EXPNO:16 EXAM HALL SEATING ARRANGEMENT

MINI PROJECT REPORT

Submitted by

STUDENT_NAME: SANTHOSH KR
REG NO: 2117240030132

in partial fulfillment for the award of the

degree of

BACHELOR OF ENGINEERING

IN

CSE(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

RAJALAKSHMI INSTITUTE OF TECHNOLOGY

KUTHAMBAKKAM, CHENNAI - 600 124

ANNA UNIVERSITY: CHENNAI 600 025

NOV/DEC - 2025

ANNA UNIVERSITY: CHENNAI - 600 025

BONAFIDE CERTIFICATE

Certified that this Mini Project report EXAM HALL SEATING ARRANGEMENT is the Bonafide work of **SANTHOSH KR (2117240030132)** , who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Dr. N. KANAGAVALLI, Ph.D	Mrs. M.A.STARLIN, M.E
Assistant Professor,	Assistant Professor,
Artificial Intelligence and Machine Learning,	Artificial Intelligence and Machine Learning,
Rajalakshmi Institute of Technology,	Rajalakshmi Institute of Technology,
Chennai – 600 124.	Chennai – 600 124.

Submitted for the Mini Project viva-voce held on ………………..

INTERNAL EXAMINER **EXTERNAL EXAMINER**

PROJECT DESCRIPTION

Exam Hall Seating Arrangement

The Exam Hall Seating Arrangement system is a software application developed to automate and simplify the process of assigning seats to students during examinations. In most educational institutions, the seating arrangement for exams is usually done manually, which is time-consuming, labor-intensive, and often prone to human errors. This project aims to overcome these limitations by providing an efficient computerized solution that ensures systematic and fair seat allocation.

The system accepts inputs such as student details, examination subjects, roll numbers, and hall capacities. Based on this data, it automatically generates a seating plan according to predefined rules and constraints. The program ensures that students from the same subject or department are not seated close to one another, thereby minimizing the chances of malpractice and maintaining examination integrity. It also helps the administration utilize classroom or hall spaces effectively by optimizing the use of available seats. The application provides a user-friendly interface that allows the administrator to manage exam details, allocate seats, and display or print the final seating arrangement. Once generated, the seating plan can be easily updated or modified as required. This automated approach significantly reduces the workload of the examination committee, improves accuracy, and ensures transparency in exam management. Overall, the Exam Hall Seating Arrangement system enhances the efficiency and reliability of conducting examinations in an organized manner.

EXAM HALL SEATING ARRANGEMENT

PO–PSO Mapping Table:

PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
3	3	3	2	3	1	-	2	1	2	2	3	3	2	1

PSO Justification:

PSO	Relevance	Justification
PSO1 – Apply fundamental computing knowledge	3	Applies OOP principles, Java Swing GUI, and MySQL database integration to build an automated exam hall seating system.
PSO2 – Design and implement solutions	2	Implements modular DAO structure and seat allocation algorithms for efficient and accurate arrangement generation.
PSO3 – Use modern tools and technologies	1	Uses Java, JDBC, and MySQL tools effectively, reflecting awareness of contemporary development practices.

PO Justification:

PO	Relevance	Justification
PO1 – Engineering Knowledge	3	Applies engineering fundamentals through the use of Java, JDBC, and database management to create a functional software system..
PO2 – Problem Analysis	3	Analyzes student and hall data to design logic for systematic and optimized seat allocation.
PO3 –	3	Implements multi-layer architecture: GUI →

PO4 – Investigations	**2**	**Involves testing seat allocation accuracy, database queries, and validation of input data.**
PO5 – Modern Tool Usage	**3**	**Uses Java Swing, JDBC, and MySQL Workbench fo management.**
PO6 – The Engineer & Society	**1**	**Contributes to digital transformation of academic exam management by replacing manual processes.**
PO8 – Ethics	**2**	**Encourages ethical handling and secure storage of student information.**
PO9 – Team Work	**1**	**Can be developed collaboratively, enabling modular division of responsibilities.**
PO10 – Communication	**2**	**Improves documentation, report writing, and structured code communication between modules.**
PO11 – Project Management	**2**	**Demonstrates task division, scheduling, and effective modular project planning.**
PO12 – Lifelong Learning	**3**	**Promotes continuous learning through integration of new technologies like JDBC, GUI design, and database optimization.**

Introduction

In educational institutions, preparing seating arrangements for examinations manually is a time-consuming and error-prone task. As the number of students and exam halls increases, manual allocation can lead to confusion, duplication, or uneven distribution of students.

The **Exam Hall Seating Arrangement System** automates this process by assigning students to halls and seats efficiently using a database-driven approach. The system provides a user-friendly interface to input student details and hall capacities, then generates a fair and optimized seating plan automatically.

This mini project demonstrates the practical application of **Object-Oriented Programming (OOP)** concepts, **Java Swing GUI**, and **MySQL database connectivity** using **JDBC**. It ensures structured data management, faster execution, and reduced human effort. Thus, this project represents an essential step toward digital transformation in academic administration.

Project Description

The **Exam Hall Seating Arrangement System** is a GUI-based application developed using **Java Swing** and **MySQL**.
It simplifies and automates the allocation of students to exam halls by reading data from the database and distributing seats according to hall capacities.

Features:

- Allows users to add student records (name, roll number, department) and hall details (hall name, capacity).
- Automatically assigns students to halls based on available capacity.
- Generates seating plans and stores them in a MySQL database.
- Displays seat allocation details on the interface in a tabular form.
- Provides options to log allocation progress and save records.

Technical Highlights:

- Developed using **Java (Swing for GUI)**.
- Uses **MySQL** for data storage and **JDBC** for database connectivity.
- Implements **DAO classes** for database abstraction.
- Follows a modular design with separate components for student, hall, and seating management.

The system demonstrates essential software engineering principles like modular programming, abstraction, encapsulation, and database integration within a single cohesive mini project.

