Report

Complex Engineering Project

CS-115: Computer Programming

Submitted by:

Usman Rasheed Siddiqui	CS-24038
Huzaifa Hanif	CS-24039
Muhammad Ahmed Qazi	CS-24045

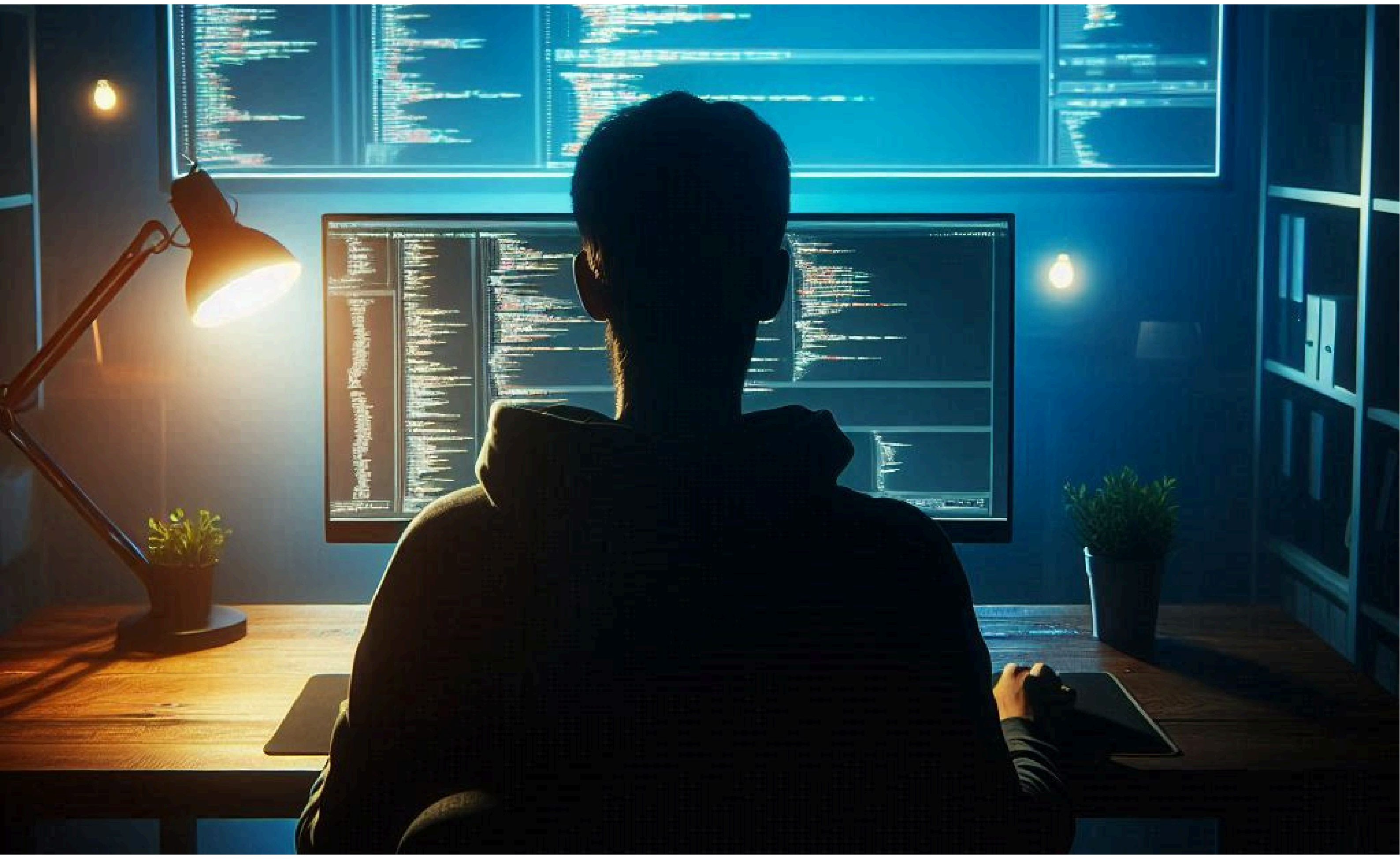
Submitted to:

Dr. Maria Waqas

Ms. Mehwish Raza

Practical Group:

G2



Date of submission: Monday, 25/11/2024

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-115
Course Title: Computer Programming
Complex Engineering Problem
FE Batch 2024, Fall Semester 2024
Grading Rubric
TERM PROJECT

Group Members:

