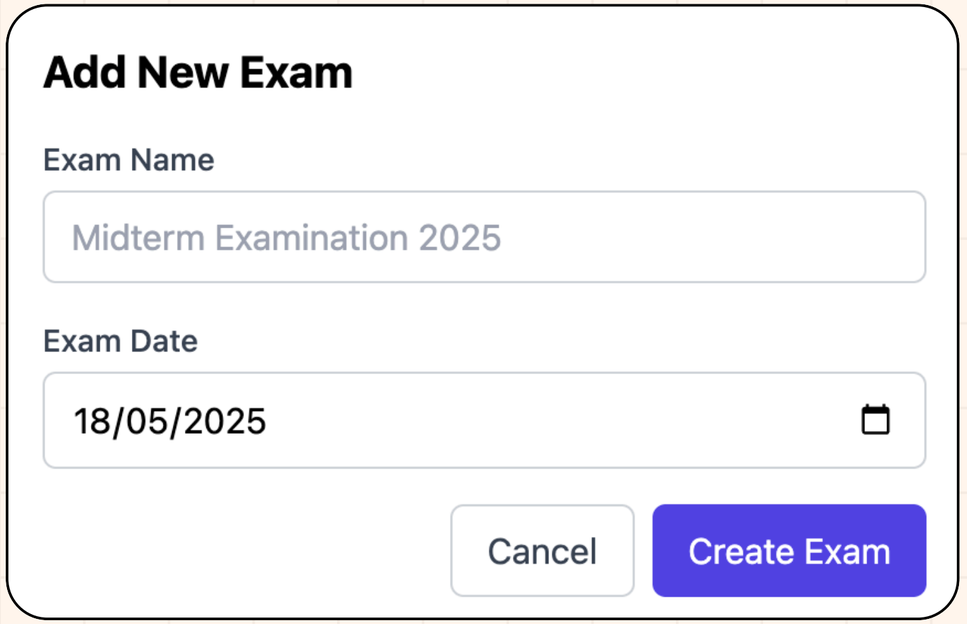
Campus Flow

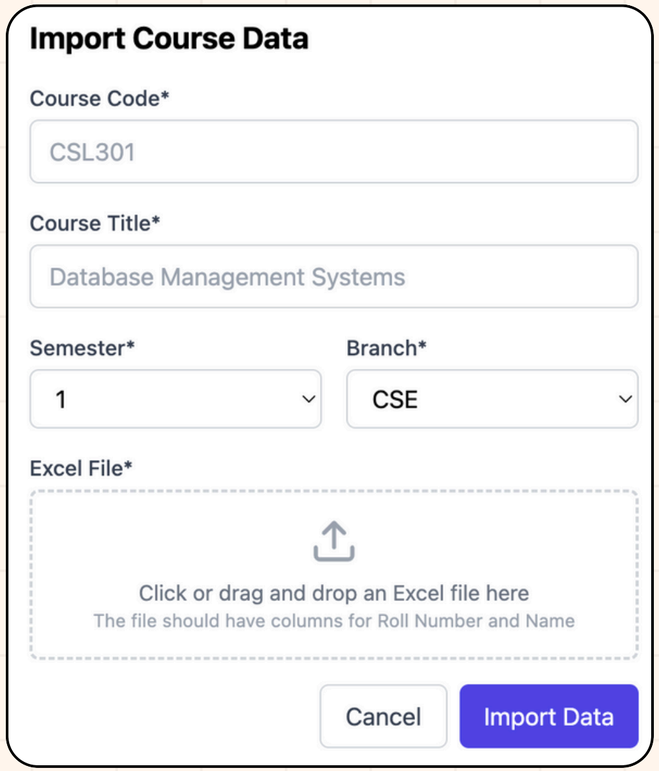
HOME SERVICE ABOUT US CONTACT US

USER WORKFLOW

1. Exam Setup

- Create a new exam with name and date
- Import course data from Excel files
- Review imported student information



