VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi-590018.

A Project Report on

"ACADEMIC ASSISTANT "

Submitted in the partial fulfillment of the requirements for the award of the

Degree ofBachelor of Engineering

In

Computer Science and Engineering

by

B Kiran
(1OX21CS028)

Under the guidance of
Prof.Angel Donny
Assistant Professor
Department of Computer Science & Engineering

Department of Computer Science and Engineering
THE OXFORD COLLEGE OF ENGINEERING
Bommanahalli, Bnaglaore 560068
2023-2024

THE OXFORD COLLEGE OF ENGINEERING

Hosur Road, Bommanahalli, Bengaluru-560068

(Affiliated to VISVESVARAYA TECHNOLOGICAL UNIVERSITY, Belagavi)

Department of Information Science and Engineering

CERTIFICATE

Certified that the project work entitled **"ACADEMIC ASSISTANT"** carried out by, **B KIRAN (1OX21CS028),** bonafide students of **The Oxford College Of Engineering, Bengaluru** in partial fulfillment for the award of Degree of Bachelor of Engineering in Computer Sceince And Engineering if the **Visvesvaraya Technological University**, Belagavi, during the year **2023-2024**. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Prof.Angel Donny	**Dr.Saravana Kumar**	**Dr. N. Kannan**
Project Guide	Professor & Head of CSE	Principal

1. **Internal Examiner:**_____

2. **External Examiner:**_____

THE OXFORD COLLEGE OF ENGINEERING

Hosur Road, Bommanahalli, Bengaluru-560068

(Affiliated to VISVESVARAYA TECHNOLOGICAL UNIVERSITY, Belagavi)

Department of Computer Science and Engineering

DECLARATION

We the students of Fifth semester B.E, at the Department of Computer Science and Engineering, **The Oxford College Of Engineering, Bengaluru** declare that the project entitled **"ACADEMIC ASSISTANT"** has been carried out by us and submitted in partial fulfillment of the course requirements for the award of degree in Bachelor of Engineering in Computer Science and Engineering discipline of **Visvesvaraya Technological University, Belagavi** during the academic year **2023-2024.** Further, the matter embodied in dissertation hasnot been submitted previously by anybody for the award of any degree or diploma to any other university.

Name	USN	Signature
B KIRAN	**1OX21CS028**	

Date: Place: Bangalore

AKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible whose constant guidance and encouragement crowned our effort with success.

We consider ourselves proud to be a part of The Oxford family, the institution that stood by our way in all our endeavors. We have a great pleasure in expressing our deep sense of gratitude to the founder chairman **late Sri S. Narasa Raju** and to our chairman **Sri S. N. V. L. Narasimha Raju** for providing us with a great infrastructure and well-furnished labs.

We would like to express our gratitude to **Dr. N. Kannan,** Principal, The Oxford College ofEngineering for providing us a congenial environment and surrounding to work in.

Our hearty thanks to **Dr.Saravana Kumar,** Professor & Head, Department of Computer Science and Engineering, The Oxford College of Engineering for his encouragement and support.

Guidance and deadlines play a very important role in successful completion of the project report on time. We convey our gratitude to **Prof.Angel Donny,** Department of Computer Science and Engineering for having constantly monitored the completion of the Project Report and setting up precise deadlines.

We also thank them for their immense support, guidance, specifications and ideas without which the Project Report would have been incomplete. Finally a note of thanks to the Department of Computer Science and Engineering, both teaching and non-teaching staff for their cooperation extended to us.

B KIRAN
(1OX20CS028)

ABSTRACT

The Academic Assistant is a comprehensive web-based solution tailored exclusively for educators, offering a suite of features to streamline academic management tasks. Designed to meet the specific needs of teachers, this platform encompasses functionalities for maintaining student information, generating class timetables, setting examination question papers, and organizing seating arrangements for exams.

The Student Information Management feature provides educators with efficient tools to manage student records, including editing, viewing, and deleting information stored in the database. This feature ensures seamless access to essential student data, facilitating effective communication and personalized academic support.

With the Class Timetable Generation feature, teachers can effortlessly create customized class schedules based on parameters such as the number of days per week, slots per day, start time, duration per slot, subject names, and teacher assignments. This automated process optimizes resource allocation and maximizes instructional time.

The Question Paper Setting feature empowers educators to efficiently design and generate examination question papers using predefined templates. By inputting question details such as date, subject name, subject code, and content, teachers can produce formatted question papers ready for download, ensuring consistency and accuracy in assessment creation.

Facilitating exam administration, the Seating Matrix Generation feature allows teachers to generate detailed seating arrangements by providing parameters such as room numbers and student groupings. The resulting matrices can be easily downloaded, streamlining logistical planning and enhancing exam organization.

In addition to these core features, the Academic Assistant includes fundamental user management functionalities such as signup, signin, signout, and profile management. This ensures secure access to the platform while providing teachers with personalized user experiences tailored to their roles and preferences.

Through the Academic Assistant project, we aim to enhance teacher productivity and administrative efficiency within educational institutions while honing our skills in software development and database management.

TABLE OF CONTENTS

1. INTRODUCTION

In the realm of education, the necessity for streamlined administrative processes tailored to the needs of educators has become increasingly apparent. From managing student information to generating class timetables and setting question papers, the demand for efficient tools has never been greater. In response to this need, we introduce "Academic Assistant," a comprehensive web-based platform designed exclusively for teachers.

"Academic Assistant" serves as a centralized hub for teachers to efficiently manage various aspects of their academic responsibilities. With a focus on simplicity and effectiveness, this platform offers a range of features aimed at enhancing the daily workflow of educators.

The key functionalities of "Academic Assistant" include:

1. **Student Information Management:** Teachers can effortlessly maintain and update student records like personal details. The intuitive interface allows for easy navigation and editing, ensuring that student information remains up-to-date at all times.

2. **Class Timetable Generation:** With the class timetable generation feature, teachers can streamline the process of scheduling classes. By inputting parameters such as the number of days per week, time slots, subjects, and teacher assignments, the system automatically generates a comprehensive timetable tailored to the unique requirements of each academic program.

3. **Question Paper Setting:** Simplifying the task of question paper creation, this feature provides teachers with a predefined template for designing exams. Teachers can input relevant details such as the date, subject name, and question format, enabling quick and efficient generation of customized question papers. Once created, question papers can be easily downloaded and distributed to students.

4. **Seating Matrix Generation:** Facilitating the organization of exams, the seating matrix generation feature enables teachers to generate seating arrangements for classrooms. By specifying parameters such as room number and student groups, the system automatically generates optimized seating plans, ensuring a smooth and orderly examination process. Seating arrangements can be conveniently downloaded and printed for reference.

In addition to these core features, "Academic Assistant" includes essential functionalities such as user authentication (signup, signin, signout), ensuring secure access to the platform. With its user-friendly interface and robust capabilities, "Academic Assistant" empowers teachers to efficiently manage their academic tasks, ultimately enhancing productivity and facilitating a seamless educational experience.

1.1 Preamble

The Academic Assistant project is designed to streamline various tasks essential for teachers, offering features tailored to their needs. It encompasses functions such as maintaining student information, generating class timetables, setting question papers, and arranging seating for exams. Utilizing a robust database management system, it ensures efficient data organization and accessibility, empowering teachers to perform their duties effectively.

1.2 Problem Statement

1. Teachers often encounter challenges in managing student information, including editing, viewing, and deleting records stored in the database.
2. Generating class timetables manually can be time-consuming and prone to errors, especially when considering factors like the number of days per week, slots per day, subject names, and teacher allocations.
3. Crafting question papers involves intricate formatting and content structuring. Without a standardized template and automated system, this process can be cumbersome and inefficient.
4. Arranging seating for exams requires meticulous planning to accommodate different student groups within designated rooms. Without a systematic approach, creating seating matrices can be complex and prone to errors.

1.3 Proposed Solution

To address the challenges outlined above, the Academic Assistant project offers the following solutions:

1. **Student Information Management:** Teachers can easily edit, view, and delete student records stored in the database through a user-friendly interface.

2. **Class Timetable Generation:**By inputting parameters such as the number of days

per week, slots per day, subject names, and teacher allocations, the system automatically generates comprehensive class timetables, saving time and reducing errors.
3. **Question Paper Setting:** Teachers can utilize predefined templates to input questions, dates, subject names, and codes, facilitating the quick creation of structured question papers. The system streamlines the process, ensuring consistency and efficiency.
4. **Seating Matrix Generation:** Teachers can effortlessly generate seating arrangements for exams by providing room numbers and student group details. The system dynamically generates seating matrices, simplifying the task of organizing exam seating arrangements.

Through these features, the Academic Assistant project aims to enhance teacher productivity and efficiency while providing a seamless user experience for managing academic tasks.

2 ANALYSIS AND SYSTEM REQUIREMENTS

2.1 Existing System

In earlier days, managing student information, class timetables, question paper setting, and seating arrangements for exams was done manually, consuming significant time and effort. Teachers had to maintain physical records, create timetables manually, and design question papers from scratch. Additionally, generating seating arrangements for exams involved cumbersome manual processes, often leading to errors and inefficiencies.

To address these challenges, our project, "Academic Assistant," offers a comprehensive solution tailored for teachers. By digitizing these processes into an integrated online platform, our system streamlines administrative tasks, enhances efficiency, and provides valuable insights for educators.

2.2 Hardware and Software Requirements

Hardware Requirements:

A minimum hard disk space of 20 Gigabytes(GB).

Intel core i3 processor or AMD Processor

RAM size of 2GB.

Keyboard.

Mouse.

Software Requirements

Windows operating system such as Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 10

Software: Xampp

Front end: HTML,Bootstrap,Javascript,CSS

Back end: Python,MySQL

3 SYSTEM DESIGN AND MODELLING

3.1Preliminary design

System design is an abstract representation of a system component and their relationship and which describe the aggregated functionally and performance of the system. It is also the overall plan or blueprint for how to obtain answer to the question being asked. The designspecifies various type of approach.

Database design is one of the most important factors to keep in mind if you are concerned with application performance management. By designing your database to be efficient in eachcall it makes and to effectively create rows of data in the database, you can reduce the amount of CPU needed by the server to complete your request, thereby ensuring a faster application.