Student No.	Name	Roll No.
S1	Muhammad Ahmed Qazi	CS-24045
S2	Usman Rasheed Siddiqui	CS-24038
S3	Huzaifa Hanif	CS-24039

CRITERIA AND SCALES				**Marks Obtained**		
				S1	**S2**	**S3**
Criterion 1: Does the application meet the desired specifications and produce the desired outputs? (CPA-1, CPA-3) **[8 marks]**						
1	2	3	4			
The application does not meet the desired specifications and is producing incorrect outputs.	The application partially meets the desired specifications and is producing incorrect or partially correct outputs.	The application meets the desired specifications but is producing incorrect or partially correct outputs.	The application meets all the desired specifications and is producing correct outputs.			
Criterion 2: How well is the code organization? **[2 marks]**						
1	2	3	4			
The code is poorly organized and very difficult to read.	The code is readable only to someone who knows what it is supposed to be doing.	Some part of the code is well organized, while some part is difficult to follow.	The code is well organized and very easy to follow.			
Criterion 3: How friendly is the application interface? (CPA-1, CPA-3) **[2 marks]**						
1	2	3	4			
The application interface is difficult to understand and use.	The application interface is easy to understand and but not that comfortable to use.	The application interface is very easy to understand and use.	The application interface is very interesting/ innovative and easy to understand and use.			
Criterion 4: How does the student performed individually and as a team member? (CPA-2, CPA-3) **[4 marks]**						
1	2	3	4			
The student did not work on the assigned task.	The student worked on the assigned task, and accomplished goals partially.	The student worked on the assigned task, and accomplished goals satisfactorily.	The student worked on the assigned task, and accomplished goals beyond expectations.			
Criterion 5: Does the report adhere to the given format and requirements? **[4 marks]**						
1	2	3	4			
The report does not contain the required information and is formatted poorly.	The report contains the required information only partially but is formatted well.	The report contains all the required information but is formatted poorly.	The report contains all the required information and completely adheres to the given format.			
			Total Marks:			

Teacher's Signature

Table of Contents

1. Introduction

1.1 Problem Description

The objective of this project is to develop a simplified database management system (DBMS) using Python. A system that allows users to define custom databases with variable fields and associated field lengths. The DBMS will enable users to add, delete, edit, view and search records within the created databases.

The system must allow to:

- ☑ Create custom databases with variable fields and field lengths.
- ☑ Perform essential database operations such as adding, deleting, editing, viewing, and searching records.
- ☑ Utilise a user-friendly interface to facilitate interaction with the system.

1.2 Project Scope

This project demonstrates the application of programming concepts learned during the course. It emphasises:

- ➔ Modular programming
- ➔ File handling for database storage
- ➔ Web development using Flask
- ➔ Responsiveness and dynamic user interaction using JavaScript and Bootstrap

2. Distinguishing Features of the Project

Key features that distinguish this project include:

- ➔ **Web-Based Interface**: Transitioned from CLI to a web-based GUI using Flask and Bootstrap, enhancing usability.
- ➔ **Dynamic Table and Field Handling**: Users can dynamically create tables and define fields with constraints (e.g., max length).
- ➔ **Responsive Design**: Built with Bootstrap for a seamless experience across devices.
- ➔ **AJAX Integration**: Utilised AJAX for asynchronous data updates without page reloads.
- ➔ **Extensibility**: Designed for easy future upgrades, including authentication and data export features.

3. System Design and Architecture

The Simple Database Management System (DBMS) is designed with a modular and extensible architecture, ensuring ease of development, scalability, and adaptability to future requirements. The system is divided into three main components: **Frontend Interface**, **Backend Logic**, and **Database Handling**.

3.1 Design Overview

➔ **Frontend Interface**:
  - The user interface is built with HTML, CSS, and Bootstrap for responsive design.
  - Forms and modals enable user interactions such as adding tables, creating records, and performing database operations.
  - AJAX is utilised for dynamic content updates, ensuring seamless interaction without full-page reloads.

➔ **Backend Logic**:
  - The backend is implemented using Flask, a lightweight Python web framework.
  - Flask routes requests from the frontend, handles processing logic, and renders the appropriate HTML templates.

➔ **Database Handling**:
  - Python's file-handling capabilities are used to store structured data in text files.
  - Each table is saved as a separate file, with field names and constraints (like maximum length) defined during database creation.

3.2 Workflow of the Application

The system follows a request-response cycle managed by Flask:

➔ **User Request**:
  - Users interact with the GUI through forms and buttons.
  - Requests (e.g., adding a record, searching a table) are sent to specific Flask routes.