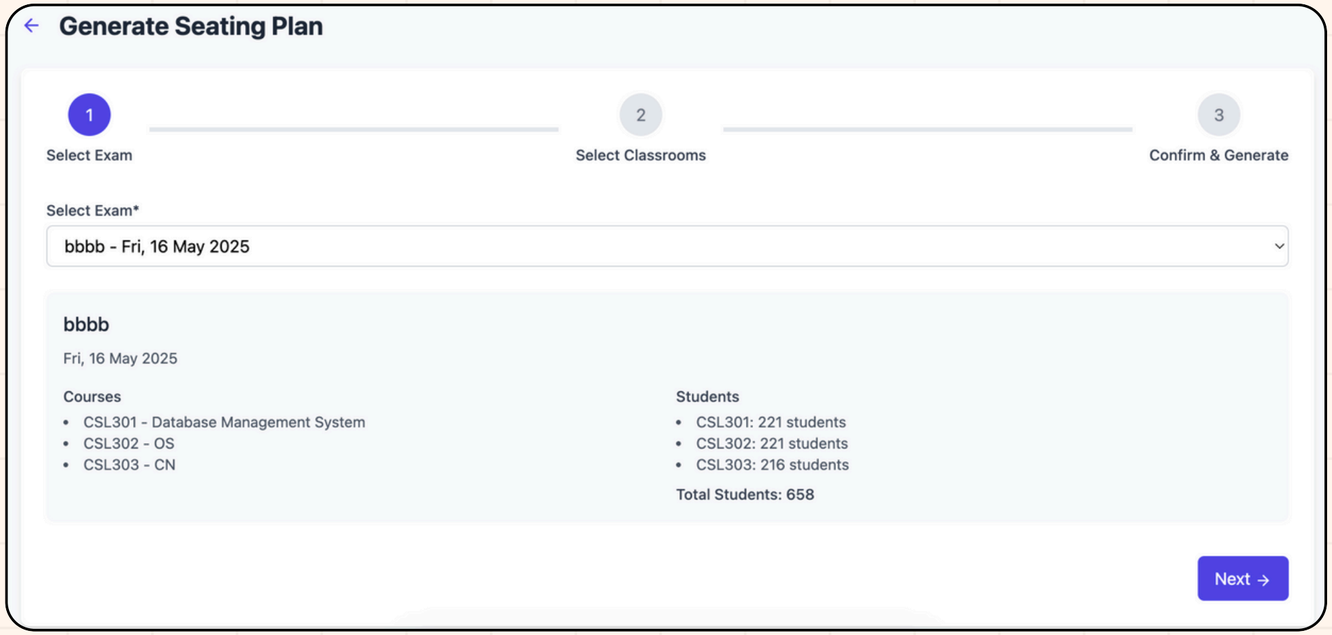


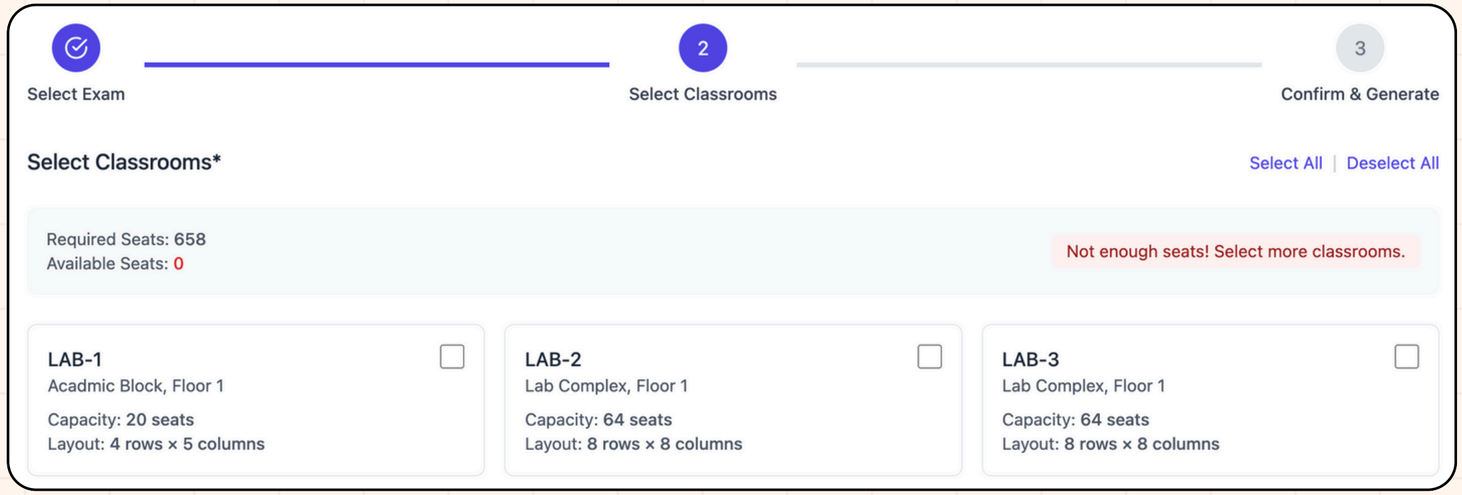
Campus Flow

USER WORKFLOW

2. Seating Plan Generation

- Select exam to generate a plan for
- Choose classrooms to include in the plan
- Generate optimized seating arrangement