Objectives

The main objectives of the project are:

1. To automate the process of generating seating arrangements for examinations.
2. To implement a GUI interface that simplifies data entry for students and exam halls.
3. To design and develop the system using OOP concepts in Java.
4. To store and manage student and hall data efficiently using MySQL.
5. To ensure fairness by sequentially or randomly allocating students to available halls.
6. To reduce manual workload and eliminate data duplication.
7. To provide exportable or displayable seating plans that can be printed or shared digitally.
8. To demonstrate integration between Java front-end and MySQL back-end.

System Requirements and Setup

Hardware Requirements

1. Processor: Intel i3 or above
2. RAM: 4 GB or higher
3. Hard Disk: Minimum 1 GB of free space
4. Display: 1024×768 resolution or higher

Software Requirements

1. Operating System: Windows / Linux / macOS
2. JDK Version: JDK 8 or above
3. IDE / Text Editor: Eclipse / IntelliJ IDEA / VS Code / NetBeans
4. Database: MySQL 8.0 or above
5. JDBC Connector: MySQL Connector/J

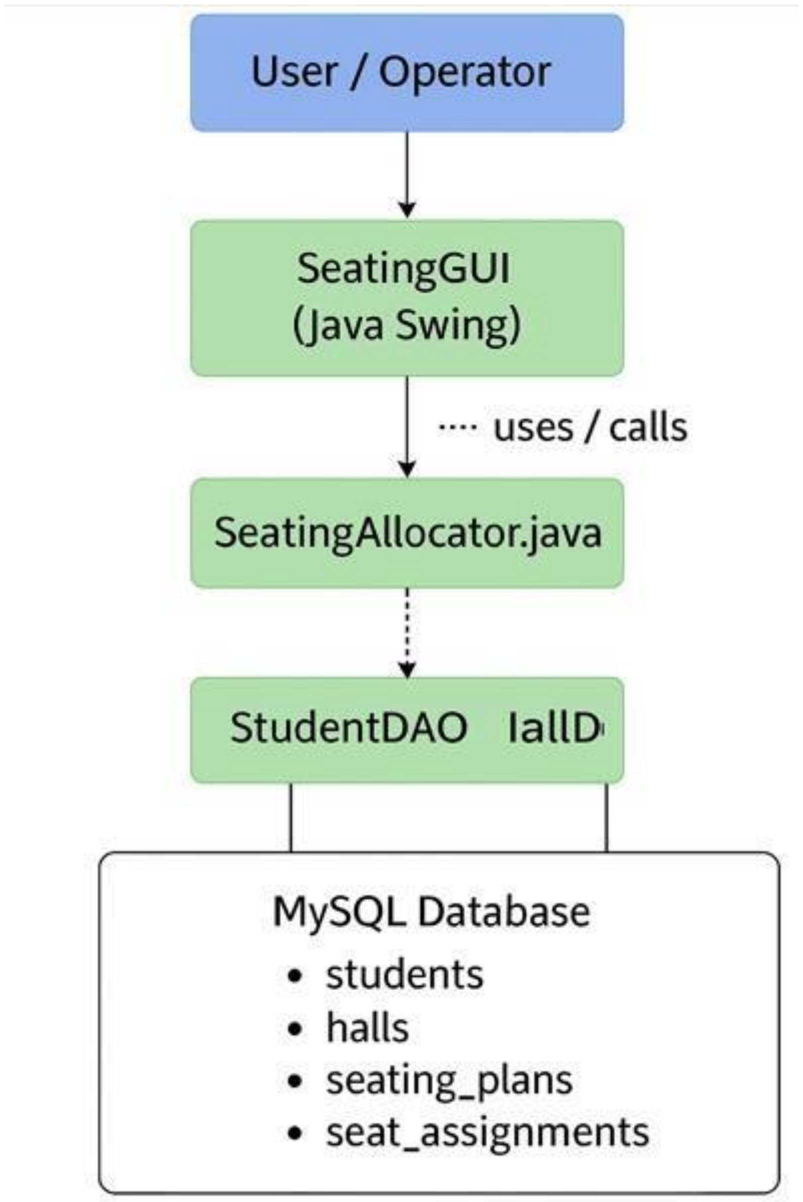
Setup Instructions

1. **Install Java Development Kit (JDK)**
   - Download and install JDK from Oracle or OpenJDK.
   - Set the **PATH** environment variable for Java.

2. **Install MySQL Server and Workbench**
    - Download MySQL and complete the setup.
    - Open MySQL Workbench and create a database using:
    - CREATE DATABASE exam_hall_db;
3. **Create Required Tables**
    - Tables:
        - students
        - halls
        - exams
        - seating_plans
        - seat_assignments
4. **Import the Project**
    - Open your IDE (Eclipse / IntelliJ IDEA).
    - Create a new Java project named **ExamHallSeatingArrangement**.
    - Add all .java source files into the com.example package.
5. **Add MySQL JDBC Connector**
    - Download mysql-connector-j.jar.
    - Add it to your project's classpath:
      Project → Properties → Java Build Path → Libraries → Add External JARs.
6. **Compile and Run the Application**
    - Open the file SeatingGUI.java.
    - Run the program.
    - The application window will open for adding students, halls, and generating seating plans.