3.2 ER Diagram

Entity-Relationship Diagram (ER Diagram): The ER diagram illustrates the relationships between data objects within the system. It depicts how entities are connected to each other and the attributes associated with each entity.

Relationships: Data objects within the system are interconnected through various relationships, which define how they are related to each other. These relationships are established based on a set of object relationship pairs.

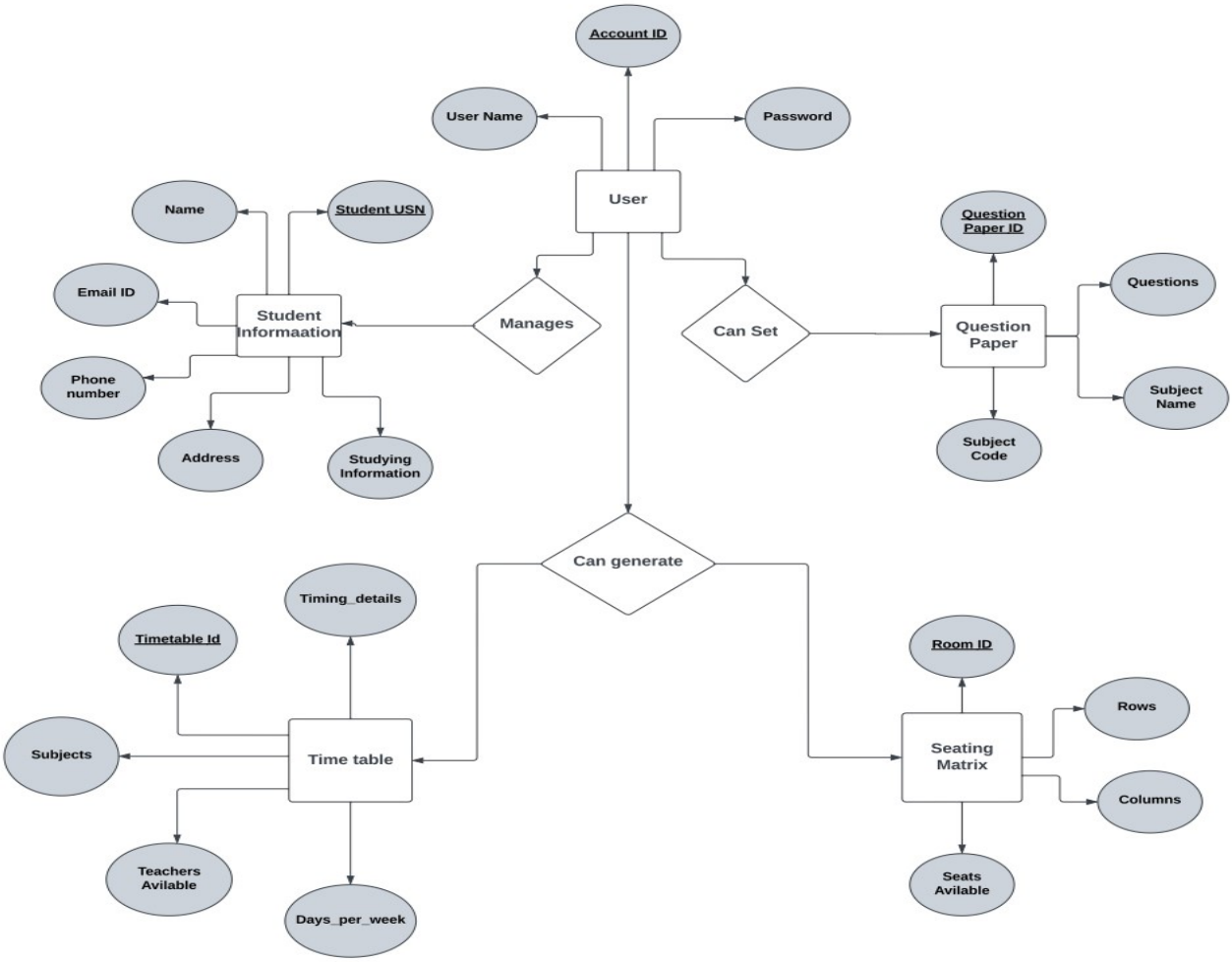
Cardinality Ratio: The cardinality ratio represents the number of objects involved in a particular relationship. It is denoted as I:N, 1:1, N:N, or N:I, indicating the relationship between entities.

The figure below describes the ER diagram of the Academic assistant, which consists of the following entities:

1. **User:** Represents users of the system, storing their unique identifiers, usernames, emails, and hashed passwords.
2. **Student_data:** Stores information about students, including their unique USNs, name,studying_info, contact details, address details.
3. **QP_questions:** Contains details of question papers, including unique paper IDs, codes, dates, subject names, subject codes, and ten questions per paper.
4. **Room_details:** Represents physical rooms for examinations, with attributes for room numbers, rows,columns,seats avilable.
5. **Timetable_data:** Stores details of academic timetables, including timetable_id, subjects, teachers,days per week.

These entities have attributes, including primary, foreign, and composite attributes, with primary attributes being underlined.

ER DIAGRAM


Account ID
User Name
Password
User
Name
Student USN
Email ID
Student Informaation
Phone number
Address
Studying Information
Manages
Can Set
Question Paper ID
Questions
Question Paper
Subject Name
Subject Code
Can generate
Timing_details
Timetable Id
Subjects
Time table
Teachers Avilable
Days_per_week
Room ID
Rows
Seating Matrix
Columns
Seats Avilable

3.3 Schema Diagram

The schema diagram for the Academic Assistant project reflects the database structure tailored for teachers, focusing on maintaining student information, generating class timetables, setting question papers, and arranging seating for exams.

1. Users Table
   - Attributes: user_id (PK), username, email, password.

2. Student Information Table:
   - Attributes: student_usn (PK),student_name, email, phone_number, address,studying details .

3. Timetable Table:
   - Attributes: timetable_id (PK), day_of_week, subject_name, teacher_name, timing_details

4. Question Papers Table:
   - Attributes: paper_id (PK), date, subject_name, subject_code, questions.

5. Seating Arrangement Table:
   - Attributes: arrangement_id (PK), room_number, rows, columns ,seats_assigned.

These tables represent the core functionalities of the Academic Assistant project, allowing teachers to manage student information, generate timetables, set question papers, and arrange seating for exams. The schema emphasizes simplicity and ease of use, ensuring efficient data management for educators.

SCHEMA DIAGRAM

USER

User_Id	Account_Id	Password

STUDENT_INFO

Student_USN	Name	Email Id	Phone No	Address	Studying_Info

QUESTION_PAPER

Question_Paper_Id	Questions	Subject_Name	Subject_Code

TIMETABLE

Timetable_Id	Subjects	Teachers_Avil	Days_Per_Week	Timing_Details

SEATING_MATRIX

Room_Id	Rows	Columns	Seats_Avilable

4. Proposed System

4.1 Operations

1. Maintaining Student Information:
- Allows teachers to insert, edit, view, and delete student information stored in the database. This includes details such as USN, name, date of birth, email, phone number, gender, section, year of studying, semester, year of joining, department, course, address, city, state, and zipcode.

2. Class Timetable Generation:
- Enables teachers to generate class timetables by specifying parameters such as the number of days per week, slots per day, start time, time per slot, subject names, teachers' names, and the number of slots per subject per week. The system then generates a timetable based on these inputs.

3. Question Paper Setting:
- Provides a feature for setting question papers using predefined templates. Teachers can input questions, date, subject name, and subject code, and the system generates a formatted question paper ready for download.

4. Seating Matrix Generation:
- Allows teachers to generate seating arrangements for exams by specifying the room number and different student groups. The system then generates a seating arrangement, which can be easily downloaded.

4.2 SQL Statements

- Insert Statement:
- The `INSERT INTO` statement is used to add new records to the database tables.

Syntax : INSERT INTO table_name VALUES(value1,value2……);

For example:

```sql
INSERT INTO students VALUES (101, 'USN123', 'John', 'Doe', '2000-01-01', 'john.doe@example.com', '1234567890', 'Male', 'A', '1st Year', '1st Semester', '2020', 'Computer Science', 'B.Tech', 'Address', 'Address2', 'City', 'State', '123456');
```

- **Update Statement**:
Syntax:UPDATE table_name SET attribute= value WHERE attribute=value;
- The `UPDATE` statement modifies existing records in a table.

For instance:

```sql
UPDATE teachers SET contact_number = '9876543210' WHERE teacher_id = 201;
```

- Create Statement:

- The `CREATE TABLE` statement is used to create tables in the database.
Syntax : CREATE TABLE table_name (attribute_name type constraints,……..);

Example:

```sql
CREATE TABLE subjects (
    subject_id INT(11) NOT NULL,
    subject_name VARCHAR(50) NOT NULL,
    teacher_id INT(11),
    class_id INT(11),
    slots_per_week INT(11),
    PRIMARY KEY (subject_id)
);
```