➔ **Flask Route Processing**:
  - Flask processes the request and performs the corresponding operation.
  - Operations include creating databases, editing tables, and retrieving records.

➔ **Response Rendering**:
  - Flask renders updated HTML templates or returns HTML responses (for AJAX calls).
  - The frontend dynamically updates the content displayed to the user.

4. Technical Implementation

This section outlines the technical details of how the core functionalities of the Simple Database Management System (DBMS) are implemented, focusing on database setup, data manipulation, and Flask integration.

4.1 Database Setup

The database setup allows for dynamic creation of databases, tables, and fields:

- ➔ **Dynamic Form Generation**: JavaScript is used to dynamically generate forms based on user inputs for table names and field definitions.
- ➔ **Field Length Constraints**: Each field in a table is assigned a maximum length, which is validated both client-side (JavaScript) and server-side (Flask) during data entry.
- ➔ **Data Storage**: Each table is saved as a separate text file, and the fields are serialised to ensure easy data retrieval and manipulation.

4.2 Flask Integration

Flask is used to manage the web application's backend logic:

- ➔ **Routes**: Routes are defined for creating a database (/create_db), opening an existing database (/opendb/<db_name>), and handling various operations (e.g., adding records, searching).
- ➔ **Template Rendering**: Flask uses Jinja2 to render HTML templates dynamically. For instance, after retrieving data from a database, the records are injected into a table within the HTML.
- ➔ **Form Handling**: Forms are handled by Flask's request object. Data is validated and processed, with appropriate error handling in case of invalid inputs.
- ➔ **AJAX Integration**: Flask responds to AJAX requests asynchronously, allowing dynamic updates of tables and records without page reloads.

4.3 Technologies Used

- ➔ **Flask**: A Python-based web framework for handling routing and dynamic template rendering.
- ➔ **JavaScript**: For client-side interactivity, including dynamic form generation and AJAX requests.
- ➔ **Bootstrap 5**: For responsive frontend design and form styling.
- ➔ **Python File Handling**: To store database records in text files, allowing for easy reading and writing of data.

5. Challenges and New Learnings

During the project, several challenges were faced and valuable lessons were learned:

- ➔ **Dynamic Form Generation**: Creating forms dynamically based on user input required mastering JavaScript and ensuring seamless interaction with Flask.
- ➔ **Data Validation and Integrity**: Implementing both client-side and server-side validation for field lengths and data integrity was complex.
- ➔ **File Handling and Storage**: Storing and managing two separate folders (in our case) could lead to confusion and corruption if not handled correctly, especially during editing records.
- ➔ **AJAX Integration**: Integrating AJAX with Flask for asynchronous updates was challenging but improved the user experience significantly.
- ➔ **User-intuitive Design**: Designing a responsive interface using Bootstrap for multiple device sizes required thorough testing instead of a Command Line Interface (CLI) which can be cumbersome to use for non-technical users.

6. Individual Contributions

CS-24045 - Muhammad Ahmed Qazi

- ➔ Led the development of the Flask-based web interface, including the integration of AJAX for dynamic interactions.
- ➔ Implemented the core database handling functions, such as creating and managing databases and tables, along with the backend logic for data manipulation (adding, deleting, and editing records).
- ➔ Designed and implemented the overall structure of the project and handled the deployment for cross-platform compatibility.

CS-24038 - Usman Rasheed Siddiqui

- ➔ Developed the functions for displaying and editing records within the database.
- ➔ Worked on ensuring the system accurately renders the records and allows users to make modifications efficiently.

CS-24039 - Huzaifa Hanif

- ➔ Implemented the search functionality to retrieve records based on specific queries.
- ➔ Contributed to the project documentation, including writing key sections of the report and ensuring proper formatting and organisation of the project details.