System Architecture :

High-Level System Architecture:



Code: Module 1 ---Student.java

Represents a student with roll number, name, and department details.

```
package com.example;

public class Student {
    private int id;
    private String rollNo;
    private String name;
    private String department;

    public Student(String rollNo, String name, String department) {
        this.rollNo = rollNo;
        this.name = name;
        this.department = department;
    }

    public Student(int id, String rollNo, String name, String department) {
        this.id = id;
        this.rollNo = rollNo;
        this.name = name;
        this.department = department;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getRollNo() { return rollNo; }
    public void setRollNo(String rollNo) { this.rollNo = rollNo; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
public String getDepartment() { return department; }
public void setDepartment(String department) { this.department = department; }
@Override
public String toString() {
return rollNo + " | " + name + " | " + department;
}
}
```

Handles DB connection creation using JDBC.
package com.example;

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBUtil {

private static final String URL =

"jdbc:mysql://localhost:3306/exam_hall_db?useSSL=false&allowPublicKeyRetrieval=true&s
erverTi mezone=UTC";

private static final String USER = "root"; // your DB user

private static final String PASS = "12345"; // your MySQL password

public static Connection getConnection() throws SQLException { return

DriverManager.getConnection(URL, USER, PASS);

}

}
```

StudentDAO.java

Performs database operations (CRUD) for the students table.

```
package com.example;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
public class StudentDAO {

    public void addStudent(Student s) {
        final String sql = "INSERT INTO students (roll_no, name, section) VALUES (?, ?, ?)";
        try (Connection conn = DBUtil.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            ps.setString(1, s.getRollNo());
            ps.setString(2, s.getName());
            ps.setString(3, s.getDepartment());
            ps.executeUpdate();
            try (ResultSet rs = ps.getGeneratedKeys()) {
                if (rs.next()) s.setId(rs.getInt(1));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public List<Student> getAllStudents() {
        List<Student> list = new ArrayList<>();
        final String sql = "SELECT student_id, roll_no, name, section FROM students ORDER BY
student_id";
        try (Connection conn = DBUtil.getConnection();
             Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                list.add(new Student(
                        rs.getInt("student_id"),
                        rs.getString("roll_no"),
                    rs.getString("name"),
                    rs.getString("section")
                ));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return list;
    }
    public int getStudentIdByRoll(String rollNo) {
        final String sql = "SELECT student_id FROM students WHERE roll_no=? LIMIT 1";
        try (Connection conn = DBUtil.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, rollNo);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) return rs.getInt(1);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1;
    }
}
```

Hall.java

Represents exam hall details including hall name and capacity.

```
package com.example;

public class Hall {
    private int id;
    private String name;
    private int capacity;
    public Hall(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
    }

    public Hall(int id, String name, int capacity) {
        this.id = id;
        this.name = name;
        this.capacity = capacity;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getCapacity() { return capacity; }
    public void setCapacity(int capacity) { this.capacity = capacity; }

    @Override
    public String toString() {
        return name + " (Cap: " + capacity + ")";
    }
}
```

HallDAO.java

Handles database operations for the halls table.

```
package com.example;

import java.sql.*;
import java.util.*;
```

```
public class HallDAO {


    public void addHall(Hall h) {
        String sql = "INSERT INTO halls (hall_name, capacity) VALUES (?, ?)";
        try (Connection conn = DBUtil.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, h.getName());
            ps.setInt(2,  h.getCapacity());
            ps.executeUpdate();
            try (ResultSet rs = ps.getGeneratedKeys()) {
                if (rs.next()) h.setId(rs.getInt(1));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    public    List<Hall>    getAllHalls()    {
        List<Hall> list = new ArrayList<>();
        String sql = "SELECT hall_id, hall_name, capacity FROM halls ORDER BY hall_id";
        try (Connection conn = DBUtil.getConnection();
             Statement st = conn.createStatement();
            ResultSet  rs  =  st.executeQuery(sql))  {
            while (rs.next()) {
                list.add(new Hall(
                        rs.getInt("hall_id"),
                        rs.getString("hall_name"),
                        rs.getInt("capacity")
                ));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return list;
    }
}
```

SeatingAllocator.java

Implements logic to allocate seats to students based on available hall capacities.

```
package com.example;

import java.util.*;

public class SeatingAllocator {

    public static class SeatAssignment {
        public String hallName;
        public int seatNo;
        public String rollNo;
        public String name;
        public String department;

        public SeatAssignment(String hallName, int seatNo, String rollNo, String name, String
department) {
            this.hallName = hallName;
            this.seatNo = seatNo;
            this.rollNo = rollNo;
            this.name = name;
            this.department = department;
        }
    }

    public List<SeatAssignment> allocate(List<Student> students, List<Hall> halls) {
        List<SeatAssignment> assigned = new ArrayList<>();
        int studentIndex = 0;
        for (Hall h : halls) {
            for (int seat = 1; seat <= h.getCapacity() && studentIndex < students.size(); seat++) {
                Student s = students.get(studentIndex++);
                assigned.add(new SeatAssignment(h.getName(), seat, s.getRollNo(), s.getName(),
s.getDepartment()));
            }
        }
        return assigned;
    }
}
```

SeatingPlanDAO.java

Manages database insertion of exam plans and seat assignments.

```
package com.example;
import java.sql.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
public class SeatingPlanDAO {
    private final StudentDAO studentDAO = new StudentDAO();
    private final HallDAO hallDAO = new HallDAO();
    public int createPlanAndSaveAssignments(String examName,
List<SeatingAllocator.SeatAssignment> assignments) throws Exception {
        Connection con = null;
        try {
            con = DBUtil.getConnection();
            con.setAutoCommit(false);
            int examId = createExam(con, examName, assignments.size());
            int planId = createSeatingPlan(con, examId, "Admin", "Auto-generated");
            Map<String, Integer> hallNameToId = new HashMap<>();
            List<Hall> halls = hallDAO.getAllHalls();
            for (Hall h : halls) hallNameToId.put(h.getName(), h.getId());
            final String sql = "INSERT INTO seat_assignments (plan_id, student_id, hall_id, seat_no)
VALUES (?, ?, ?, ?)";
```