5. IMPLEMENTATION

5.1 SQL CODE

```
------------------------------CREATION OF THE TABLE-----------------------------
-- User table

CREATE TABLE User (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(100) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(1000) NOT NULL
);

-- Students table

CREATE TABLE Students (
    student_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    student_usn VARCHAR(50) UNIQUE NOT NULL,
    first_name VARCHAR(200) NOT NULL,
    last_name VARCHAR(200) NOT NULL,
    date_of_birth VARCHAR(1000) NOT NULL,
    email VARCHAR(1000) UNIQUE NOT NULL,
    phone_number VARCHAR(1000) NOT NULL,
    gender VARCHAR(1000) NOT NULL,
    section VARCHAR(1000) NOT NULL,
    year_of_studying VARCHAR(1000) NOT NULL,
    semester VARCHAR(1000) NOT NULL,
    year_of_joining VARCHAR(1000) NOT NULL,
    department VARCHAR(1000) NOT NULL,
    course VARCHAR(1000) NOT NULL,
    address VARCHAR(1000) NOT NULL,
    address2 VARCHAR(1000) NOT NULL,
    city VARCHAR(1000) NOT NULL,
    state VARCHAR(1000) NOT NULL,
    zipcode VARCHAR(1000) NOT NULL
);


-- Questions table

CREATE TABLE Questions (
    paper_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    code VARCHAR(20) UNIQUE NOT NULL,
    date VARCHAR(20) NOT NULL,
    sub_name VARCHAR(40) NOT NULL,
    sub_code VARCHAR(40) NOT NULL,
    question1 VARCHAR(1000) NOT NULL,
```

```
    question2 VARCHAR(1000) NOT NULL,
    question3 VARCHAR(1000) NOT NULL,
    question4 VARCHAR(1000) NOT NULL,
    question5 VARCHAR(1000) NOT NULL,
    question6 VARCHAR(1000) NOT NULL,
    question7 VARCHAR(1000) NOT NULL,
    question8 VARCHAR(1000) NOT NULL,
    question9 VARCHAR(1000) NOT NULL,
    question10 VARCHAR(1000) NOT NULL
);
-- Room table

CREATE TABLE Room (
    room_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    room_no VARCHAR(10) NOT NULL,
    col VARCHAR(10) NOT NULL,
    row VARCHAR(10) NOT NULL,
    seat VARCHAR(10) NOT NULL
);
-- Timetable table

CREATE TABLE Timetable (
    id INTEGER PRIMARY KEY,
    subject_names TEXT,
    teachers_list TEXT,
     slots_per_week TEXT,
    days_in_week INTEGER,
    slots_per_day INTEGER,
    time_per_slot VARCHAR(10),
    start_time VARCHAR(10),
    repetation VARCHAR(10)
);

----------INSERTION OF THE VALUES TO TABLES----------

-- Insert statement for User table

INSERT INTO User (username, email, password) VALUES ('example_user1', 'user1@example.com',
'hashed_password1');
INSERT INTO User (username, email, password) VALUES ('example_user2', 'user2@example.com',
'hashed_password2');
INSERT INTO User (username, email, password) VALUES ('example_user3', 'user3@example.com',
'hashed_password3');


-- Insert statement for Students table

INSERT INTO Students (student_usn, first_name, last_name, date_of_birth, email, phone_number,
gender, section, year_of_studying, semester, year_of_joining, department, course, address, address2,
city, state, zipcode)
```

```
VALUES ('USN001', 'Alice', 'Smith', '2000-05-15', 'alice@example.com', '1234567890', 'Female', 'B', '2', '3', '2019', 'Computer Science', 'B.Tech', '456 Oak St', 'Apt 202', 'City', 'State', '54321');
INSERT INTO Students (student_usn, first_name, last_name, date_of_birth, email, phone_number, gender, section, year_of_studying, semester, year_of_joining, department, course, address, address2, city, state, zipcode)
VALUES ('USN002', 'Bob', 'Johnson', '2001-03-20', 'bob@example.com', '9876543210', 'Male', 'A', '1', '2', '2020', 'Mathematics', 'B.Sc', '789 Elm St', 'Apt 303', 'City', 'State', '67890');
INSERT INTO Students (student_usn, first_name, last_name, date_of_birth, email, phone_number, gender, section, year_of_studying, semester, year_of_joining, department, course, address, address2, city, state, zipcode)
VALUES ('USN003', 'Charlie', 'Brown', '1999-11-10', 'charlie@example.com', '5551234567', 'Male', 'C', '3', '1', '2018', 'Physics', 'M.Sc', '321 Pine St', 'Apt 404', 'City', 'State', '13579');

-- Insert statement for Questions table

INSERT INTO Questions (code, date, sub_name, sub_code, question1, question2, question3, question4, question5, question6, question7, question8, question9, question10)
VALUES ('CODE001', '2024-02-18', 'Mathematics', 'MATH101', 'Question 1A', 'Question 2A', 'Question 3A', 'Question 4A', 'Question 5A', 'Question 6A', 'Question 7A', 'Question 8A', 'Question 9A', 'Question 10A');
INSERT INTO Questions (code, date, sub_name, sub_code, question1, question2, question3, question4, question5, question6, question7, question8, question9, question10)
VALUES ('CODE002', '2024-02-18', 'Physics', 'PHY101', 'Question 1B', 'Question 2B', 'Question 3B', 'Question 4B', 'Question 5B', 'Question 6B', 'Question 7B', 'Question 8B', 'Question 9B', 'Question 10B');
INSERT INTO Questions (code, date, sub_name, sub_code, question1, question2, question3, question4, question5, question6, question7, question8, question9, question10)
VALUES ('CODE003', '2024-02-18', 'Chemistry', 'CHEM101', 'Question 1C', 'Question 2C', 'Question 3C', 'Question 4C', 'Question 5C', 'Question 6C', 'Question 7C', 'Question 8C', 'Question 9C', 'Question 10C');

-- Insert statement for Room table

INSERT INTO Room (room_no, col, row, seat) VALUES ('101', '5', '5', '25');
INSERT INTO Room (room_no, col, row, seat) VALUES ('102', '6', '6', '36');
INSERT INTO Room (room_no, col, row, seat) VALUES ('103', '7', '7', '49');

-- Insert statement for Timetable table
INSERT INTO Timetable (id, subject_names, teachers_list, slots_per_week, days_in_week, slots_per_day, time_per_slot, start_time, repetation)
VALUES (1, 'Math, Science, English', 'Teacher1, Teacher2, Teacher3', '3, 3, 3', 5, 3, '30', '08:00', 'allow');
INSERT INTO Timetable (id, subject_names, teachers_list, slots_per_week, days_in_week, slots_per_day, time_per_slot, start_time, repetation)
VALUES (2, 'Physics, Chemistry, Biology', 'Teacher4, Teacher5, Teacher6', '3, 3, 3', 5, 3, '30', '08:00', 'allow');
INSERT INTO Timetable (id, subject_names, teachers_list, slots_per_week, days_in_week, slots_per_day, time_per_slot, start_time, repetation)
VALUES (3, 'History, Geography, Civics', 'Teacher7, Teacher8, Teacher9', '3, 3, 3', 5, 3, '30', '08:00', 'allow');
```