7. Potential Improvements for the Future

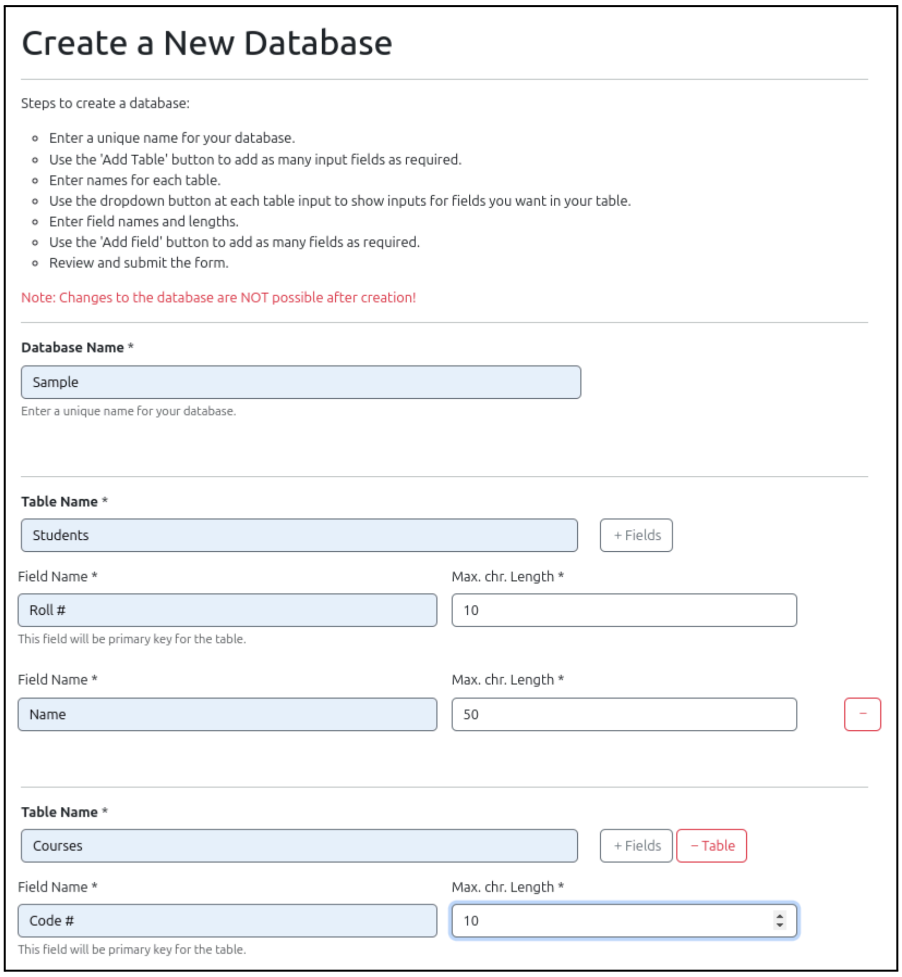
Several improvements can enhance the functionality, security, and user experience of the DBMS:

- ➔ **Public Accessibility**:
    - Host the DBMS online by obtaining a domain and integrating it with a reliable hosting service.
    - Optimise for seamless public access while maintaining stable performance.
- ➔ **Enhanced Graphical User Interface (GUI)**:
    - Revamp the GUI to make it more intuitive and visually appealing.
    - Consider using modern frameworks like PyQt or Tkinter for desktop solutions, or enhance the web interface with Flask and Bootstrap for a responsive design.
- ➔ **Robust Security Features**:
    - Implement encryption for sensitive data (e.g., passwords, data transmissions).
    - Add role-based access control (RBAC) to prevent unauthorised access.
    - Introduce audit logs to track user activity for enhanced security.
- ➔ **Role-Based Viewing Modes**:
    - Create viewing modes for different user roles (Admin, User, Viewer) with tailored permissions.
    - Ensure flexibility for each role: Admins can manage users, Users can edit records, and Viewers have read-only access.
- ➔ **Sign-In Functionality**:
    - Implement a secure login system for personalised access to the database.
    - Allow users to manage their databases remotely through their accounts.
    - Consider integrating OAuth or other third-party authentication services for ease of use.
- ➔ **Data Visualisation with Graphs**:
    - Integrate charting libraries (e.g., Matplotlib, Plotly) to visually represent database reports.
    - Provide interactive dashboards displaying key metrics, trends, and insights from the database.

8. Test-cases

Test case 1: Database Creation

- ☑ **Objective**: Verify that the system allows users to create a database with tables and fields.



Create a New Database

Steps to create a database:

- Enter a unique name for your database.
- Use the 'Add Table' button to add as many input fields as required.
- Enter names for each table.
- Use the dropdown button at each table input to show inputs for fields you want in your table.
- Enter field names and lengths.
- Use the 'Add field' button to add as many fields as required.
- Review and submit the form.

Note: Changes to the database are NOT possible after creation!

Database Name *: Sample
Enter a unique name for your database.