```
            try (PreparedStatement ps = con.prepareStatement(sql)) {
                for (SeatingAllocator.SeatAssignment a : assignments) {
                    int studentId = studentDAO.getStudentIdByRoll(a.rollNo);
                    Integer hallId = hallNameToId.get(a.hallName);
                    if (studentId > 0 && hallId != null) {
                        ps.setInt(1, planId);
                        ps.setInt(2, studentId);
                        ps.setInt(3, hallId);
                        ps.setInt(4, a.seatNo);
                        ps.addBatch();
                    }
                }
                ps.executeBatch();
            }
            con.commit();
            return planId;
        } catch (Exception ex) {
            if (con != null) con.rollback();
            throw ex;
        } finally {
            if (con != null) { con.setAutoCommit(true); con.close(); }
        }
    }
    private int createExam(Connection con, String name, int total) throws SQLException {
        String sql = "INSERT INTO exams (exam_name, exam_date, total_students) VALUES (?, 
CURDATE(), ?)";
        try (PreparedStatement ps = con.prepareStatement(sql, 
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, name);
            ps.setInt(2, total);
            ps.executeUpdate();
            try (ResultSet rs = ps.getGeneratedKeys()) { if (rs.next()) return rs.getInt(1); }
        }
        return -1;
    }
    private int createSeatingPlan(Connection con, int examId, String createdBy, String notes) throws
SQLException {
        String sql = "INSERT INTO seating_plans (exam_id, created_by, notes) VALUES (?, ?, ?)";
        try (PreparedStatement ps = con.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setInt(1, examId);
            ps.setString(2, createdBy);
            ps.setString(3, notes);
            ps.executeUpdate();
            try (ResultSet rs = ps.getGeneratedKeys()) { if (rs.next()) return rs.getInt(1); }
        }
        return -1;
    }
}
```

SeatingGUI.java

Provides Java Swing GUI for adding students and halls, allocating seats, displaying results, and saving seating plans to the MySQL database.