5.2 PYTHON CODE

```
from flask import Flask, render_template, request, session, redirect, url_for,flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from flask_login import login_user, logout_user,login_required,current_user
from flask_login import LoginManager
from werkzeug.security import generate_password_hash, check_password_hash
import random
from datetime import datetime, timedelta

app = Flask(__name__, static_url_path='/static')
app.secret_key = 'eduaid321'
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:@localhost/hms'
db = SQLAlchemy(app)

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view='login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True,autoincrement=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(1000), nullable=False)

class Students(UserMixin, db.Model):
    student_id = db.Column(db.Integer, primary_key=True,autoincrement=True)
    student_usn = db.Column(db.String(50),unique=True, nullable=False)
    first_name = db.Column(db.String(200), nullable=False)
    last_name = db.Column(db.String(200), nullable=False)
    date_of_birth = db.Column(db.String(1000), nullable=False)
    email = db.Column(db.String(1000),unique=True, nullable=False)
    phone_number = db.Column(db.String(1000), nullable=False)
    gender = db.Column(db.String(1000), nullable=False)
    section = db.Column(db.String(1000), nullable=False)
    year_of_studying = db.Column(db.String(1000), nullable=False)
    semester = db.Column(db.String(1000), nullable=False)
    year_of_joining = db.Column(db.String(1000), nullable=False)
    department = db.Column(db.String(1000), nullable=False)
    course = db.Column(db.String(1000), nullable=False)
    address = db.Column(db.String(1000), nullable=False)
    address2 = db.Column(db.String(1000), nullable=False)
```

```
    city = db.Column(db.String(1000), nullable=False)
    state = db.Column(db.String(1000), nullable=False)
    zipcode = db.Column(db.String(1000), nullable=False)


class Questions(UserMixin, db.Model):
    paper_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    code = db.Column(db.String(20),unique=True, nullable=False)

    date = db.Column(db.String(20) , nullable=False)
    sub_name = db.Column(db.String(40) , nullable=False)
    sub_code = db.Column(db.String(40) , nullable=False)
    question1 = db.Column(db.String(1000) , nullable=False)
    question2 = db.Column(db.String(1000), nullable=False)
    question3 = db.Column(db.String(1000), nullable=False)
    question4 = db.Column(db.String(1000), nullable=False)
    question5 = db.Column(db.String(1000), nullable=False)
    question6 = db.Column(db.String(1000), nullable=False)
    question7 = db.Column(db.String(1000), nullable=False)
    question8 = db.Column(db.String(1000), nullable=False)
    question9 = db.Column(db.String(1000), nullable=False)
    question10 = db.Column(db.String(1000), nullable=False)

class Room(UserMixin,db.Model):
    room_id=db.Column(db.Integer, primary_key=True, autoincrement=True)
    room_no=db.Column(db.String(10), nullable=False)
    col=db.Column(db.String(10)  , nullable=False)
    row=db.Column(db.String(10) , nullable=False)
    seat=db.Column(db.String(10) , nullable=False)
    class Timetable(UserMixin,db.Model):
    id=db.Column(db.Integer, primary_key=True)
    subject_names = db.Column(db.Text)
    teachers_list = db.Column(db.Text)
    slots_per_week = db.Column(db.Text)
    days_in_week = db.Column(db.Integer)
    slots_per_day = db.Column(db.Integer)
    time_per_slot = db.Column(db.String(10))
    start_time=db.Column(db.String(10))
    repetation=db.Column(db.String(10))

@app.route("/")
def home():
   return render_template('index.html')

@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()
```

```
        if user and check_password_hash(user.password, password):
            login_user(user)

            return redirect(url_for('home'))
        else:
            flash( 'Invalid email or password',"danger")
            return render_template('login.html')
    else:
        return render_template('login.html')
@app.route("/signup", methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            flash("Email already exists","danger")
            return render_template('signup.html')
        else:
            new_user = User(username=username, email=email,
password=generate_password_hash(password))
            db.session.add(new_user)
            db.session.commit()
            return redirect(url_for('login'))
    else:
        return render_template('signup.html')

@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

days = ["","Monday", "Tuesday", "Wednesday", "Thursday", "Friday","Saturday"]

@app.route("/saved_schedules", methods=['POST','GET'])
@login_required
def saved_schedule():
    query=Timetable.query.all()
    return render_template('saved_schedules.html',query=query)

@app.route("/schedule", methods=['POST','GET'])
@login_required
def schedule():
    if request.method == 'POST':
        id=request.form['id']
        days_in_week = request.form.get('days_in_week')
        slots_per_day = request.form.get('slots_per_day')
        subject_names = request.form.getlist('subjectName[]')
```

```
        teachers_list = request.form.getlist('teachers[]')
        slots_per_week_list = request.form.getlist('slotsPerWeek[]')
        time_per_slot = request.form.get('time_per_slot')
        start_time = request.form.get('start_time')
        repetation = request.form.get('repetation')
        exists=Timetable.query.filter_by(id=id).first()
        if exists:
            flash("the Schedule already exists give other ID","danger")
            return render_template('schedule.html')

        else:
            subject_names_str = ','.join(subject_names)
            teachers_list_str = ','.join(teachers_list)
            slots_per_week_str = ','.join(slots_per_week_list)
            timetable = Timetable(
                id=id,
            subject_names=subject_names_str,
            teachers_list=teachers_list_str,
            slots_per_week=slots_per_week_str,
            days_in_week=days_in_week,
            slots_per_day=slots_per_day
            ,start_time=start_time,
            time_per_slot=time_per_slot
            ,repetation=repetation
                )
            db.session.add(timetable)
            db.session.commit()
            flash("Schedule saved successfully","success")
            return render_template('schedule.html')
    else:
        return render_template('schedule.html')

@app.route("/delete_timetable/<int:id>", methods=['POST', 'GET'])
def delete_timetable(id):
    try:

        timetable_to_delete = Timetable.query.get(id)
        if timetable_to_delete:
            db.session.delete(timetable_to_delete)
            db.session.commit()
            flash('Saved  data deleted successfully!', 'success')
            return render_template('saved_schedules.html')
        else:
            flash('Student not found!', 'danger')
            return render_template('saved_schedules.html')
    except Exception as e:
        flash(f'Error deleting data: {str(e)}', 'danger')

    return render_template('saved_schedules.html')
```

```
@app.route('/preview_timetable/<int:id>')
@login_required
def preview_timetable(id):
    query=Timetable.query.all()

    timetable_data=Timetable.query.filter_by(id=id).first()
    sub=timetable_data.subject_names.split(',')
    print("timetable_data:", timetable_data)
    print("sub:", sub)
    if(timetable_data.slots_per_day>len(sub)):
        print("1stif")
        if timetable_data.repetation=="allow":
            print("2ndif")
            generated_timetable=repetation(timetable_data)
            print("Generated timetable (with repetation):", generated_timetable)
            time_slots=generate_time_slots(timetable_data.start_time,
timetable_data.time_per_slot,timetable_data.slots_per_day)
            start_time=timetable_data.start_time
            time_per_slot=timetable_data.time_per_slot
            print("Generated time slots:", time_slots)
            print("Start time:", start_time)
            print("Time per slot:", time_per_slot)
            return
render_template('preview_timetable.html',data=generated_timetable,start_time=start_time,time_per_
slot=time_per_slot,time_slots=time_slots,days=days)
        else:
            print("3rdif")
            flash("Without repetation creation of time table is not possible ","danger")
            return render_template('saved_schedules.html',query=query)
    else:
        print("4thif")
        generated_timetable=generate_timetable(timetable_data)
        print("Generated timetable:", generated_timetable)
        time_slots=generate_time_slots(timetable_data.start_time,
timetable_data.time_per_slot,timetable_data.slots_per_day)
        start_time=timetable_data.start_time
        time_per_slot=timetable_data.time_per_slot
        print("Generated time slots:", time_slots)
        print("Start time:", start_time)
        print("Time per slot:", time_per_slot)
        return
render_template('preview_timetable.html',data=generated_timetable,start_time=start_time,time_per_
slot=time_per_slot,time_slots=time_slots,days=days)

def generate_time_slots(start_time, time_per_slot, num_slots_per_day):
    print("slots is working 1")
    # Convert time_per_slot to an integer
    time_per_slot = int(time_per_slot)
```

```
    # Parse start_time
    try:
        start_datetime = datetime.strptime(start_time, '%H:%M')
    except ValueError:
        # Handle the case where start_time is in an invalid format
        return []

    # Calculate end_datetime for each slot and format the time slots
    time_slots = []
    for _ in range(num_slots_per_day):
        end_datetime = start_datetime + timedelta(minutes=time_per_slot)
        time_slot_str = f"{start_datetime.strftime('%I:%M %p')} to
{end_datetime.strftime('%I:%M %p')}"
        time_slots.append(time_slot_str)
        start_datetime = end_datetime
    print("slots is working 2")
    return time_slots


def generate_timetable(timetable_data):
    days = timetable_data.days_in_week
    slots_per_day = timetable_data.slots_per_day
    subjects = timetable_data.subject_names.split(',')
    teachers = timetable_data.teachers_list.split(',')

    if len(subjects) > len(teachers):
        raise ValueError("Insufficient number of teachers for the subjects")

    timetable = {}
    for day in range(1, days + 1):
        timetable[day] = {}
        for slot in range(1, slots_per_day + 1):
            timetable[day][slot] = {'subject': None, 'teacher': None}

    subject_teacher_mapping = dict(zip(subjects, teachers))

    # Track the subjects already assigned to each day
    assigned_subjects_per_day = {day: set() for day in range(1, days + 1)}

    for day in range(1, days + 1):
        for slot in range(1, slots_per_day + 1):
            shuffled_subjects = subjects.copy()
            random.shuffle(shuffled_subjects)
            for subject in shuffled_subjects:
                # Check if the subject has already been assigned to the current day
                if subject not in assigned_subjects_per_day[day]:
                    teacher = subject_teacher_mapping[subject]
                    timetable[day][slot]['subject'] = subject
                    timetable[day][slot]['teacher'] = teacher
                    assigned_subjects_per_day[day].add(subject)
```

```
                    break  # Move to the next slot
return timetable

def repetation(timetable_data):
    days = timetable_data.days_in_week
    slots_per_day = timetable_data.slots_per_day
    subjects = timetable_data.subject_names.split(',')
    teachers = timetable_data.teachers_list.split(',')

     if len(subjects) > len(teachers):
       raise ValueError("Insufficient number of teachers for the subjects")

    timetable = {}
    for day in range(1, days + 1):
        timetable[day] = {}
        for slot in range(1, slots_per_day + 1):
            timetable[day][slot] = {'subject': None, 'teacher': None}
    subject_teacher_mapping = dict(zip(subjects, teachers))
    slots_per_week_list = timetable_data.slots_per_week.split(',')
    if len(slots_per_week_list) != len(subjects):
        raise ValueError("Mismatch in the number of subjects and slots per week")

    for subject, slots_per_week in zip(subjects, slots_per_week_list):
        teacher = subject_teacher_mapping[subject]
        slots_remaining = int(slots_per_week)

            while slots_remaining > 0:
           day = random.randint(1, days)
           slot = random.randint(1, slots_per_day)

           if timetable[day][slot]['subject'] is None:
               timetable[day][slot]['subject'] = subject
               timetable[day][slot]['teacher'] = teacher
               slots_remaining -= 1
    return timetable

@app.route("/student_info")
@login_required
def student_info():
    query=Students.query.all()
    return render_template('student_info.html',query=query)

@app.route("/student_update",methods=['GET', 'POST'])
@login_required
def student_update():
    if request.method == 'POST':
        student_usn = request.form['student_usn']
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        date_of_birth = request.form['date_of_birth']
```

```
        email=request.form['email']
        phone_number=request.form['phone_number']
        gender=request.form['gender']
        section=request.form['section']
        year_of_studying=request.form['year_of_studying']
        semester=request.form['semester']
        year_of_joining=request.form['year_of_joining']
        department=request.form['department']
        course=request.form['course']
        address=request.form['address']
        address2=request.form['address2']
        city=request.form['city']
        state=request.form['state']
        zipcode=request.form['zipcode']
        existing_student = Students.query.filter_by(student_usn=student_usn,email=email).first()
        if existing_student:
            flash("Student already exist","danger")
            return render_template('student_update.html')
        else:
            new_student = Students(student_usn=student_usn,  first_name=first_name,
            last_name=last_name,date_of_birth=date_of_birth, email=email,
            phone_number=phone_number,  gender=gender,  section=section,
            year_of_studying=year_of_studying, semester=semester,  year_of_joining=year_of_joining,
            department=department,  course=course,  address=address,  address2=address2,
            city=city,  state=state,  zipcode=zipcode  )
            db.session.add(new_student)
            db.session.commit()
            flash('Student information updated successfully', 'success')
            return render_template('student_update.html')
    else:
        return render_template('student_update.html')

@app.route("/edit/<string:student_id>",methods=['POST','GET'])
@login_required
def edit_stu(student_id):
    posts=Students.query.filter_by(student_id=student_id).first()
    if request.method == 'POST':
        student_usn = request.form['student_usn']
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        date_of_birth = request.form['date_of_birth']
        email=request.form['email']
        phone_number=request.form['phone_number']
        gender=request.form['gender']
        section=request.form['section']
        year_of_studying=request.form['year_of_studying']
        semester=request.form['semester']
        year_of_joining=request.form['year_of_joining']
        department=request.form['department']
        course=request.form['course']
```

```
        address=request.form['address']
        address2=request.form['address2']
        city=request.form['city']
        state=request.form['state']
        zipcode=request.form['zipcode']
        try:
            student=Students.query.filter_by(student_id=student_id).first()
            student.student_usn = student_usn
            student.first_name = first_name
            student.last_name = last_name
            student.date_of_birth = date_of_birth
            student.email = email
            student.phone_number = phone_number
            student.gender = gender
            student.section = section
            student.year_of_studying = year_of_studying
            student.semester = semester
            student.year_of_joining = year_of_joining
            student.department = department
            student.course = course
            student.address = address
            student.address2 = address2
            student.city = city
            student.state = state
            student.zipcode = zipcode
            db.session.commit()
            flash('Student updated successfully!', 'success')
            return redirect(url_for('student_info'))
        except Exception as e:
            flash(f'Error updating student: {e}', 'danger')
            return render_template('student_edit.html')

    return render_template('student_edit.html',posts=posts)

@app.route("/view/<string:student_id>")
@login_required
def view_stu(student_id):
  student=Students.query.filter_by(student_id=student_id).first()
  return render_template('student_view.html',posts=student)
@app.route("/delete/<int:student_id>", methods=['POST', 'GET'])
def delete_student(student_id):
    try:
        student_to_delete = Students.query.get(student_id)

        if student_to_delete:
            db.session.delete(student_to_delete)
            db.session.commit()
            flash('Student deleted successfully!', 'success')
            return redirect(url_for('student_info'))
        else:
```

```
            flash('Student not found!', 'danger')
            return redirect(url_for('student_info'))

    except Exception as e:

        flash(f'Error deleting student: {str(e)}', 'danger')

    return redirect(url_for('student_info'))

@app.route("/q_paper",methods=['GET','POST'])
@login_required
def q_paper():
    if request.method=='POST':
        code=request.form['code']

        date=request.form['date']
        sub_name=request.form['sub_name']
        sub_code=request.form['sub_code']
        question1=request.form['question1']
        question2=request.form['question2']
        question3=request.form['question3']
        question4=request.form['question4']
        question5=request.form['question5']
        question6=request.form['question6']
        question7=request.form['question7']
        question8=request.form['question8']
        question9=request.form['question9']
        question10=request.form['question10']
        existing_code = Questions.query.filter_by(code=code).first()
        if existing_code:

            flash("Code already exist","danger")
            return redirect(url_for('q_paper'))
        else:
            questions =
Questions(code=code,date=date,sub_name=sub_name,sub_code=sub_code,question1=question1,que
stion2=question2,question3=question3,question4=question4,question5=question5,question6=questio
n6,question7=question7,question8=question8,question9=question9,question10=question10)
            db.session.add(questions)
            db.session.commit()
            flash('Data saved successfully', 'success')
            return redirect(url_for('q_paper'))
    else:

        return render_template('q_paper.html')

@app.route('/saved_qp')
@login_required
def saved_qp():
    query=Questions.query.all()
```

```
    return render_template('saved_qp.html',query=query)

@app.route("/paper_copy/<string:code>")
@login_required
def paper_copy(code):
    paper=Questions.query.filter_by(code=code).first()
    return render_template("paper_copy.html",posts=paper)



@app.route("/seating_matrix")
@login_required
def seating_matrix():
    query=Room.query.all()
    return render_template('seating_matrix.html',query=query)


@app.route("/add_room",methods=['GET', 'POST'])
@login_required
def add_room():
    if request.method == 'POST':
        room_no=request.form['room_no']
        col=request.form['col']
        row=request.form['row']
        seat=request.form['seat']
        existing_room = Room.query.filter_by(room_no=room_no).first()
        if existing_room:
            flash("Room already exist","danger")
            return redirect(url_for('add_room'))
        else:
            new_room=Room(room_no=room_no,row=row,col=col,seat=seat)
            db.session.add(new_room)
            db.session.commit()
            flash('Data saved successfully', 'success')
            return redirect(url_for('add_room'))
    else:

        return render_template('add_room.html')


@app.route("/delete_room/<int:room_no>", methods=['POST', 'GET'])
def delete_room(room_no):
    try:
        room_to_delete = Room.query.get(room_no)
        if room_to_delete:
            db.session.delete(room_to_delete)
            db.session.commit()
            flash('Room deleted successfully!', 'success')
            return redirect(url_for('seating_matrix'))
        else:
```

```
            flash('Room not found!', 'danger')
            return redirect(url_for('seating_matrix'))
    except Exception as e:
        flash(f'Error deleting room: {str(e)}', 'danger')
    return redirect(url_for('seating_matrix'))


@app.route("/generate", methods=['POST', 'GET'])
@login_required
def generate():
    if request.method == 'POST':
        room_no = request.form['room_no']
        missing_l = request.form['missing'].split(',')
        print(missing_l)
        query = Room.query.filter_by(room_no=room_no).first()
        row = int(query.row)
        col = int(query.col)
        seat = int(query.seat)
        student_groups = []
        for key, value in request.form.items():
            if key.startswith('start'):
                start_str = value
                end_str = request.form['end' + key[5:]]
                start_num = int(start_str[-3:])
                end_num = int(end_str[-3:])
                student_group = [start_str[:7] + str(num).zfill(3) for num in range(start_num, end_num +
1)]
                student_groups.append(student_group)
        student_groups_without_missing = []
        for group in student_groups:
            for student in group:
                if student in missing_l:
                    group.remove(student)
            student_groups_without_missing.append(group)
        print("Student Groups:", student_groups)
        print("Student Groups without Missing Values:", student_groups_without_missing)

        total_students = sum(len(group) for group in student_groups_without_missing)
        if total_students > int(seat):
            flash('Students exceed the limit of seats in the class.', 'error')
            return render_template('generate.html')
        seating_arr = generate_seating_arrangement(student_groups_without_missing,  col,row)
        print(seating_arr)
        session['seating_arr'] = seating_arr
        print("Seating Arrangement:")
        return redirect(url_for('seating_view', seating_arr=seating_arr))
    else:
        return render_template('generate.html')
def generate_seating_arrangement(student_groups, row, col):
    temp_matrix = [['] * col for _ in range(row)]
```

```
    group_index = 0
    for i in range(row):
        for j in range(col):
                if group_index >= len(student_groups):
                group_index = 0
            if student_groups[group_index]:
                # Fill the seat if it's available
                if temp_matrix[i][j] == '':
                    temp_matrix[i][j] = student_groups[group_index].pop(0)
            group_index += 1
  for group in student_groups:
        if group:
            # If there are still students left, fill remaining empty seats with them
            for i in range(row):
                for j in range(col):
                    if group and temp_matrix[i][j] == '':
                        temp_matrix[i][j] = group.pop(0)
    return temp_matrix

@app.route('/seating_view',methods=['POST','GET'])
@login_required
def seating_view():
    seating_arr = session.get('seating_arr')
    return render_template('seating_view.html',seating_arr=seating_arr)

@app.route("/profile")
@login_required
def profile():
    return render_template('profile.html',username=current_user.username,email=current_user.email)

if __name__ == "__main__":
    app.run(debug=True)
```