Table Name *: Students

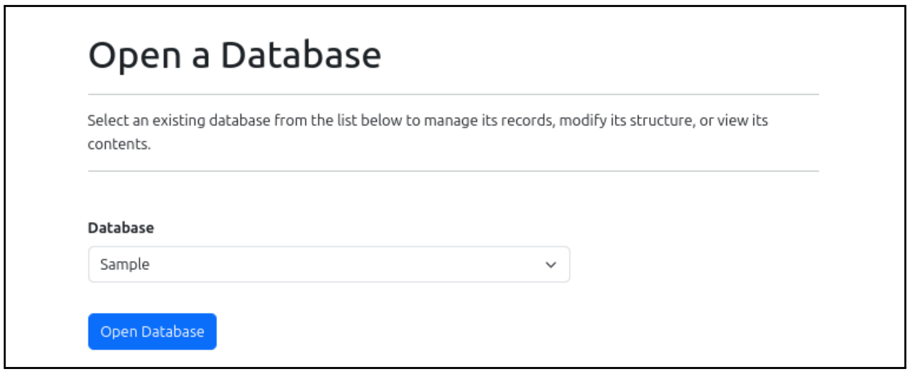
Field Name *: Roll #
Max. chr. Length *: 10
This field will be primary key for the table.

Field Name *: Name
Max. chr. Length *: 50

Table Name *: Courses

Field Name *: Code #
Max. chr. Length *: 10
This field will be primary key for the table.

Figure 1: **Create Database** page with the form for defining table names and fields.



Open a Database

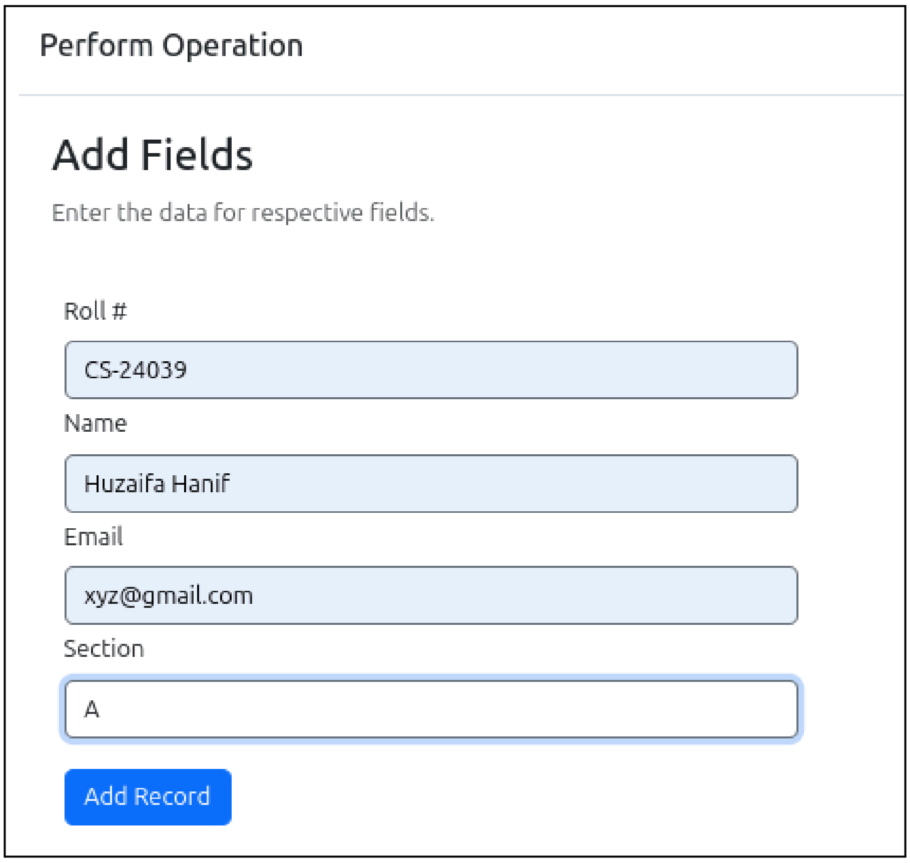
Select an existing database from the list below to manage its records, modify its structure, or view its contents.

Database: Sample

Figure 2: **List of databases** in the DBMS.

Test Case 2: Add and Display Records

- ☑ **Objective**: Ensure that records can be added to a table and displayed correctly.



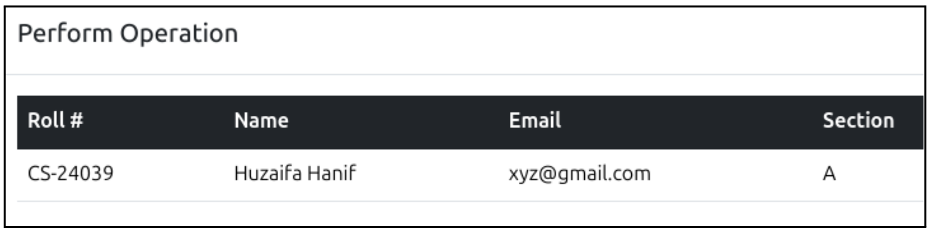
Perform Operation

Add Fields

Enter the data for respective fields.

Roll #	CS-24039
Name	Huzaifa Hanif
Email	xyz@gmail.com
Section	A

Figure 3: **Add Record** form, showing fields being filled in (add a sample record).



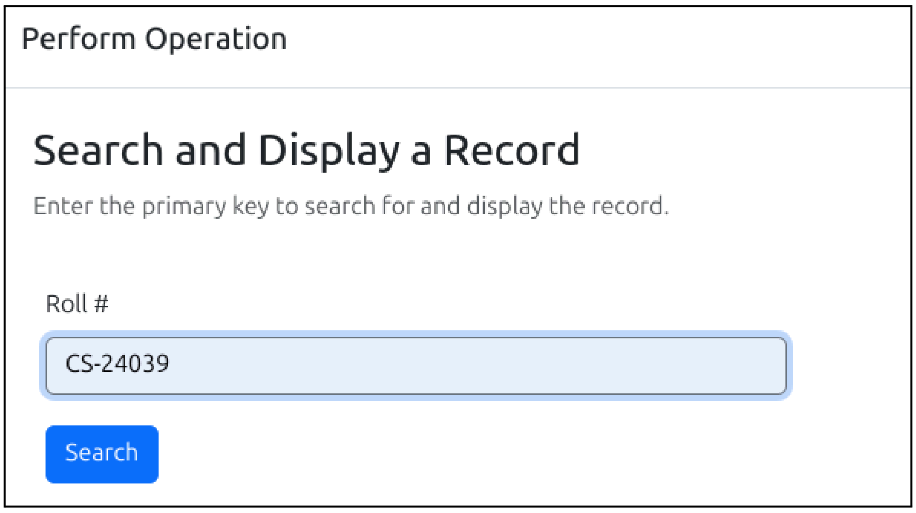
Perform Operation

Roll #	Name	Email	Section
CS-24039	Huzaifa Hanif	xyz@gmail.com	A

Figure 4: **Table view** showing the newly added record.

Test Case 3: Search Record

☑ **Objective**: Verify that the search functionality correctly locates and displays records.



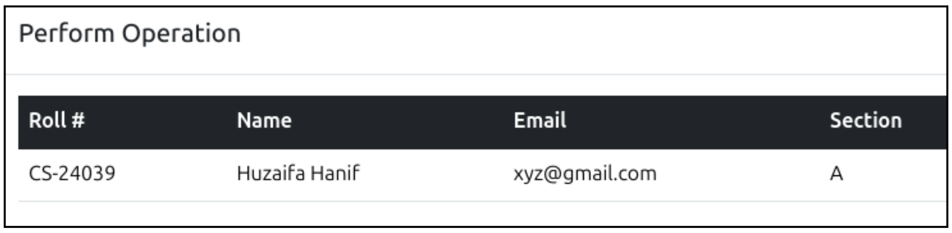
Perform Operation

Search and Display a Record

Enter the primary key to search for and display the record.

Roll #: CS-24039

Figure 5: **Search Record** form with the search criteria entered (e.g., searching for a record by primary key or field value).



Perform Operation

Roll #	Name	Email	Section
CS-24039	Huzaifa Hanif	xyz@gmail.com	A

Figure 6: **Search results** that show the matching record(s) in the table.

9. List of references

Python Documentation
Python Software Foundation. "Python 3 Documentation." Python.org, https://docs.python.org/3/.

Flask Documentation
Pallets Projects. "Flask Documentation." Flask.palletsprojects.com, https://flask.palletsprojects.com/en/stable/

JavaScript Documentation
Mozilla Developer Network (MDN). "JavaScript Guide." MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide.

Bootstrap Documentation
Bootstrap. "Bootstrap 5 Documentation." GetBootstrap.com, https://getbootstrap.com/docs/5.3/getting-started/introduction/

Scan to Access the Repository
Containing All Code, Files,
and Documentation