```
package com.example;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.sql.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
public class SeatingGUI extends JFrame {
    // UI fields
    private JTextField rollField;
    private JTextField nameField;
    private JTextField deptField;
    private JTextField hallField;
    private JTextField capField;
    private JTable table;
    private JTextArea logArea;
    private final StudentDAO studentDAO = new StudentDAO();
    private final HallDAO hallDAO = new HallDAO();
    public SeatingGUI() {
        setTitle("Exam Hall Seating Arrangement");
        setSize(980, 620);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        initComponents();
    }
    private void initComponents() {
        // Left column: inputs and buttons
        JPanel leftPanel = new JPanel();
        leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
        leftPanel.setPreferredSize(new Dimension(360, 0));
        JPanel studentPanel = new JPanel(new GridLayout(4, 2, 6, 6));
        studentPanel.setBorder(BorderFactory.createTitledBorder("Add Student"));
        rollField = new JTextField();
        nameField = new JTextField();
        deptField = new JTextField();
        JButton addStudentBtn = new JButton("Add Student");
        studentPanel.add(new JLabel("Roll No:"));
        studentPanel.add(rollField);
        studentPanel.add(new JLabel("Name:"));
        studentPanel.add(nameField);
        studentPanel.add(new JLabel("Department:"));
        studentPanel.add(deptField);
        studentPanel.add(new JLabel());
        studentPanel.add(addStudentBtn);
        JPanel hallPanel = new JPanel(new GridLayout(3, 2, 6, 6));
        hallPanel.setBorder(BorderFactory.createTitledBorder("Add Hall"));
        hallField = new JTextField();
        capField = new JTextField();
        JButton addHallBtn = new JButton("Add Hall");
        hallPanel.add(new JLabel("Hall Name:"));
        hallPanel.add(hallField);
```

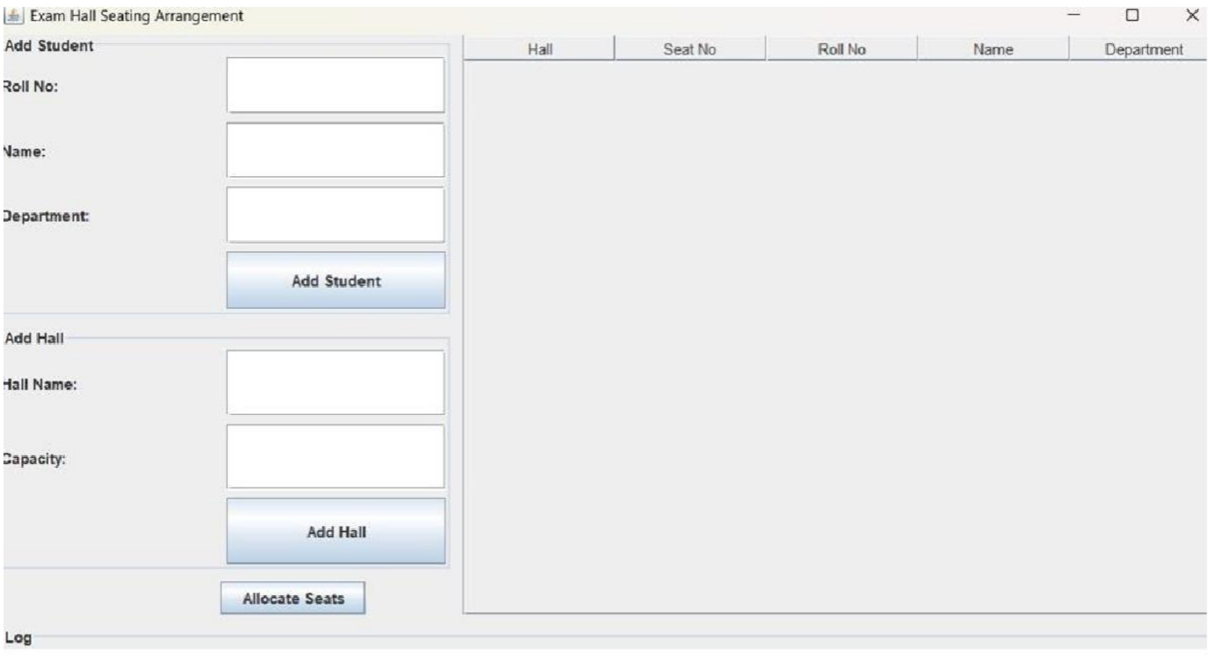
```
        hallPanel.add(new JLabel("Capacity:"));
        hallPanel.add(capField);
        hallPanel.add(new JLabel());
        hallPanel.add(addHallBtn);
        JButton allocateBtn = new JButton("Allocate Seats");
        leftPanel.add(studentPanel);
        leftPanel.add(Box.createRigidArea(new Dimension(0, 8)));
        leftPanel.add(hallPanel);
        leftPanel.add(Box.createRigidArea(new Dimension(0, 8)));
        leftPanel.add(allocateBtn);
        DefaultTableModel model = new DefaultTableModel(
                new Object[]{"Hall", "Seat No", "Roll No", "Name", "Department"}, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        table = new JTable(model);
        JScrollPane tableScroll = new JScrollPane(table);
        logArea = new JTextArea(6, 10);
        logArea.setEditable(false);
        JScrollPane logScroll = new JScrollPane(logArea);
        logScroll.setBorder(BorderFactory.createTitledBorder("Log"));
        getContentPane().setLayout(new BorderLayout(8, 8));
        getContentPane().add(leftPanel, BorderLayout.WEST);
        getContentPane().add(tableScroll, BorderLayout.CENTER);
        getContentPane().add(logScroll, BorderLayout.SOUTH);
        addStudentBtn.addActionListener(e -> addStudentToDB());
        addHallBtn.addActionListener(e -> addHallToDB());
        allocateBtn.addActionListener(e -> allocateSeatsNow());
    }
    private void addStudentToDB() {
        String roll = rollField.getText().trim();
        String name = nameField.getText().trim();
        String dept = deptField.getText().trim();
        if (roll.isEmpty() || name.isEmpty() || dept.isEmpty()) {
            appendLog("+ Please fill all student fields.");
            return;
        }
        try {
            Student s = new Student(roll, name, dept);
            studentDAO.addStudent(s);
            appendLog("✅ Added Student: " + s.toString());
            rollField.setText("");
            nameField.setText("");
            deptField.setText("");
        } catch (Exception ex) {
            appendLog("+ Error adding student: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
    private void addHallToDB() {
        String hallName = hallField.getText().trim();
        String capText = capField.getText().trim();
        if (hallName.isEmpty() || capText.isEmpty()) {
            appendLog("+ Please fill all hall fields.");
            return;
        }
        try {
            int cap = Integer.parseInt(capText);
            Hall h = new Hall(hallName, cap);
            hallDAO.addHall(h);
            appendLog("✅ Added Hall: " + h.toString());
            hallField.setText("");
            capField.setText("");
        } catch (NumberFormatException nfe) {
            appendLog("¡ Capacity must be a number.");
        } catch (Exception ex) {
            appendLog("+ Error adding hall: " + ex.getMessage());
            ex.printStackTrace();
        }
```

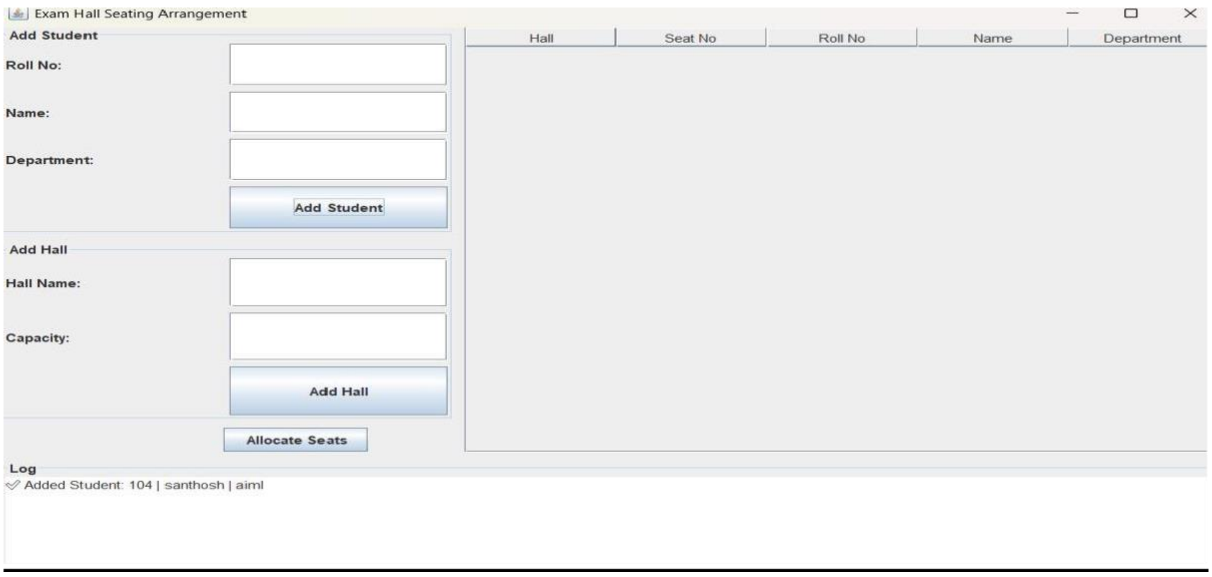
```
}
private void allocateSeatsNow() {
    try {
        List<Student> students = studentDAO.getAllStudents();
        List<Hall> halls = hallDAO.getAllHalls();
        if (students.isEmpty()) {
            appendLog("¡ No students to allocate.");
            return;
        }
        if (halls.isEmpty()) {
            appendLog("¡ No halls available.");
            return;
        }
        SeatingAllocator allocator = new SeatingAllocator();
        List<SeatingAllocator.SeatAssignment> assignments = allocator.allocate(students, halls);
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        model.setRowCount(0);
        for (SeatingAllocator.SeatAssignment a : assignments) {
            model.addRow(new Object[]{a.hallName, a.seatNo, a.rollNo, a.name, a.department});
        }
        appendLog("📊 Total Students: " + students.size() + ", Assigned: " + assignments.size());
        saveAssignmentsToDB(assignments, students.size());
    } catch (Exception ex) {
        appendLog("+ Allocation error: " + ex.getMessage());
        ex.printStackTrace();
    }
}
private void saveAssignmentsToDB(List<SeatingAllocator.SeatAssignment> assignments, int totalStudents) {
    Connection con = null;
    try {
        con = DBUtil.getConnection();
        con.setAutoCommit(false);
        // 1) create exam
        int examId;
        String examSQL = "INSERT INTO exams (exam_name, exam_date, total_students) VALUES (?, CURDATE(), ?)";
        try (PreparedStatement ps = con.prepareStatement(examSQL, Statement.RETURN_GENERATED_KEYS)) {
```