6. TESTING

This gives the outline of all the testing methods that are carried out to get a bug free application. Quality can be achieved by testing the product using different techniques at different phases of the project development.

Testing Process

Testing is a crucial aspect of the Academic Assistant project, ensuring the reliability and quality of the application. It involves creating and executing test cases to identify and rectify errors at different stages of the project development.

Testing Objective

The main objectives of the testing process in the Academic Assistant project are:

- Executing the program with the intent of finding errors.
- Creating test cases with a high probability of discovering undiscovered errors.
- Uncovering undiscovered errors through successful tests.

Levels of Testing

Different levels of testing are employed in the Academic Assistant project, each focusing on different aspects of the system. The main levels include unit testing, integration testing, system testing, and acceptance testing.

6.1 Unit Testing

Unit testing in the Academic Assistant project focuses on verifying the functionality of individual modules. Each module is tested to ensure proper functionality and error-free operation within its boundaries.

Negative Test Case for Student Information Maintenance:

Function Name	Input	Expected Output	Error	Resolved
Edit	Student Information	Invalid data format Should display an error message	Incorrect data format received	Consume()

Positive Test Case for Student Information Maintenance:

Function Name	Input	Expected Output	Error	Resolved
Edit	Student Information	Expected update success message	--	--

6.2 Integration Testing

Integration testing in the Academic Assistant project involves combining tested modules to ensure proper integration and functionality. Errors are identified and addressed during this phase.

Test Case Based on Class Timetable Generation:

Function Name	Input	Expected Output	Error	Resolved
Negative Checking Generate Class Timetable	input parameters	Error In Generation	Output not seen	Consume()
Positive Checking Generate Class Timetable	Valid input parameters	Successful generation of timetable	--	--

6.3 System Testing

System testing in the Academic Assistant project evaluates the entire application against the requirements document. It ensures that all modules and components function correctly and meet the specified requirements.

Test Cases for the Project

Steps	Action	Expected Output
Step 1: Sign Up	The sign-up page appears when a new user accesses the system	User registration form displayed.
Step 2: Sign In	After signing up, the user can sign in to access the system.	Successful login redirects to the user's dashboard. Listed features : 1. Student Database 2. Timetable Generator 3. Question paper Generator 4. Seating Matrix Generator
Step 3: Edit Student Information	The teacher accesses the student information section and edits student details.	Successful update message displayed.
Step 4: Generate Class Timetable	The teacher inputs parameters for generating a class timetable.	Timetable generated successfully.
Step 5: Generate Question Paper	The teacher inputs parameters for generating a Question Paper.	Question Paper generated successfully
Step 6: Generate Class Timetable	The teacher inputs parameters for generating a Seating Matrix .	Seating Matrix generated successfully

.

7. CONCLUSION

The Academic Assistant project, titled "Academic Assistant," has been successfully developed to cater specifically to the needs of teachers. With its comprehensive features, the application facilitates efficient management of student information, class timetables, question paper setting, and seating arrangements for exams.

Key highlights of the project implementation include:

1. **Feature-rich Functionality :** The application encompasses essential features such as maintaining student information, generating class timetables, setting question papers, and creating seating matrices for exams.

2. **User-Friendly Interface:** Designed with teachers in mind, the interface offers ease of use and accessibility, allowing teachers to navigate seamlessly through various functionalities.

3. **Enhanced Efficiency:** By automating tasks such as timetable generation and question paper setting, the application significantly improves efficiency and reduces manual workload for teachers.

4. **Database Integration:** The integration of a robust database ensures that student information and other relevant data are securely stored and easily accessible when needed.

Overall, the Academic Assistant project has successfully addressed the specific requirements of teachers, providing them with a reliable tool to streamline their tasks and enhance productivity in academic settings.