Campus Flow

HOME SERVICE ABOUT US CONTACT US

USER WORKFLOW

3. Plan Review and Adjustment

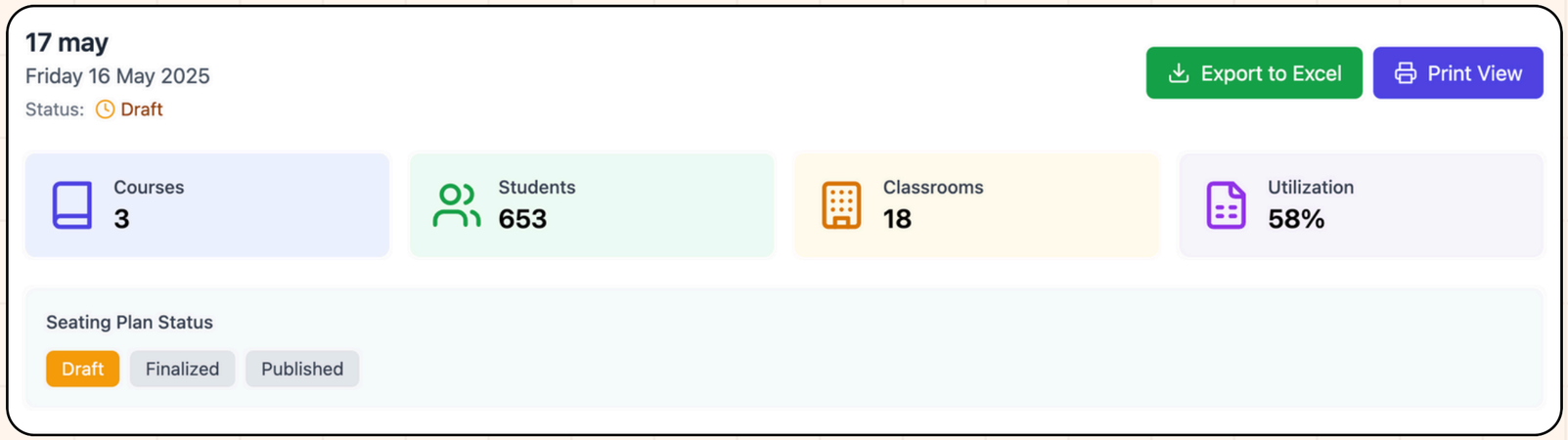
- View generated seating plan with color coding
- Manually adjust individual seats if needed
- Search for specific students in the plan

Row / Col	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	BT20CSE048	BT19CSH001	BT22CSE007	BT19CSH013	BT22CSE018	BT19CSH023
Row 2	BT20CSE137	BT19CSH002	BT22CSE009	BT19CSH014	BT22CSE019	BT19CSH024
Row 3	BT21CSE145	BT19CSH003	BT22CSE010	BT19CSH015	BT22CSE020	BT19CSH025
Row 4	BT21CSE169	BT19CSH004	BT22CSE011	BT19CSH016	BT22CSE021	BT19CSH026
Row 5	BT21CSE200	BT19CSH006	BT22CSE012	BT19CSH017	BT22CSE022	BT19CSH027
Row 6	BT22CSE001	BT19CSH007	BT22CSE013	BT19CSH018	BT22CSE023	BT19CSH028
Row 7	BT22CSE002	BT19CSH009	BT22CSE014	BT19CSH019	BT22CSE024	BT19CSH029
Row 8	BT22CSE003	BT19CSH010	BT22CSE015	BT19CSH020	BT22CSE025	BT19CSH030
Row 9	BT22CSE004	BT19CSH011	BT22CSE016	BT19CSH021	BT22CSE026	BT19CSH031
Row 10	BT22CSE006	BT19CSH012	BT22CSE017	BT19CSH022	BT22CSE027	BT19CSH032

Campus Flow

USER WORKFLOW

- **4. Finalization and Export**
- Update plan status (draft, finalized, published)
- Export to Excel with professional formatting
- Print or share the finalized plan



Campus Flow

HOME SERVICE ABOUT US CONTACT US

RESULTS

The algorithm's combination of greedy allocation, forward checking, and optimized pairing efficiently creates seating plans that prevent cheating while maximizing classroom utilization - all stored in a structured MongoDB document that tracks statistics and enables Excel export with color-coded visualization.

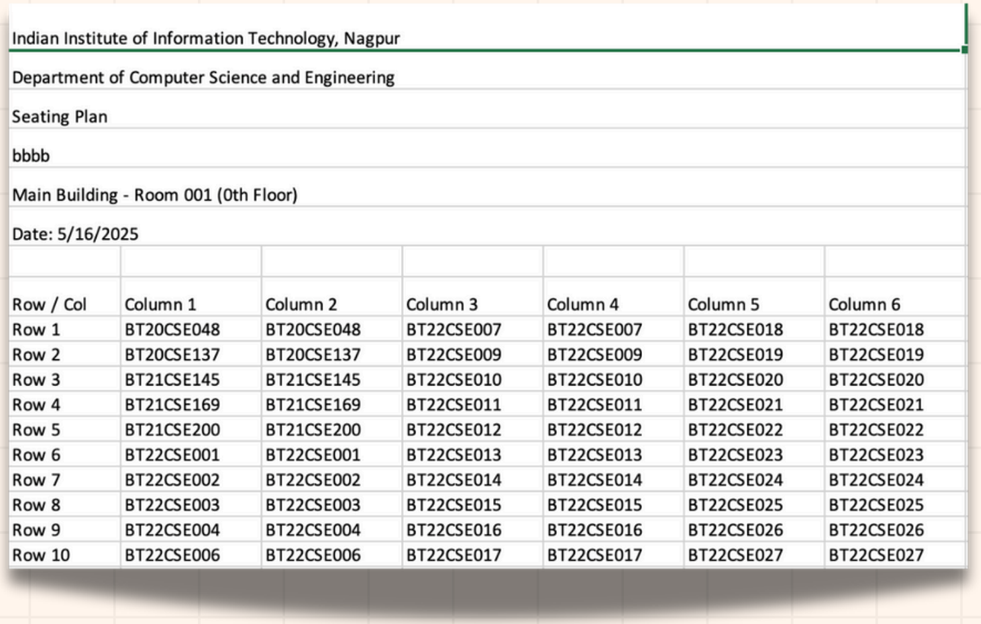
Classroom Seating Plan

SH | H1 | H2 | 201 | 202 | 203 | 204 | 001 | 002 | 101 | 102 | CRE | LAB-1 | LAB-2 | LAB-3 | LAB-4 | LAB-5 | CRE

SH (Main Building)
Floor: 1 | Capacity: 60 seats

Dimensions: 10 rows × 6 columns

Row / Col	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	BT20CSE048	BT19CSH001	BT22CSE007	BT19CSH013	BT22CSE018	BT19CSH023
Row 2	BT20CSE137	BT19CSH002	BT22CSE009	BT19CSH014	BT22CSE019	BT19CSH024
Row 3	BT21CSE145	BT19CSH003	BT22CSE010	BT19CSH015	BT22CSE020	BT19CSH025
Row 4	BT21CSE169	BT19CSH004	BT22CSE011	BT19CSH016	BT22CSE021	BT19CSH026
Row 5	BT21CSE200	BT19CSH006	BT22CSE012	BT19CSH017	BT22CSE022	BT19CSH027
Row 6	BT22CSE001	BT19CSH007	BT22CSE013	BT19CSH018	BT22CSE023	BT19CSH028
Row 7	BT22CSE002	BT19CSH009	BT22CSE014	BT19CSH019	BT22CSE024	BT19CSH029
Row 8	BT22CSE003	BT19CSH010	BT22CSE015	BT19CSH020	BT22CSE025	BT19CSH030
Row 9	BT22CSE004	BT19CSH011	BT22CSE016	BT19CSH021	BT22CSE026	BT19CSH031
Row 10	BT22CSE006	BT19CSH012	BT22CSE017	BT19CSH022	BT22CSE027	BT19CSH032



Indian Institute of Information Technology, Nagpur

Department of Computer Science and Engineering

Seating Plan

bbbb

Main Building - Room 001 (0th Floor)

Date: 5/16/2025

Row / Col	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
Row 1	BT20CSE048	BT20CSE048	BT22CSE007	BT22CSE007	BT22CSE018	BT22CSE018
Row 2	BT20CSE137	BT20CSE137	BT22CSE009	BT22CSE009	BT22CSE019	BT22CSE019
Row 3	BT21CSE145	BT21CSE145	BT22CSE010	BT22CSE010	BT22CSE020	BT22CSE020
Row 4	BT21CSE169	BT21CSE169	BT22CSE011	BT22CSE011	BT22CSE021	BT22CSE021
Row 5	BT21CSE200	BT21CSE200	BT22CSE012	BT22CSE012	BT22CSE022	BT22CSE022
Row 6	BT22CSE001	BT22CSE001	BT22CSE013	BT22CSE013	BT22CSE023	BT22CSE023
Row 7	BT22CSE002	BT22CSE002	BT22CSE014	BT22CSE014	BT22CSE024	BT22CSE024
Row 8	BT22CSE003	BT22CSE003	BT22CSE015	BT22CSE015	BT22CSE025	BT22CSE025
Row 9	BT22CSE004	BT22CSE004	BT22CSE016	BT22CSE016	BT22CSE026	BT22CSE026
Row 10	BT22CSE006	BT22CSE006	BT22CSE017	BT22CSE017	BT22CSE027	BT22CSE027

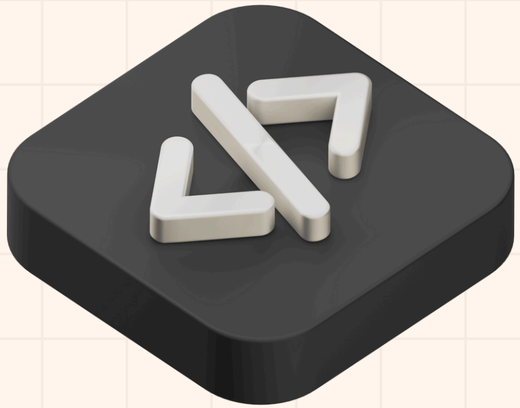
Campus Flow

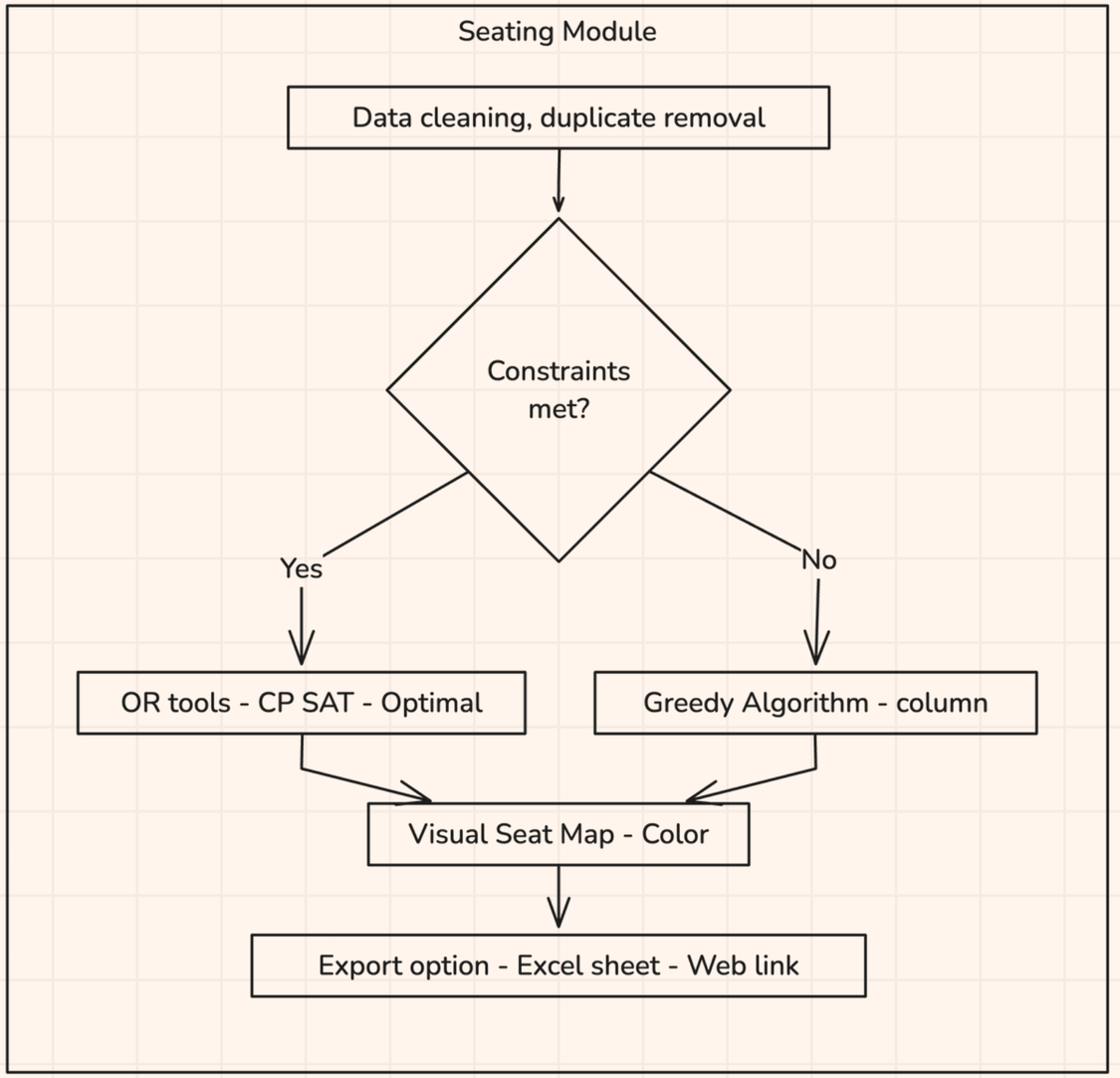
DATASET

Course with there enrolled roll numbers and name of students

	A	B
1	Roll No.	Name
2	BT20CSD001	Gaurav Kumar
3	BT20CSD002	Karumanchi Meghaditya
4	BT20CSD003	Divyansh Nigam
5	BT20CSD004	Aryan Yashwant Pendharkar
6	BT20CSD006	Sujal Rajesh Ahirrao
7	BT20CSD007	Adit Shriyans
8	BT20CSD009	Shivendra Tripathi
9	BT20CSD010	Anay Gour
10	BT20CSD011	Bhavesh Bhikabhau Mankar
11	BT20CSD012	Abhyudya Bhatnagar
12	BT20CSD013	Mudili Rohit
13	BT20CSD014	Nachiket Ningappa Doddamani
14	BT20CSD015	Snehil Shah
15	BT20CSD016	Pratik Nivrutti Nimbole
16	BT20CSD017	Rachit Kakani
17	BT20CSD018	Naval Sunil Makwana
18	BT20CSD019	Adwait Ketan Pande
19	BT20CSD020	Dipesh Chaitanya Singnurkar
20	BT20CSD021	Moksh Pankaj Kajaliya
21	BT20CSD022	Vinayak Sandur
22	BT20CSD023	Sparsh Gautam
23	BT20CSD024	Shivraj Aniruddha Bhalekar
24	BT20CSD025	Anahat Mahesh Kadam
25	BT20CSD026	Putta Siddhardha
26	BT20CSD027	Manish Kumar Soni
27	BT20CSD028	Rakesh Vemula
28	BT20CSD029	Saksham Singh
29	BT20CSD030	Sai Nayan Garipelly
30	BT20CSD031	Siripuram Bhanu Prakash
31	BT20CSD032	Manish Kaushik
32	BT20CSD033	Abhishek Sarkar
33	BT20CSD034	Sujay Jayesh Kamble
34	BT20CSD035	Yash Vikas Pawar
35	BT20CSD036	Arnav Deepak Tiwari
36	BT20CSD037	Shubham Kumar

ARCHITECTURE





FUTURE ENHANCEMENTS

Advanced Constraints

- Support for accessibility requirements
- Custom distance parameters between students
- Time-based constraints for multiple exam sessions

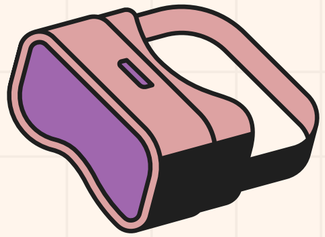
AI Integration

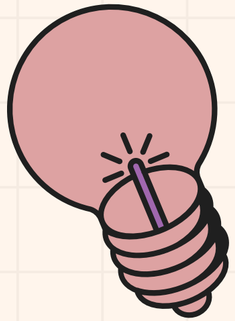
- Integration with Google OR-Tools CP-SAT solver
- Machine learning for pattern detection in cheating
- Predictive analytics for exam planning

Additional Features

- Mobile app for students to find their seats
- Integrated attendance tracking
- Real-time updates during exams

Campus Flow





THANK YOU