```
    ps.setString(1, "AutoExam");
    ps.setInt(2, totalStudents);
    ps.executeUpdate();
    try (ResultSet rs = ps.getGeneratedKeys()) {
        if (rs.next()) examId = rs.getInt(1);
        else throw new SQLException("Failed to create exam (no id generated).");
    }
}
// 2) create seating plan
int planId;
String planSQL = "INSERT INTO seating_plans (exam_id, created_by, notes) VALUES (?, ?, ?)";
try (PreparedStatement ps = con.prepareStatement(planSQL, Statement.RETURN_GENERATED_KEYS)) {
    ps.setInt(1, examId);
    ps.setString(2, "Admin");
    ps.setString(3, "Auto-generated plan");
    ps.executeUpdate();
    try (ResultSet rs = ps.getGeneratedKeys()) {
        if (rs.next()) planId = rs.getInt(1);
        else throw new SQLException("Failed to create seating plan (no id).");
    }
}
// 3) prepare mapping hallName -> hallId (so we avoid extra DB lookups inside loop)
Map<String, Integer> hallNameToId = new HashMap<String, Integer>();
List<Hall> halls = hallDAO.getAllHalls();
for (Hall h : halls) {
    hallNameToId.put(h.getName(),  h.getId());
}
// 4) insert assignments
String assignSQL = "INSERT INTO seat_assignments (plan_id, student_id, hall_id, seat_no) VALUES (?, ?, ?, ?)";
try (PreparedStatement ps = con.prepareStatement(assignSQL)) {
    for (SeatingAllocator.SeatAssignment a : assignments) {
        int studentId = studentDAO.getStudentIdByRoll(a.rollNo);
        if (studentId <= 0) {
            // skip or warn
            System.out.println("WARN: unknown student roll " + a.rollNo + " - skipping DB insert for this record.");
            continue;
            }
            Integer hallId = hallNameToId.get(a.hallName);
            if (hallId == null) {
                // try reloading halls once
                List<Hall> refreshed = hallDAO.getAllHalls();
                hallNameToId.clear();
                for (Hall hh : refreshed) hallNameToId.put(hh.getName(), hh.getId());
                hallId = hallNameToId.get(a.hallName);
                if (hallId == null) throw new SQLException("Unknown hall name: " + a.hallName);
            }
            ps.setInt(1, planId);
            ps.setInt(2, studentId);
            ps.setInt(3, hallId);
            ps.setInt(4, a.seatNo);
            ps.addBatch();
        }
        int[] results = ps.executeBatch();
        appendLog("✅ Saved " + results.length + " seat assignments to DB (planId=" + planId + ")");
    }
    con.commit();
} catch (Exception ex) {
    appendLog("+ Error saving seating plan: " + ex.getMessage());
    ex.printStackTrace();
    if (con != null) {
        try {
            con.rollback();
            appendLog("🟦Transaction rolled back.");
        } catch (SQLException se) {
            se.printStackTrace();
        }
    }
} finally {
    if (con != null) {
        try { con.setAutoCommit(true); con.close(); } catch (SQLException ignored) {}
    }
}
```

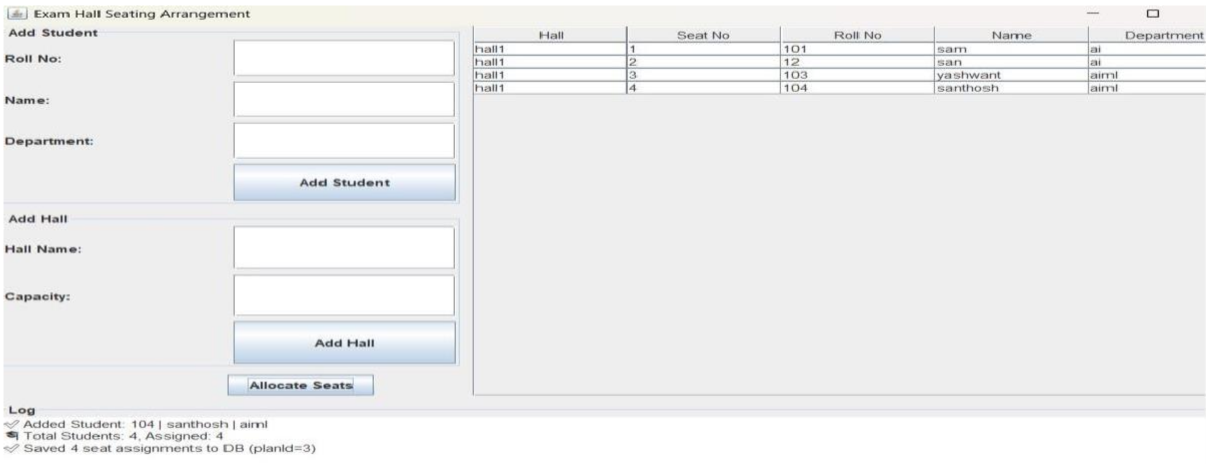
```
    }
    private void appendLog(String s) {
        logArea.append(s + "\n");
    }
    public static void main(String[] args) {
        // Swing UI thread
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                SeatingGUI gui = new SeatingGUI();
                gui.setVisible(true);
            }
        });
    }
}
```

OUTPUT:



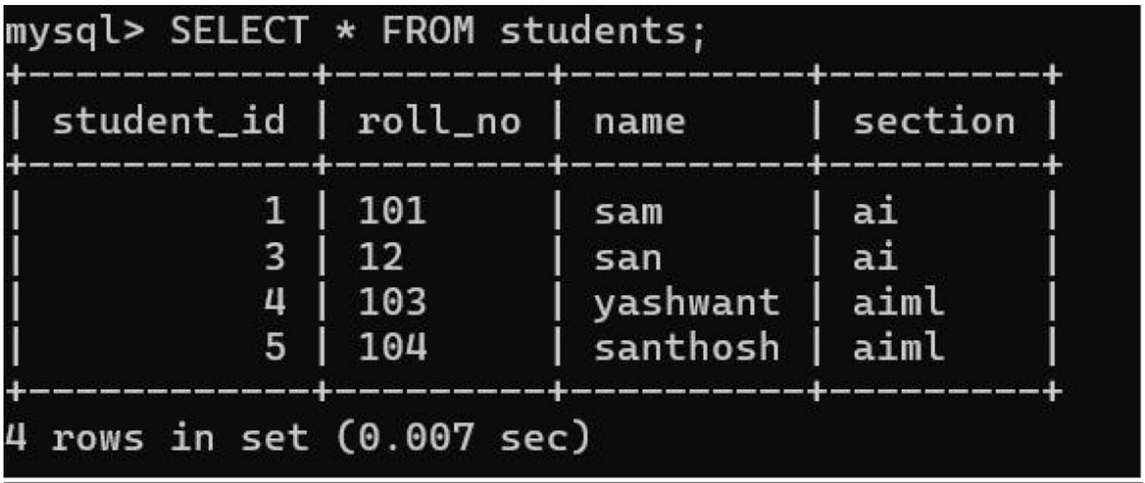


```
✓ Added Student: 104 | santhosh | aiml
```

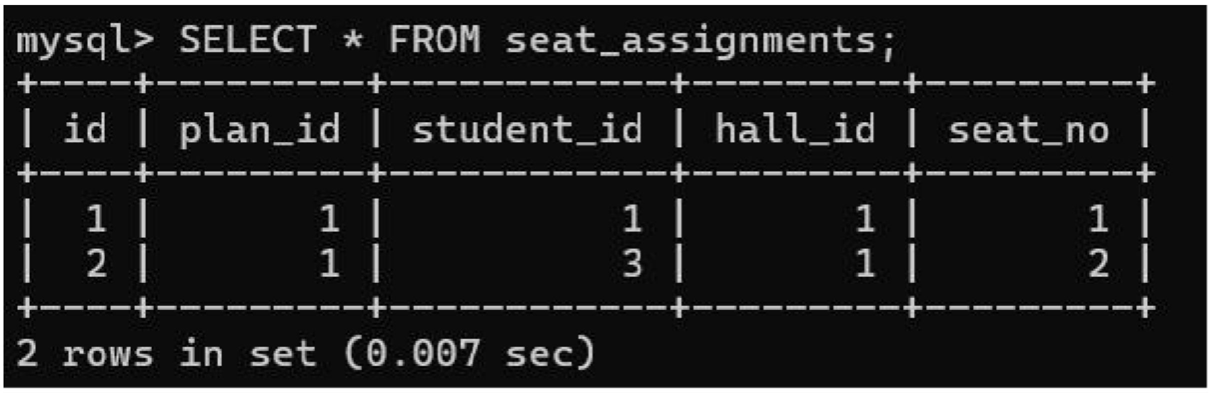


Hall	Seat No	Roll No	Name	Department
hall1	1	101	sam	ai
hall1	2	12	san	ai
hall1	3	103	yashwant	aiml
hall1	4	104	santhosh	aiml

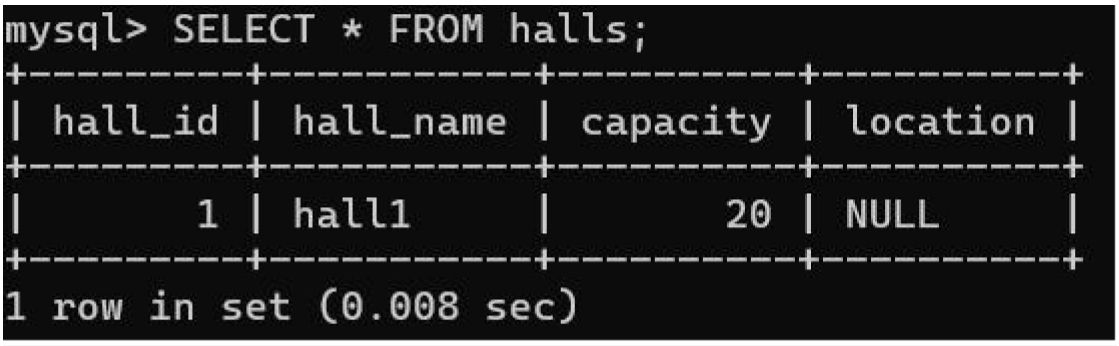
```
✓ Added Student: 104 | santhosh | aiml
Total Students: 4, Assigned: 4
✓ Saved 4 seat assignments to DB (planId=3)
```



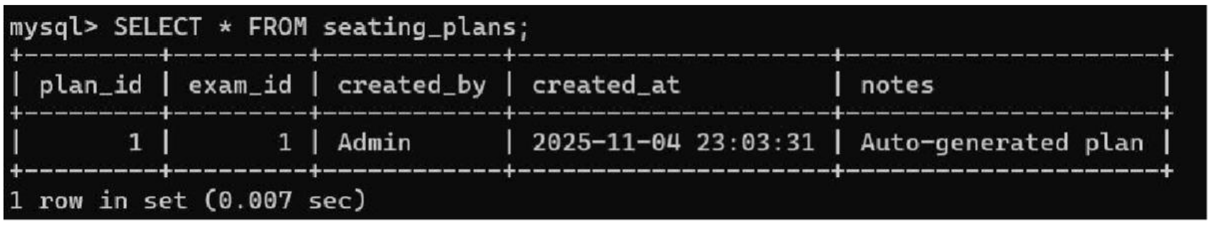
```
mysql> SELECT * FROM students;
+------------+---------+----------+---------+
| student_id | roll_no | name     | section |
+------------+---------+----------+---------+
|          1 | 101     | sam      | ai      |
|          3 | 12      | san      | ai      |
|          4 | 103     | yashwant | aiml    |
|          5 | 104     | santhosh | aiml    |
+------------+---------+----------+---------+
4 rows in set (0.007 sec)
```



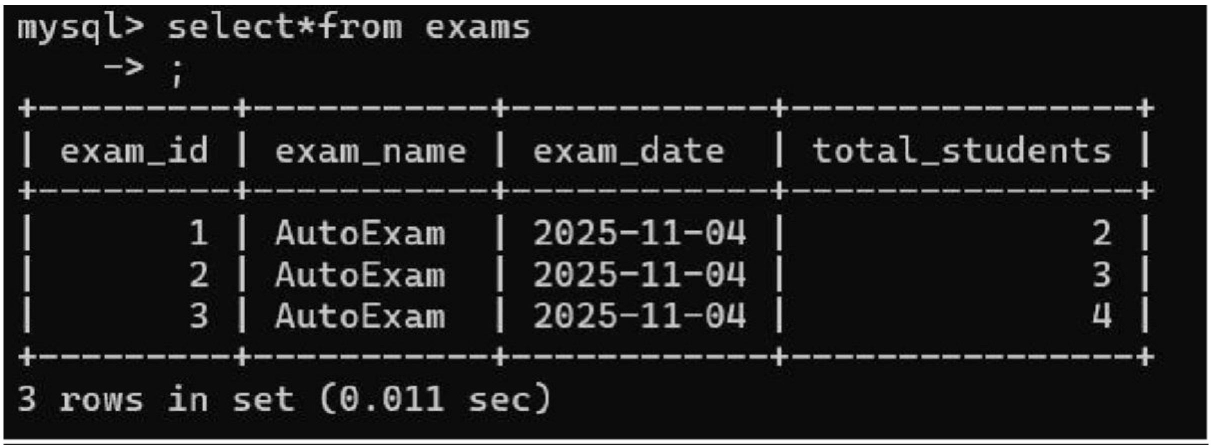
```
mysql> SELECT * FROM seat_assignments;
+----+---------+------------+---------+---------+
| id | plan_id | student_id | hall_id | seat_no |
+----+---------+------------+---------+---------+
|  1 |       1 |          1 |       1 |       1 |
|  2 |       1 |          3 |       1 |       2 |
+----+---------+------------+---------+---------+
2 rows in set (0.007 sec)
```



```
mysql> SELECT * FROM halls;
+---------+-----------+----------+----------+
| hall_id | hall_name | capacity | location |
+---------+-----------+----------+----------+
|       1 | hall1     |       20 | NULL     |
+---------+-----------+----------+----------+
1 row in set (0.008 sec)
```



```
mysql> SELECT * FROM seating_plans;
+---------+---------+------------+---------------------+---------------------+
| plan_id | exam_id | created_by | created_at          | notes               |
+---------+---------+------------+---------------------+---------------------+
|       1 |       1 | Admin      | 2025-11-04 23:03:31 | Auto-generated plan |
+---------+---------+------------+---------------------+---------------------+
1 row in set (0.007 sec)
```



```
mysql> select*from exams
    -> ;
+---------+-----------+------------+----------------+
| exam_id | exam_name | exam_date  | total_students |
+---------+-----------+------------+----------------+
|       1 | AutoExam  | 2025-11-04 |              2 |
|       2 | AutoExam  | 2025-11-04 |              3 |
|       3 | AutoExam  | 2025-11-04 |              4 |
+---------+-----------+------------+----------------+
3 rows in set (0.011 sec)
```

Conclusion & Future Work

Conclusion

The Exam Hall Seating Arrangement System is an efficient and user-friendly application developed to automate the complex task of assigning students to examination halls. The system replaces the traditional manual method with a computerized approach that ensures accuracy, consistency, and time efficiency.

Through this project, the use of Java Swing, Object-Oriented Programming (OOP), and MySQL Database Management has been effectively demonstrated. The modular structure, including separate DAO and GUI components, simplifies maintenance and enhances reusability.
The software successfully performs all essential operations such as adding student and hall details, generating seating arrangements, and storing them securely in a database.

This project not only strengthened the understanding of Java programming concepts, but also provided practical exposure to database connectivity, event handling, and GUI development. It is a complete solution that can be implemented in real-time academic environments for efficient exam management.

Future Work

Although the system achieves its intended objectives, it can be further enhanced with the following improvements:

1. Randomized Allocation:
   Introduce a random or pattern-based allocation algorithm to prevent students of the same department from sitting together.
2. Data Export Feature:
   Add options to export seating plans as PDF or Excel files for easy printing and sharing.
3. Admin Login System:
   Implement a secure login module to restrict unauthorized access to the system.
4. Department-Based Filtering:
   Allow allocation to be done department-wise or semester-wise for specific exams.
5. Online Integration:
   Extend the system into a web-based or cloud-based platform so that data can be accessed remotely by faculty and administrators.
6. Hall Availability Visualization:
   Include a graphical interface to visualize hall occupancy and seat distribution dynamically.
7. Mobile-Friendly Version:
   Develop an Android-based version for on-the-go usage by examination coordinators.

GitHub Reference: https://github.com/santhosh-kr714/oop-miniproject