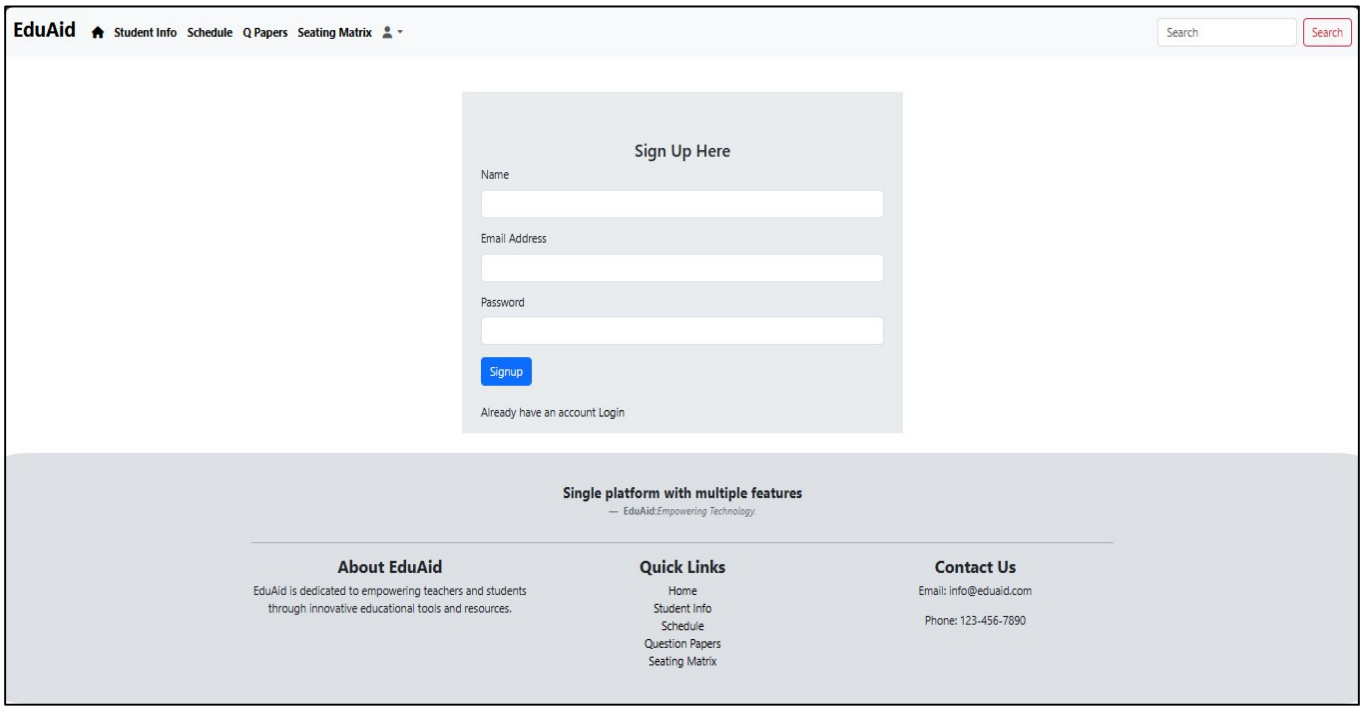
8. REFERENCES

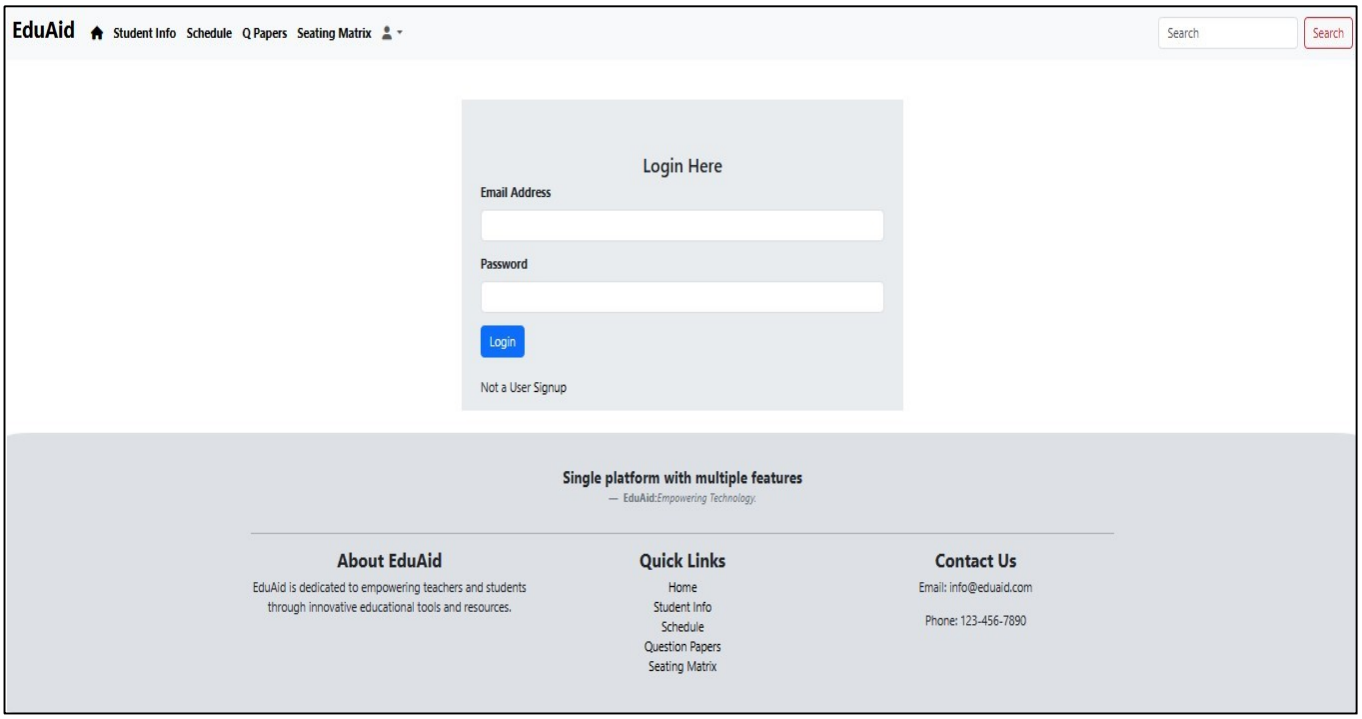
- **Flask Documentation**
- **Google**
- **Youtube**
- **StackOverflow**
- **ARK Pro Coder**
- **W3 Schools**

9. APPENDIX

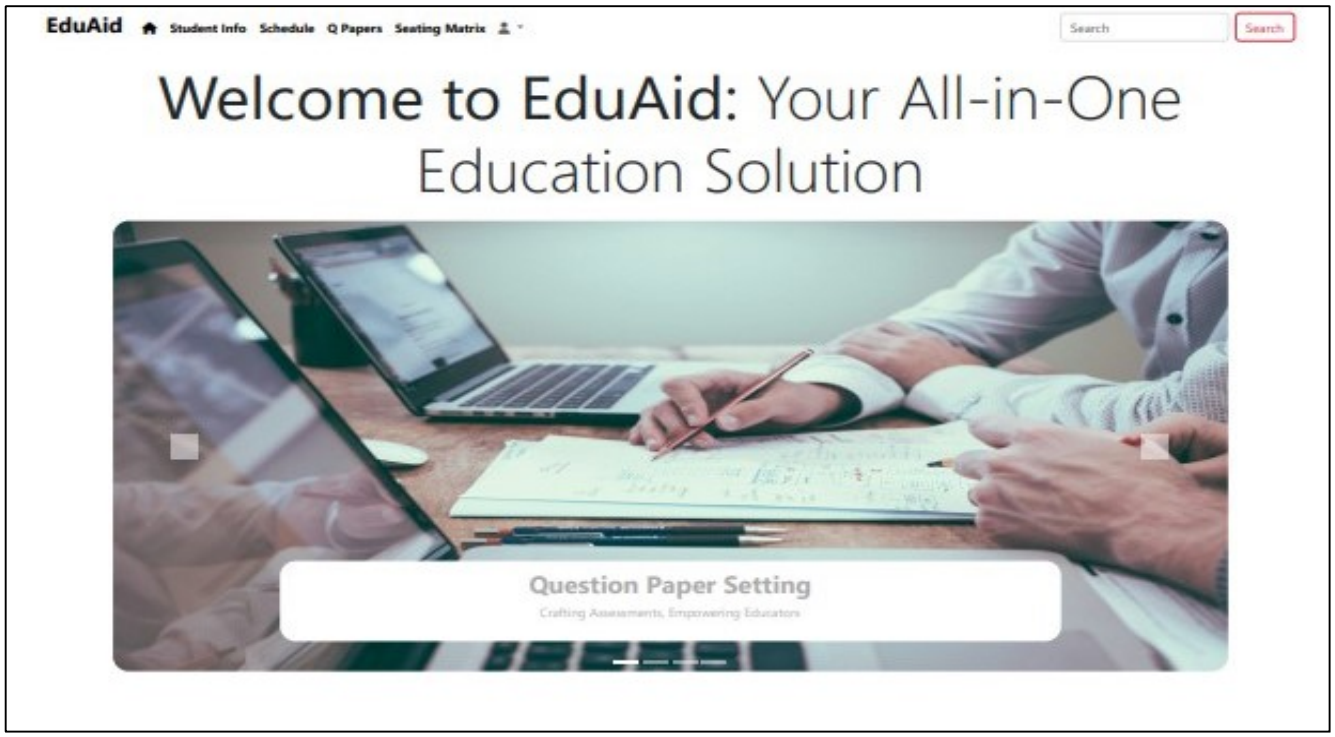
Snapshots

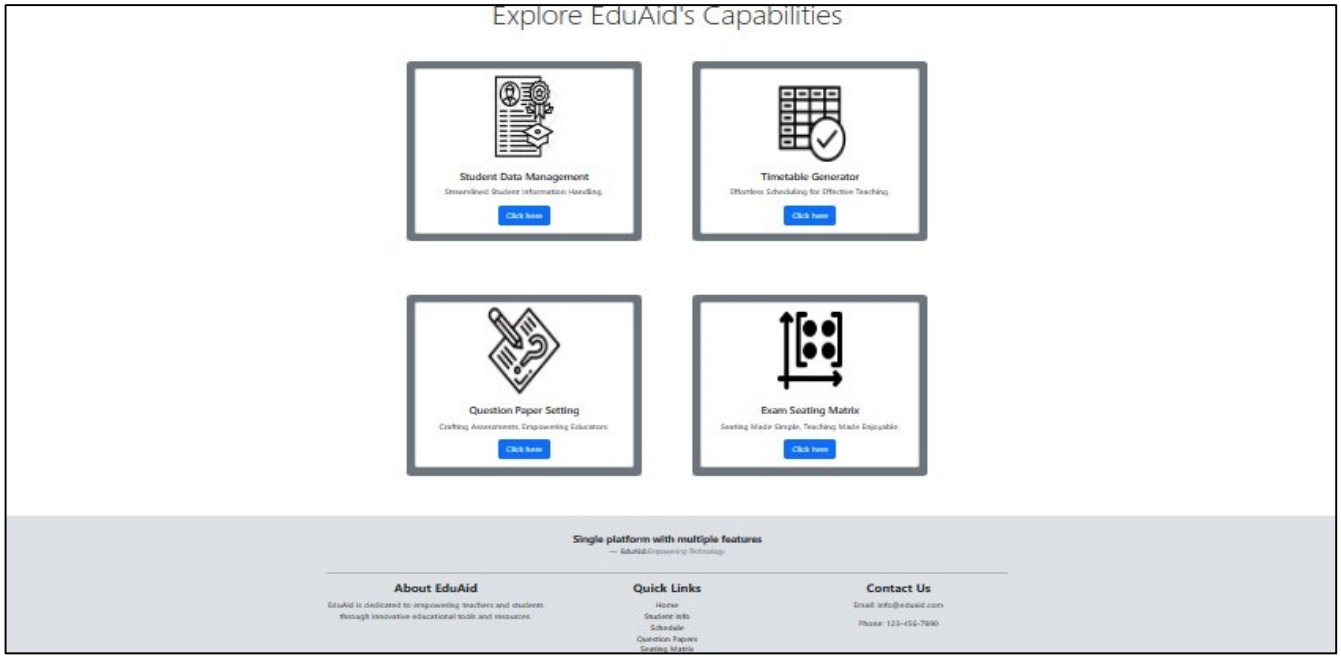
Authentication interface:



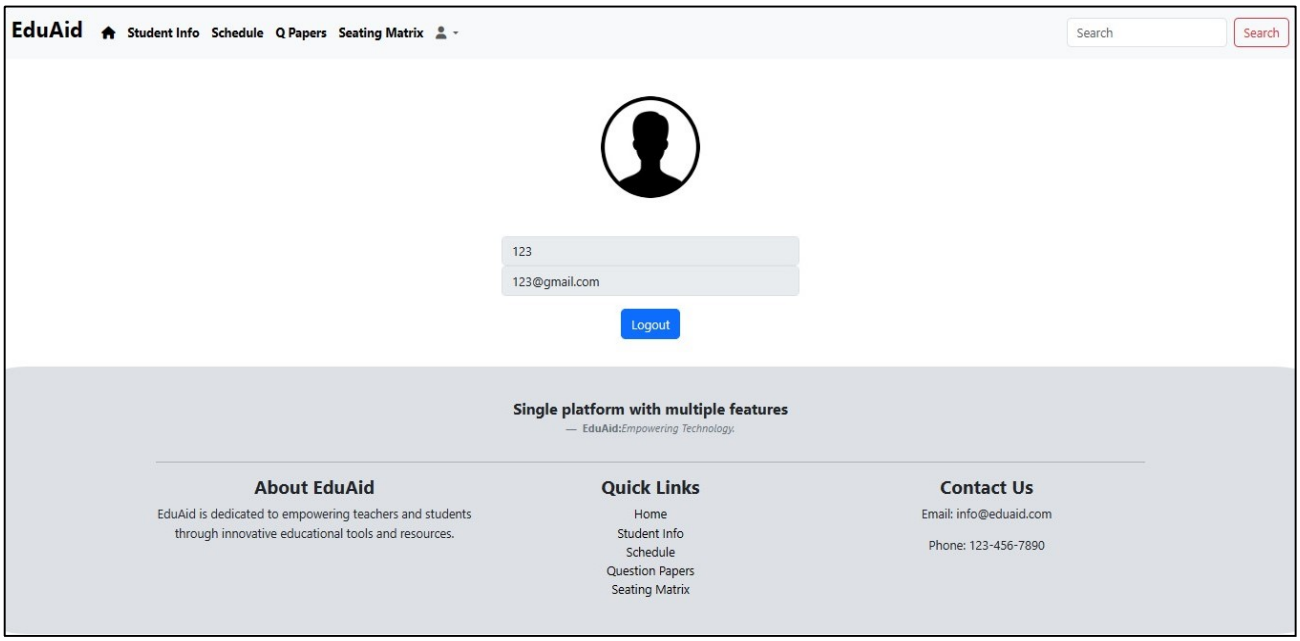


Landing page:

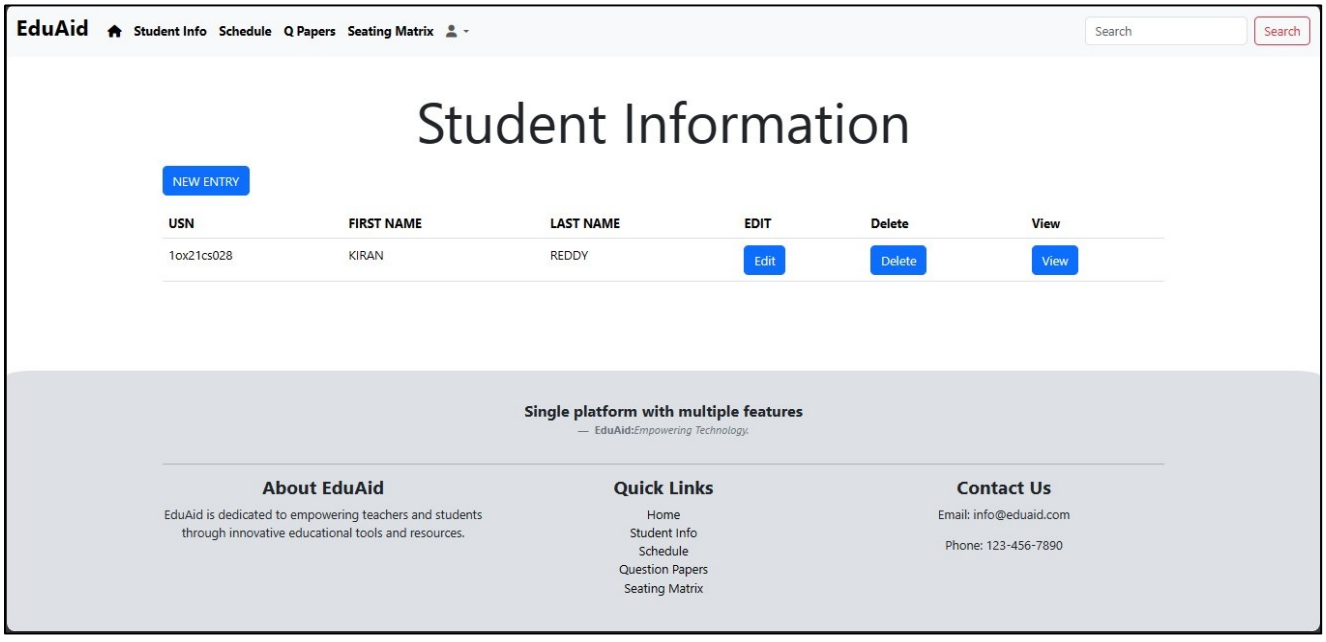




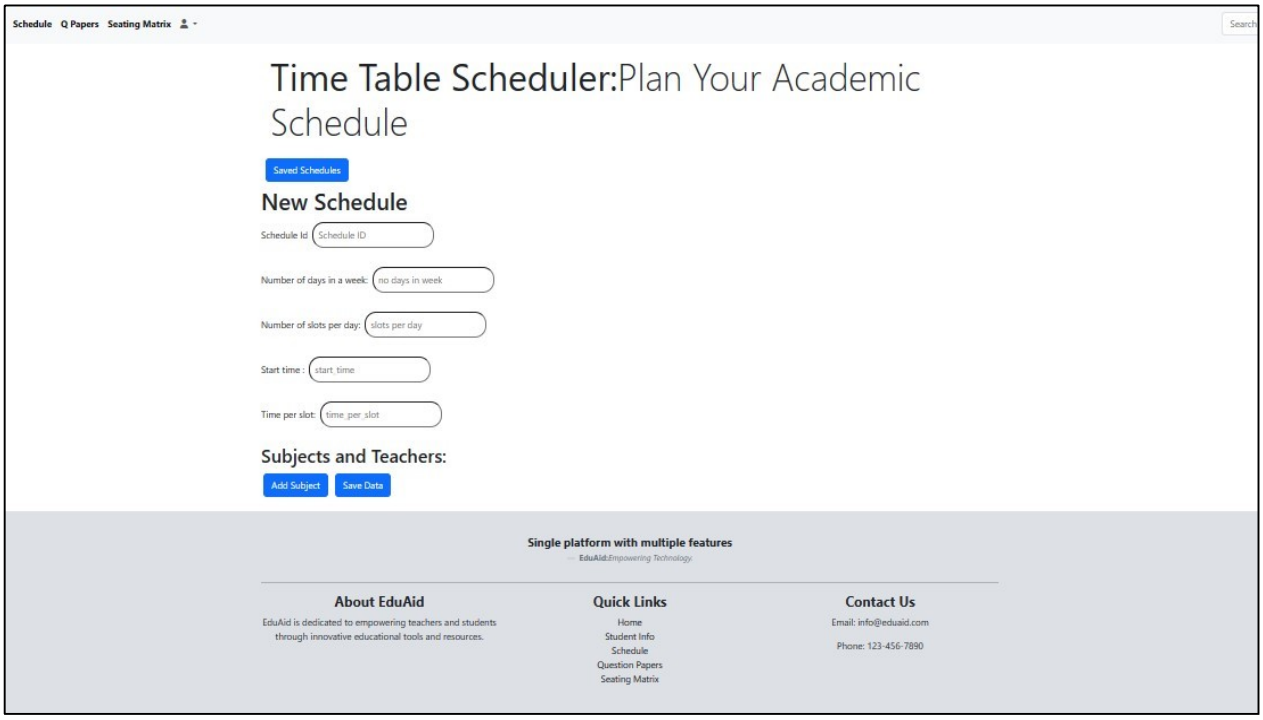
Profile:

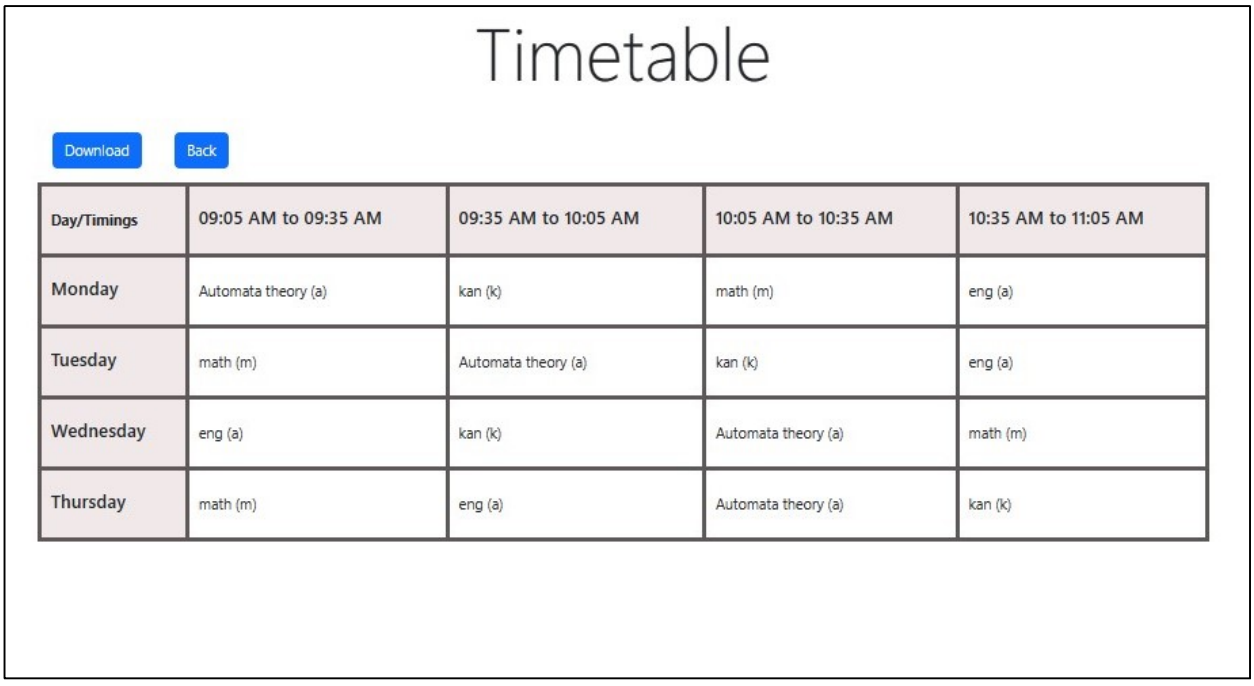


Student Information:



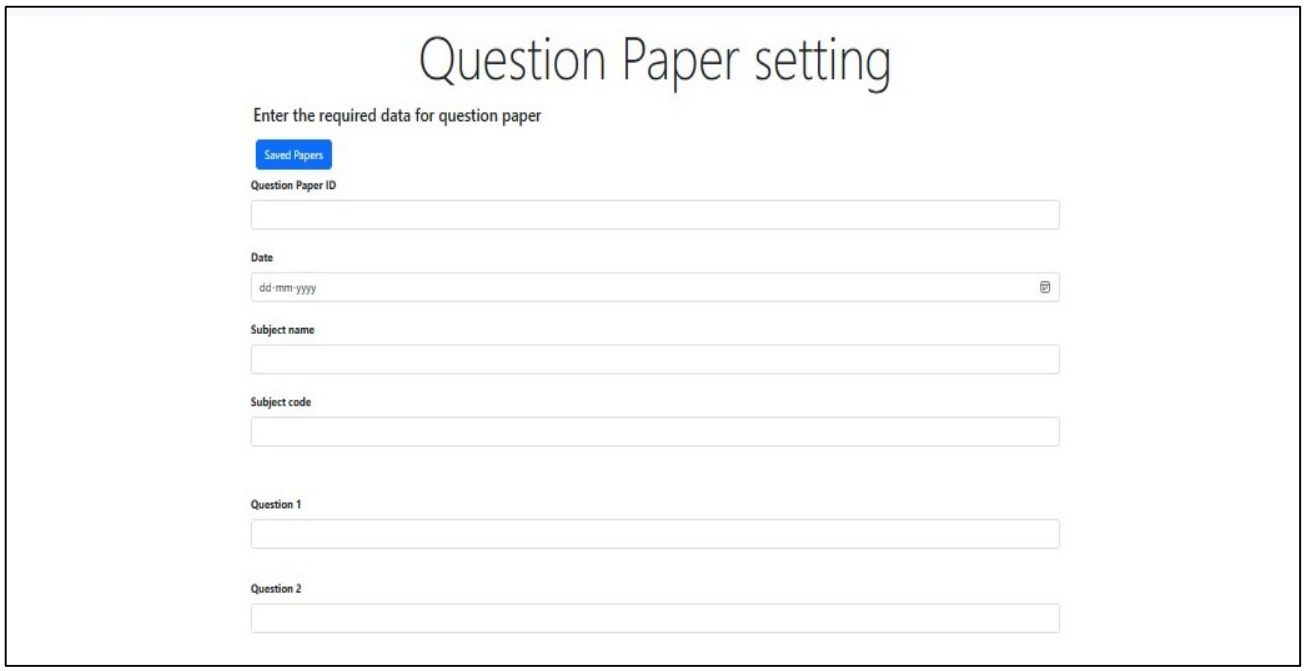
Timetable Generator:

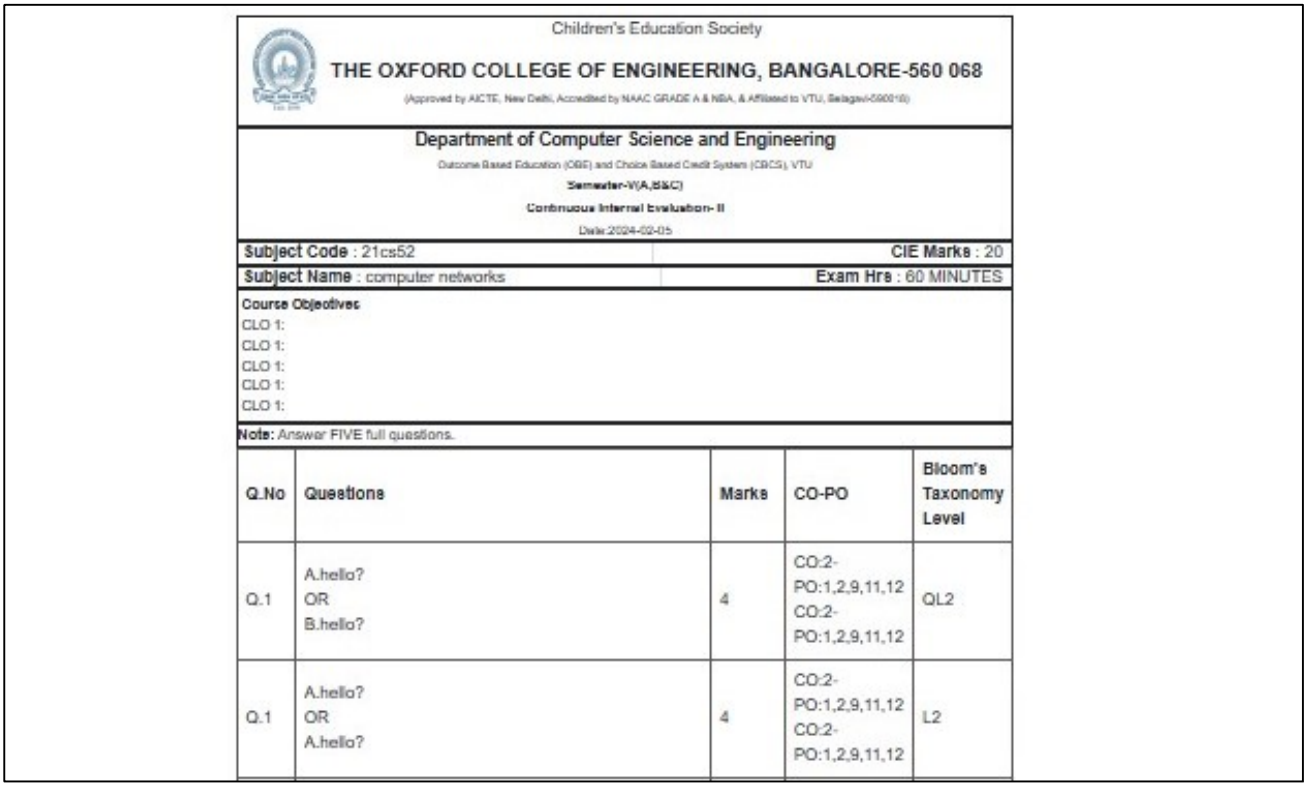




Day/Timings	09:05 AM to 09:35 AM	09:35 AM to 10:05 AM	10:05 AM to 10:35 AM	10:35 AM to 11:05 AM
Monday	Automata theory (a)	kan (k)	math (m)	eng (a)
Tuesday	math (m)	Automata theory (a)	kan (k)	eng (a)
Wednesday	eng (a)	kan (k)	Automata theory (a)	math (m)
Thursday	math (m)	eng (a)	Automata theory (a)	kan (k)

Question Paper generator:





Children's Education Society

THE OXFORD COLLEGE OF ENGINEERING, BANGALORE-560 068

(Approved by AICTE, New Delhi, Accredited by NAAC GRADE A & NBA, & Affiliated to VTU, Belagavi-590018)

Department of Computer Science and Engineering

Outcome Based Education (OBE) and Choice Based Credit System (CBCS), VTU

Semester-V(A,B&C)

Continuous Internal Evaluation- II

Date:2024-02-05

Subject Code : 21cs52 **CIE Marks** : 20

Subject Name : computer networks **Exam Hrs** : 60 MINUTES

Course Objectives:

CLO 1:
CLO 1:
CLO 1:
CLO 1:
CLO 1:

Note: Answer FIVE full questions.

Q.No	Questions	Marks	CO-PO	Bloom's Taxonomy Level
Q.1	A.hello? OR B.hello?	4	CO:2-PO:1,2,9,11,12 CO:2-PO:1,2,9,11,12	QL2
Q.1	A.hello? OR A.hello?	4	CO:2-PO:1,2,9,11,12 CO:2-PO:1,2,9,11,12	L2

Seating Matrix generator: