

# COMP3030 - Final Summary Report

Vu Ai Thanh

Truong Gia Bao

Nguyen Tuan Anh

25-05-2025

## Table of contents

<b>1</b>	<b>Conceptual &amp; Physical Design</b>	<b>1</b>
1.1	Functional and Non-functional Designs . . . . .	1
1.2	Entity-Relationship Diagram . . . . .	3
1.3	Proof of Normalization Form . . . . .	4
<b>2</b>	<b>Implement of DB Entities</b>	<b>4</b>
2.1	Database schema . . . . .	4
2.2	Views . . . . .	4
2.3	Stored Procedures . . . . .	4
2.4	Triggers . . . . .	5
<b>3</b>	<b>Performance Tuning</b>	<b>5</b>
3.1	Problem with filter . . . . .	6
3.2	Selecting over partitions/indexes . . . . .	7
<b>4</b>	<b>Security Configuration</b>	<b>8</b>
4.1	Access Control & Authorization . . . . .	8
4.2	Password Storage . . . . .	8
4.3	Prevent SQL injection . . . . .	9
4.4	End-to-End Testing & Web Integration . . . . .	9
<b>5</b>	<b>Presentation &amp; Material</b>	<b>26</b>

## 1 Conceptual & Physical Design

The project is aimed to provide a yet minimal but still functional of a management application for a education institution. The main actors that interacts to the app contains: **Administrators**, **Instructors**, and **Students**. There activities will go around the **Courses** at school.

### 1.1 Functional and Non-functional Designs

#### 1.1.1 Functional

##### 1.1.1.1 User Authentication and Authorization:

- The system must allow users (students, instructors, and admins) to log in using email and password credentials.

- Users must be assigned roles (student, instructor, admin) with role-specific access to features (e.g., students can enroll/unenroll, instructors can view/manage courses, admins can manage users and courses).

#### 1.1.1.2 Course Management

- The system must display a list of courses with filters by course name, department, and schedule, supporting pagination (10 courses per page).
- Users can view detailed course information (e.g., ID, name, department, instructor, location, schedule, semester, availability) on a course detail page.
- Instructors and admins can add, edit, and delete courses, while students can only view course details.
- The system must track and display enrolled students for each course.

#### 1.1.1.3 Enrollment Management:

- Students must be able to enroll in a course if availability is greater than zero and they are not already enrolled, with a confirmation dialog before submission.
- Students must be able to unenroll from a course if they are enrolled, with a confirmation dialog before submission.
- The system must update course availability automatically when students enroll or unenroll.
- The system must prevent enrollment if the course is full (availability  $\leq 0$ ) or if the student is already enrolled.

#### 1.1.1.4 User Interface and Navigation:

- The system must provide a responsive web interface using Bootstrap for consistent styling across devices.
- Each role (student, instructor, admin) must have a dedicated dashboard with role-specific options.
- Flash messages must be displayed to provide feedback on actions (e.g., success or error messages for login, enrollment).

#### 1.1.1.5 Analytics

- For management roles like **Administrators** they will have access to several dashboards to report about the courses at the institution .

### 1.1.2 Non-functional

- **Performance:**
  - Having high uptime, with all errors are being handled to prevent crashes.
  - Queries should execute within a reasonable time, with minimal complexity. Current target is 0.5 seconds for a dataset of 1000 enrollments.
- **Security:**
  - Passwords are encrypted when being stored in the database, with supports of password hashing function **bcrypt**.
  - Access control systems based on roles are designed properly, prevent unauthorized information access or disclosure.

- Preventions or mitigations of common web security vulnerabilities as well as applications using SQL database system: SQL Injection, Cross-site Scripting, etc.
- **Scalability:**
  - The application is scalable with increasing numbers of users in an acceptable level, against around 300 users under peak time.
- **Usability:**
  - Provide easy-to-navigate interface, with responsive User Interface with **Bootstrap** library.
- **Reliability:**
  - Ensure data integrity with normalized database design (3NF) and constraints.

## 1.2 Entity-Relationship Diagram

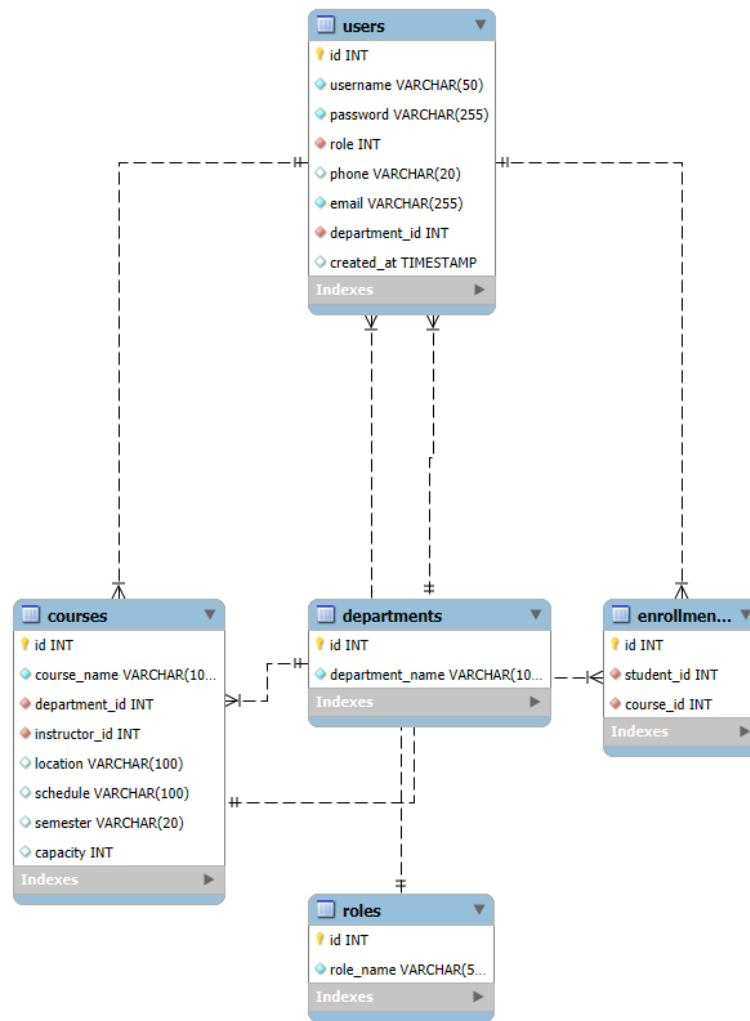


Figure 1: Entity-Relationship Diagram of the project

## 1.3 Proof of Normalization Form

### 1.3.1 First Normal Form (1NF):

- All tables have primary keys
- All attributes are atomic
- No repeating groups

### 1.3.2 Second Normal Form (2NF):

- All non-key attributes are fully functionally dependent on their primary keys
- No partial dependencies exist

### 1.3.3 Third Normal Form (3NF):

- No transitive dependencies
- Proper use of foreign keys for references
- Related data is properly normalized into separate tables

## 2 Implement of DB Entities

### 2.1 Database schema

The full script is in our GitHub repository at this [link](#).

### 2.2 Views

Currently, there is one view `view_course_details` in the database. It provides a simplified and denormalized view of course information with instructor details. This view joins `courses` and `users` tables to provide course information along with the instructor's name, making it easier to retrieve complete course information without writing complex joins in application queries.

```
CREATE VIEW view_course_details AS
SELECT
    c.id AS course_id,
    c.course_name,
    c.department_id,
    c.instructor_id,
    u.username AS instructor_name,
    c.location,
    c.schedule,
    c.semester,
    c.availability
FROM
    courses c
    LEFT JOIN users u ON c.instructor_id = u.id;
```

### 2.3 Stored Procedures

There are 04 stored procedures within the database, which are used as quick reference for most used queries within the lifecycle of the application:

Stored Procedures	Purposes
<code>get_user_role_counts()</code>	Collect number of users across roles within the databases, adaptive with future new roles.
<code>get_course_department_counts()</code>	Collect number of courses across departments, adaptive with future new departments & courses in the institute.
<code>get_student_department_counts()</code>	Collect number of students across departments, adaptive with future new departments, or changing numbers of students
<code>get_course_count_by_semester()</code>	Filtering number of courses in each semesters

Table 1: Stored Procedures used in the application

## 2.4 Triggers

In the functionalities of the application, there are features that automatically update available slots in each course after enrollment. To efficiently operate this feature, we introduced two triggers:

- `before_enrollment_insert()`: This trigger is used to check if there are enough seats left in the course. Whenever a student registered a course, this trigger will reduce the available slots by one. When there is no seat available, it will raise an exception for the business logic to know and handle.
- `after_enrollment_delete()`: In several cases, student may want to drop the course. This trigger simply increase one seat for the course whenever a student unenroll it.

Due to the atomic nature of a trigger in MySQL, there are no race condition error within the the app lifecycle, within the scope of the database itself.

## 3 Performance Tuning

From our default schema, by design of MySQL, several indexes are automatically created (based on UNIQUE keyword, or using INTEGER).

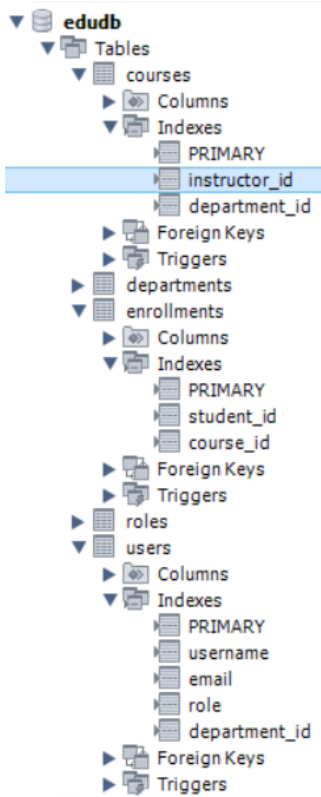


Figure 2: Default indexes in the database

Without any new indexes, normal operations in the database is very fast. For example, the query for a user in the database by email

```
1 • EXPLAIN SELECT * FROM edudb.users WHERE email = 'instructor_77@gmail.com';
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	users	NULL	const	email	email	1022	const	1	100.00	NULL

Figure 3: Performance of email matching to user

### 3.1 Problem with filter

In the application, there is a function to allow users to filter courses based on its schedule. Due to schedule could be any combination of dates within a day, the query could be very slow as there is no current index on **dates**.

**Courses**

Search by Course Name  
Enter course name

Filter by Department  
All Departments

Filter by Schedule  
All Schedules

Apply Filter

ID	Course Name	Department	Id	Day	Room	Schedule	Semester	Availability
3	Mobile App Development	CBM	19	Monday	8	MTR	Spring 2023	-
10	Data Structures	CAS	26	Tuesday	102	MTR	Fall 2024	-
13	Database Systems	CBM	19	Wednesday	102	MTR	Summer 2024	-
16	Web Development	Unknown	111	Thursday	Room 102	MTR	Fall 2023	35
24	Cloud Computing	CBM	85	Friday	Room 104	MTR	Fall 2031	40
52	Introduction to Python	CHS	105		Room 101	MTR	Fall 2030	-
63	Mobile App Development	CAS	54		Room 101	MTR	Summer 2035	-
75	Data Structures	CBM	166		Room 101	MTR	Spring 2024	-
77	Operating Systems	CHS	74		Online	MTR	Summer 2026	43
81	Operating Systems	CAS	132		Room 103	MTR	Summer 2025	12

Previous 1 2 3 Next

Figure 4: Filtering function of the course search

For example, this query on dates required traversal of all 200 courses in our test database, with no potential keys are used.

1 • `EXPLAIN SELECT * FROM edudb.courses WHERE schedule = "M"`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	SIMPLE	courses	NULL	ALL	NULL	NULL	NULL	NULL	200	10.00	Using where

Figure 5: Sample date query

## 3.2 Selecting over partitions/indexes

To minimize the overhead and complexities of adding more columns, we selected to add an index on table `courses`, over the field `schedule`.

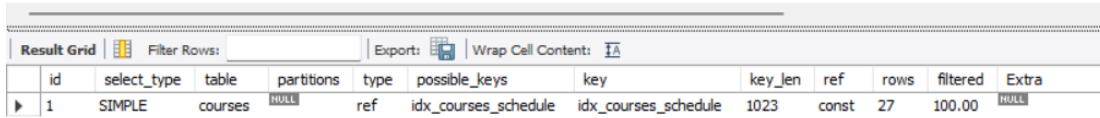
```
CREATE INDEX idx_courses_schedule ON courses(schedule);
```

### 3.2.1 Failed attempts

1. At first, we tried to use hash partitions, by introducing an user defined function to map the schedule as a bitmask for hashing. However, MySQL do not support custom hash functions.
2. We also tried to use list partitions on the string (as there are only 32 combinations of sorted calendar). However, we found out that the InnoDB engine of MySQL do not support partitions for tables that contains foreign keys.

### 3.2.2 Performance after tuning

```
1 • EXPLAIN SELECT * FROM courses WHERE schedule = "M";
```



	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	SIMPLE	courses	NULL	ref	idx_courses_schedule	idx_courses_schedule	1023	const	27	100.00	NULL

Figure 6: Performance after adding the index

With the introduction of this new index, the query performance boost significantly. For example, the database now only needs to filter 19 rows, which are also shared the date is “M”. The reason behind this behavior is that, the index introduced created a hash map from newly created `schedule` value. However, compared to represent the schedule as a bitmask, this is slightly slower; but required a much smaller complexity.

## 4 Security Configuration

### 4.1 Access Control & Authorization

- The system allow users (students, instructors, and admins) to log in using email and password credentials.
- Users must be assigned roles (student, instructor, admin) with role-specific access to features (e.g., students can enroll/unenroll, instructors can view/manage courses, admins can manage users and courses).
- Code References:
  - [auth\\_service.py](#): Handles registration, password validation, login, and password hashing.
  - [role\\_required.py](#): Defines a decorator to restrict route access based on user roles.

### 4.2 Password Storage

- For user password, we are not using native functions in SQL. There are two main reasons:

#### 4.2.1 Lack of custom hash functions

MySQL only offers plain cryptographic hash functions, and a lot of them are deprecated like MD5 or SHA1. Also, it only supports AES-256 for symmetric encryption, but it is not are in the scope of our usage for password storage (else, we need to introduce mechanism for user’s secret key)

#### 4.2.2 Lack of salt before hashing

For plain usage of cryptographic hash function, user’s password is still at risk of rainbow search attack, which attackers attempts to brute-force for the password (for matching hash), or looking up in public database.



- Therefore, passwords in the database are encrypted when being stored in the database, with supports of password hashing function `bcrypt`, which provides the mechanism for a easy to implement password hashing scheme.

### 4.3 Prevent SQL injection

A crucial component of the application is making sure that database interactions are secure. All functions that communicate with the database are grouped under the `app/db/` directory in order to prevent SQL injection attacks. These functions are implemented to only run SQL queries that are parameterized, which significantly reduces the possibility of malicious SQL code injection. By separating SQL code from user input and guaranteeing that input data is handled as data rather than executable code, parameterized queries improve the system's overall security posture.

```
cursor.execute("SELECT id FROM roles WHERE role_name = %s", (role_name,))
role_id = cursor.fetchone()[0]
cursor.execute(
    "INSERT INTO users (email, username, password, role, phone, department_id)
    VALUES (%s, %s, %s, %s, %s, %s)",
    (email, username, password, role_id, phone, department_id)
)
```

### 4.4 End-to-End Testing & Web Integration

Due to the scope of this project, we conducted **manual end-to-end testing** to verify the system's functionality by inspecting both the user interface and the underlying database.

#### 4.4.1 Insert Sample Data

Sample data was injected into the system using the following commands:

```
docker cp test.py edu-flask-db-project_web_1:/tmp/test.py
Mac/Linux: docker exec -it edu-flask-db-project_web_1 python3 /tmp/test.py
Windows: docker exec -it edu-flask-db-project_web_1 python /tmp/test.py
```

```

Created student: student_1260, ID: 1461, Email: student_1260@yahoo.com, Password: UpRJENhq
Created student: student_1261, ID: 1462, Email: student_1261@yahoo.com, Password: ORisi6pc
Created student: student_1262, ID: 1463, Email: student_1262@example.com, Password: hpFRvYT8
Created student: student_1263, ID: 1464, Email: student_1263@example.com, Password: EKIXykOT
Created student: student_1264, ID: 1465, Email: student_1264@yahoo.com, Password: 26ykeGRs
Created student: student_1265, ID: 1466, Email: student_1265@example.com, Password: KBbmQCFb
Created student: student_1266, ID: 1467, Email: student_1266@yahoo.com, Password: BciRa7e9
Created student: student_1267, ID: 1468, Email: student_1267@yahoo.com, Password: KgZ7XV0g
Created student: student_1268, ID: 1469, Email: student_1268@yahoo.com, Password: AEptgrFC
Created student: student_1269, ID: 1470, Email: student_1269@yahoo.com, Password: HTkUBXrk
Created student: student_1270, ID: 1471, Email: student_1270@example.com, Password: fIKP1CDF
Created student: student_1271, ID: 1472, Email: student_1271@example.com, Password: N6UfBXZW
Created student: student_1272, ID: 1473, Email: student_1272@hotmail.com, Password: IwbbeIFS
Created student: student_1273, ID: 1474, Email: student_1273@gmail.com, Password: koLrTrEm
Created student: student_1274, ID: 1475, Email: student_1274@yahoo.com, Password: UpQj7F12
Created student: student_1275, ID: 1476, Email: student_1275@hotmail.com, Password: rFrEnL0B
Created student: student_1276, ID: 1477, Email: student_1276@yahoo.com, Password: SlQRhKhH
Created student: student_1277, ID: 1478, Email: student_1277@hotmail.com, Password: JeSon8My
Created student: student_1278, ID: 1479, Email: student_1278@yahoo.com, Password: 5o9UePIq
Created student: student_1279, ID: 1480, Email: student_1279@gmail.com, Password: 4541kmp3
Created student: student_1280, ID: 1481, Email: student_1280@hotmail.com, Password: EB5Ikg0b
Created student: student_1281, ID: 1482, Email: student_1281@yahoo.com, Password: seSjvI1C
Created student: student_1282, ID: 1483, Email: student_1282@gmail.com, Password: UZ5Liwdm
Created student: student_1283, ID: 1484, Email: student_1283@example.com, Password: 6GuuzF1A
Created student: student_1284, ID: 1485, Email: student_1284@gmail.com, Password: BzKMsa7a
Created student: student_1285, ID: 1486, Email: student_1285@example.com, Password: Fg3leuMJ
Created student: student_1286, ID: 1487, Email: student_1286@yahoo.com, Password: h0x7ywgM
Created student: student_1287, ID: 1488, Email: student_1287@gmail.com, Password: RC1e0HRy

```

#### 4.4.2 Registration

All registered accounts are initially assigned the role of “guest.” Only administrators have the authority to assign roles such as “student” or “instructor.”

The screenshot shows a web browser window with the address bar displaying 'localhost:5000/register'. The page header includes the 'EduSys' logo and 'Login Register' links. The main content area is titled 'Register' and contains a form with the following fields:

- Email:** test\_register@gmail.com
- Username:** test\_register@gmail.com
- Password:** (masked with dots)
- Confirm Password:** (masked with dots)

Below the form is a blue 'Register' button. At the bottom of the form, there is a link: 'Already have an account? [Login](#)'.

An error message is displayed to enforce strong password requirements.

Password must be at least 8 characters long and include uppercase, lowercase, number, and special character

This was the result of testing with the password Database#2025.

EduSys Login Register

Registration successful! Please log in.

Email  
admin@admin.admin

Password  
Database#2025

Login

Don't have an account? [Register](#)

The corresponding record in the database, showing the password is securely hashed using **bcrypt**:

```
mysql> SELECT u.*, r.role_name FROM users u JOIN roles r ON u.role = r.id WHERE
↳ u.email = "test_register@gmail.com";
+-----+-----+-----+-----+-----+-----+-----+-----+
↳ -----+-----+-----+-----+-----+-----+-----+-----+
↳ -----+-----+-----+-----+-----+-----+-----+-----+
| id   | username      | password                                     | department_id | created_at |
↳ | role | phone | email                                     | department_id | created_at |
↳ | role_name |
+-----+-----+-----+-----+-----+-----+-----+-----+
↳ -----+-----+-----+-----+-----+-----+-----+-----+
↳ -----+-----+-----+-----+-----+-----+-----+-----+
| 3202 | test_register | $2b$12$ar.ffzBCzJ9s.xXFXReoo0ZahTzEEDpV6EOr0QBebndDKwLnre7. | 4 | NULL |
↳ | test_register@gmail.com | 5 | 2025-05-25 11:06:39 | guest |
↳ |
+-----+-----+-----+-----+-----+-----+-----+-----+
↳ -----+-----+-----+-----+-----+-----+-----+-----+
↳ -----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### 4.4.3 Login

#### 4.4.3.1 As Admin

The administrator account is predefined in the source code and can be customized freely. Below is the admin dashboard upon successful login:

## Admin Dashboard

Logged in successfully!

### Manage Courses

View, create, edit, or delete courses.

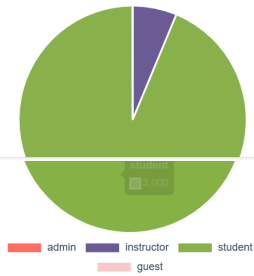
[Courses Dashboard](#)

### Manage Users

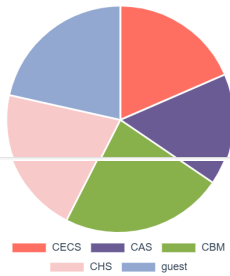
View, create, edit, or delete users.

[Users Dashboard](#)

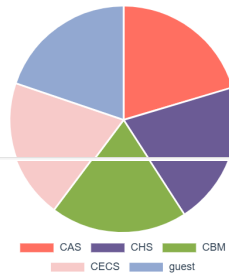
User Role Distribution



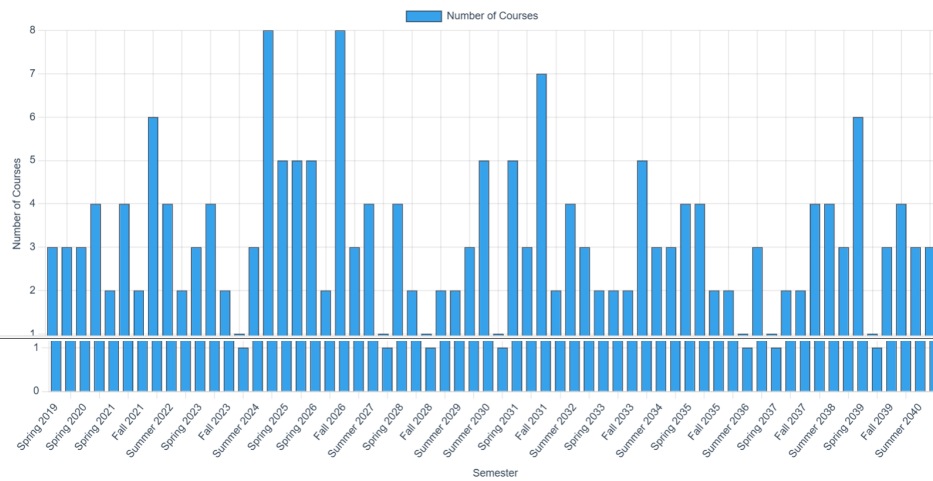
Course Department Distribution



Student Department Distribution

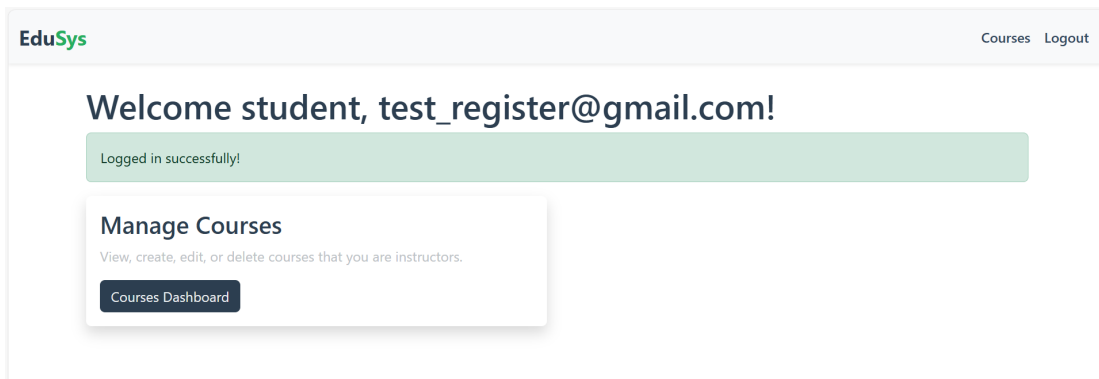


Number of Courses Over Time



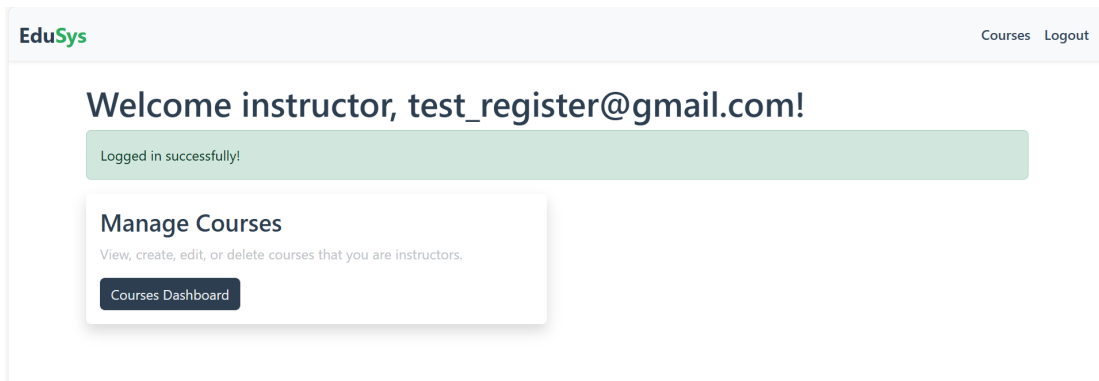
#### 4.4.3.2 As Student

Students can view all available courses and manage their enrollments.



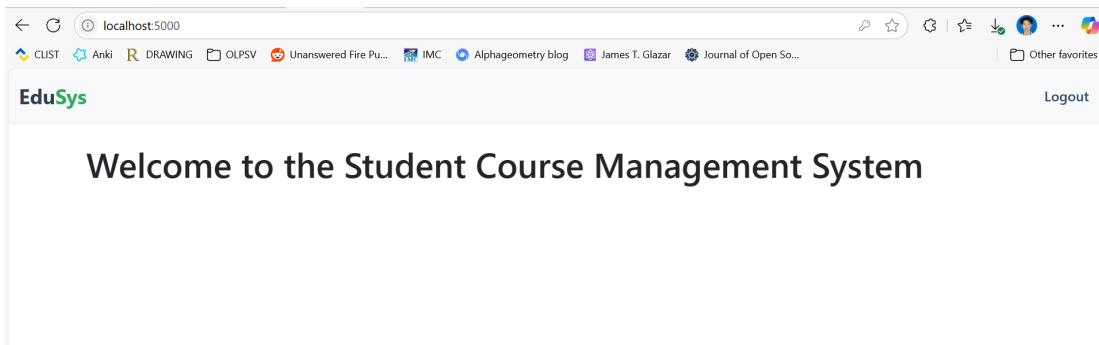
#### 4.4.3.3 As Instructor

Instructors have access to view and manage only the courses they are responsible for.



#### 4.4.3.4 As Guest

Guests do not have access to core functionalities until their role is upgraded.



### 4.4.4 Admin Functions

#### 4.4.4.1 Manage Courses

##### 4.4.4.1.1 Add Course

**Courses Dashboard**

Search by course name All Departments Add Course

ID	Course Name	Department	Instructor	Location	Schedule	Semester	Availability	Actions
1	DevOps	CBM						Edit Delete
2	Database Systems	CECS						Edit Delete
3	Artificial Intelligence	CHS						Edit Delete
4	Algorithms	CBM						Edit Delete
5	Mobile App Development	CAS						Edit Delete
6	DevOps	gue						Edit Delete
7	DevOps	gue						Edit Delete

Error in schedule: Schedule must be comma-separated codes like "M, W, F" (M=Mon, T= Tue, etc.).

Error in semester: Semester must be in the format "Spring/Fall/Summer YYYY".

Validation is applied to ensure all fields follow correct format. Confirmation can be verified through both UI and database:

**Courses Dashboard**

Test All Departments All Schedules Search Add Course

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability	Actions
201	Test_Adding_Course	CECS	2	C405	M	Fall 2025	100	Edit Delete

Previous **1** Next

```
mysql> SELECT * FROM courses WHERE course_name = 'Test_Adding_course';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | course_name | department_id | instructor_id | location | schedule | semester | availability |
+-----+-----+-----+-----+-----+-----+-----+
| 201 | Test_Adding_Course | 1 | 2 | C405 | M | Fall 2025 | 100 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

#### 4.4.4.1.2 Delete Course

The screenshot shows the EduSys web application interface. At the top, a browser address bar displays the URL: `localhost:5000/admin/dashboard/courses/?search=Test&department_id=&schedule=`. A confirmation dialog box is open, titled "localhost:5000 says", with the text "Are you sure you want to delete this course?" and two buttons: "OK" and "Cancel".

The main content area is titled "Courses Dashboard". It includes a search bar with the text "Test", dropdown menus for "All Departments" and "All Schedules", and a "Search" button. Below the search bar is a green "Add Course" button.

A table lists the courses. The table has the following columns: ID, Course Name, Department, Instructor ID, Location, Schedule, Semester, Availability, and Actions. The table contains one row with the following data:

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability	Actions
201	Test_Adding_Course	CECS	2	C405	M	Fall 2025	100	<a href="#">Edit</a> <a href="#">Delete</a>

Below the table is a pagination bar with "Previous", "1", and "Next" buttons.

The bottom part of the screenshot shows the same "Courses Dashboard" but with the message "No courses found." displayed in the table area, indicating the course has been successfully deleted.

Record removal is confirmed through the following result:

The screenshot shows the EduSys web application interface. At the top, a browser address bar displays the URL: `localhost:5000/admin/dashboard/courses/?search=Test&department_id=&schedule=`. A confirmation dialog box is open, titled "localhost:5000 says", with the text "Are you sure you want to delete this course?" and two buttons: "OK" and "Cancel".

The main content area is titled "Courses Dashboard". It includes a search bar with the text "Test", dropdown menus for "All Departments" and "All Schedules", and a "Search" button. Below the search bar is a green "Add Course" button.

A table lists the courses. The table has the following columns: ID, Course Name, Department, Instructor ID, Location, Schedule, Semester, Availability, and Actions. The table contains one row with the following data:

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability	Actions
201	Test_Adding_Course	CECS	2	C405	M	Fall 2025	100	<a href="#">Edit</a> <a href="#">Delete</a>

Below the table is a pagination bar with "Previous", "1", and "Next" buttons.

The bottom part of the screenshot shows the same "Courses Dashboard" but with the message "No courses found." displayed in the table area, indicating the course has been successfully deleted.

```
mysql> SELECT * FROM courses WHERE course_name = 'Test_Adding_course';
Empty set (0.00 sec)
```

#### 4.4.4.1.3 Add Students to Course

Students can be assigned or removed from a course by selecting the course from the list:

## Course Details

**Course ID:** 202

**Course Name:** Test\_add\_student\_instructor

**Department ID:** 1

**Instructor ID:** 3202

**Instructor Name:** test\_register

**Location:** C405

**Schedule:** M

**Semester:** Fall 2025

**Availability:** 100

Add New Student

## Enrolled Students

No students enrolled yet.

EduSys Courses Logout

### Add Student to Test\_add\_student\_instructor

Select Student

Select a student

Enroll Student

Close

## Course Detail - Test\_add\_student\_instructor

### Course Details

**Course ID:** 202

**Course Name:** Test\_add\_student\_instructor

**Department ID:** 1

**Instructor ID:** 3202

**Instructor Name:** test\_register

**Location:** C405

**Schedule:** M

**Semester:** Fall 2025

**Availability:** 100

Add New Student

**Availability:** 99

Add New Student

## Enrolled Students

ID	Email	Username	Department	Action
202	student_001@yahoo.com	student_001	CAS	<a href="#">Remove</a>

```
SELECT u.id AS student_id, u.username, u.email, d.department_name
FROM enrollments e
JOIN users u ON e.student_id = u.id
JOIN courses c ON e.course_id = c.id
JOIN departments d ON u.department_id = d.id
WHERE c.course_name = 'test_add_student_instructor';
```



#### 4.4.4.1.4 Remove Students from Course

The screenshot shows the EduSys admin interface at `localhost:5000/admin/dashboard/courses/202`. A modal dialog titled "localhost:5000 says" is open, asking "Are you sure you want to unenroll this student?". The background shows the course details for "Test\_add\_student\_instructor" with fields for Department ID, Instructor ID, Instructor Name, Location, Schedule, Semester, and Availability. Below this is a table of enrolled students.

ID	Email	Username	Department	Action
202	student_001@yahoo.com	student_001	CAS	<button>Remove</button>

The bottom part of the screenshot shows the "Course Detail - Test\_add\_student\_instructor" page with a "Course Details" section containing the same fields as the modal dialog, but with an "Availability" of 100.

```
mysql> SELECT u.id AS student_id, u.username, u.email, d.department_name
-> FROM enrollments e
-> JOIN users u ON e.student_id = u.id
-> JOIN courses c ON e.course_id = c.id
-> JOIN departments d ON u.department_id = d.id
-> WHERE c.course_name = 'test_add_student_instructor';
Empty set (0.02 sec)
```

#### 4.4.4.2 Manage Users

#### 4.4.4.2.1 Set Instructor Role

EduSys

Courses Logout

## Users Dashboard

ID	Username	Email	Role	Phone	Department	Actions
3202	test_register	test_register@gmail.com	guest	-	guest	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

EduSys

Courses Logout

## Users Dashboard

ID	Username	Email	Role	Phone	Department	Actions
3202	test_register	test_register@gmail.com	guest	-	guest	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Edit User: test\_register

Username

Email

Role

Phone

Department ID

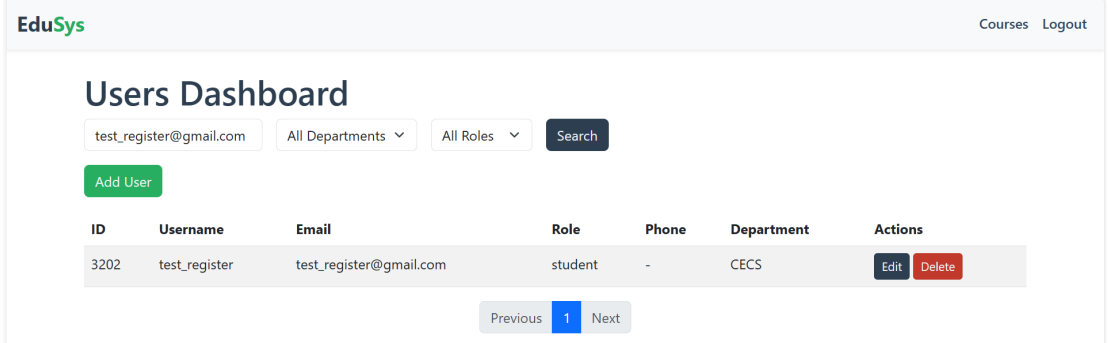
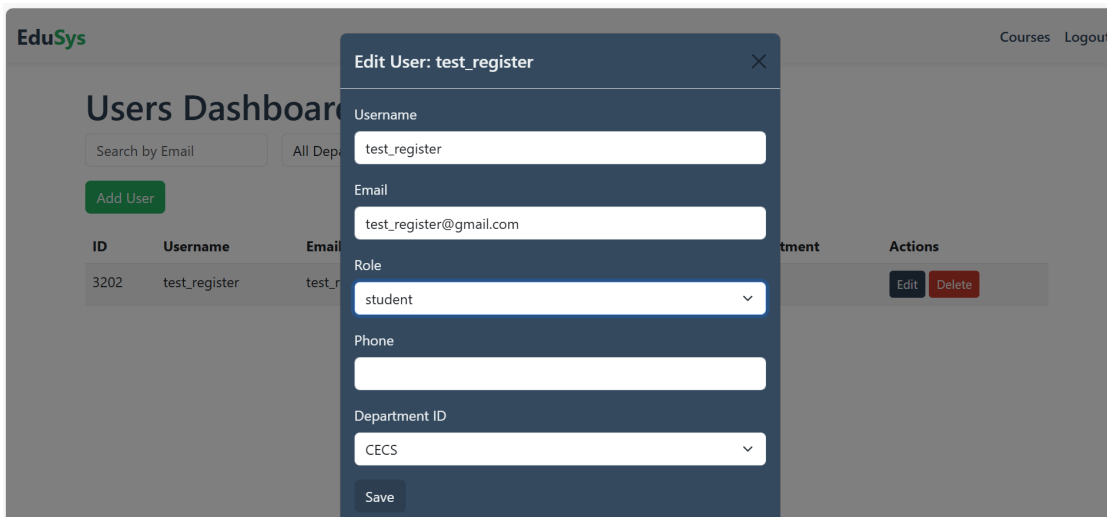
EduSys

Courses Logout

## Users Dashboard

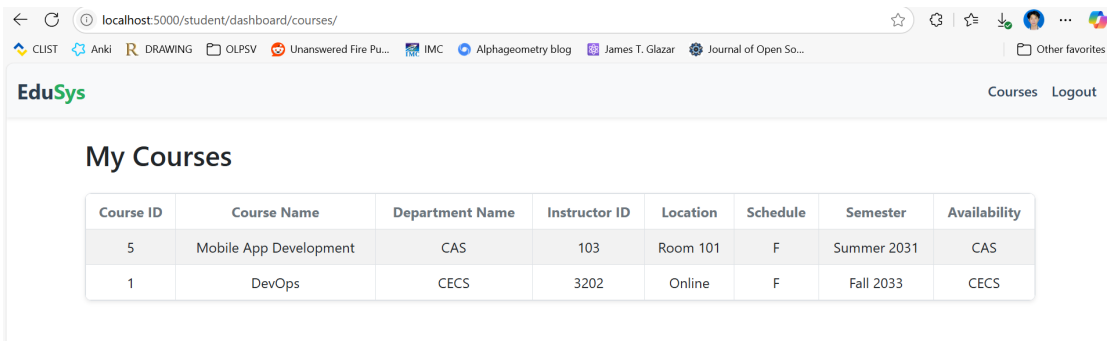
ID	Username	Email	Role	Phone	Department	Actions
3202	test_register	test_register@gmail.com	instructor	-	CECS	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

#### 4.4.4.2.2 Set Student Role



#### 4.4.5 Student Functions

Students are able to view their current enrollments.



##### 4.4.5.1 Enroll in Courses

###### 4.4.5.1.1 Non-Full Course

localhost:5000/courses/

CLUSTAnkiDRAWINGOLPSVUnanswered Fire Pu...IMCAlphageometry blogJames T. GlazarJournal of Open So...Other favori...

EduSysCoursesLogout

## Courses

Search by Course Name

Filter by Department

Filter by Schedule

Apply Filters

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability
1	DevOps	CECS	3202	Online	F	Fall 2033	25
2	Database Systems	CECS	148	Room 103	T	Spring 2031	18
3	Artificial Intelligence	CHS	168	Online	MF	Spring 2039	-
4	Algorithms	CBM	70	Online	MF	Spring 2022	35
5	Mobile App Development	CAS	103	Room 101	F	Summer 2031	28
6	DevOps	guest	34	Room 103	MTR	Spring 2027	27
7	DevOps	guest	46	Room 103	T	Summer 2035	13
8	Artificial Intelligence	CBM	91	Room 101	T	Summer 2034	20

localhost:5000/courses/1

CLUSTAnkiDRAWINGOLPSVUnanswered Fire Pu...Open So...Other favo...

## Course Detail - De

Course Details

**Course ID:** 1

**Course Name:** DevOps

**Department ID:** 1

**Instructor ID:** 3202

**Instructor Name:** test\_register

**Location:** Online

**Schedule:** F

**Semester:** Fall 2033

Click to enroll in this course

Enroll Now

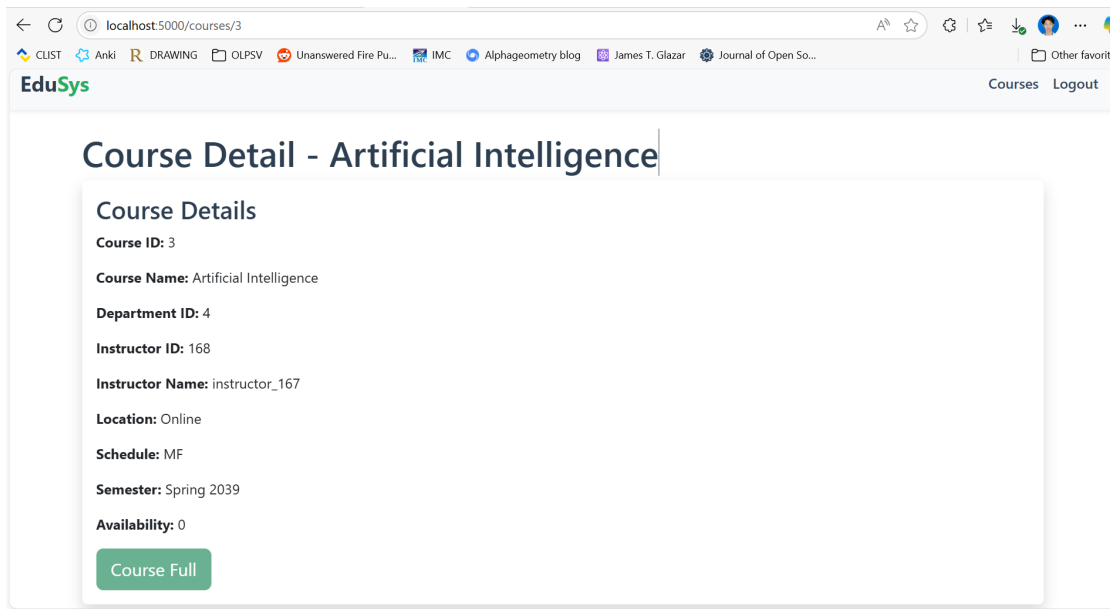
localhost:5000 says  
Are you sure you want to enroll in this course?  
OKCancel

766	student_565@gmail.com	student_565	CBM
1054	student_853@yahoo.com	student_853	CBM
1185	student_984@example.com	student_984	CBM
1283	student_1082@example.com	student_1082	CBM
1297	student_1096@hotmail.com	student_1096	CBM
1418	student_1217@yahoo.com	student_1217	CBM
1722	student_1521@gmail.com	student_1521	CBM
3203	test_enroll_student@gmail.com	test_enroll_student	CBM
324	student_123@hotmail.com	student_123	CECS
372	student_171@yahoo.com	student_171	CECS
389	student_188@hotmail.com	student_188	CECS
427	student_226@example.com	student_226	CECS
515	student_314@example.com	student_314	CECS
541	student_340@hotmail.com	student_340	CECS
555	student_354@yahoo.com	student_354	CECS

```
mysql> SELECT e.*
-> FROM enrollments e
-> JOIN users u ON e.student_id = u.id
-> JOIN courses c ON e.course_id = c.id
-> WHERE u.username = 'test_enroll_student' AND c.course_name = 'DevOps';
```

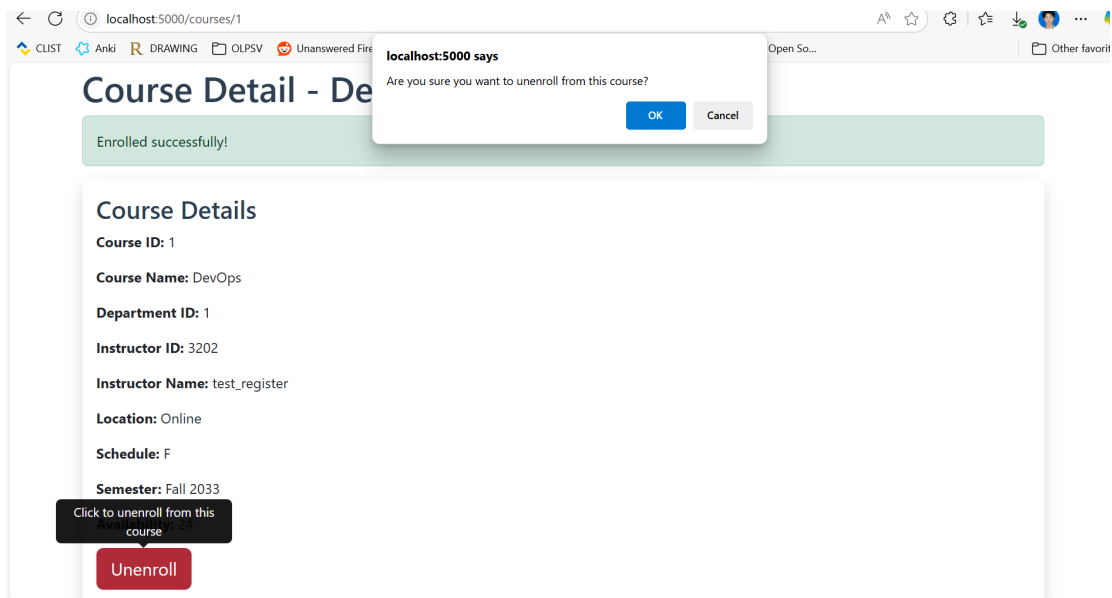
```
+-----+-----+-----+
| id    | student_id | course_id |
+-----+-----+-----+
| 9499  | 3203      | 1         |
+-----+-----+-----+
1 row in set (0.00 sec)
```

#### 4.4.5.1.2 Full Course



When the course has no remaining availability, the system blocks enrollment.

#### 4.4.5.2 Unenroll from Course



localhost:5000/courses/1

## Course Detail - DevOps

Unenrolled successfully!

### Course Details

**Course ID:** 1

**Course Name:** DevOps

**Department ID:** 1

**Instructor ID:** 3202

**Instructor Name:** test\_register

**Location:** Online

**Schedule:** F

**Semester:** Fall 2033

**Availability:** 25

[Enroll Now](#)

```
mysql> SELECT e.*
-> FROM enrollments e
-> JOIN users u ON e.student_id = u.id
-> JOIN courses c ON e.course_id = c.id
-> WHERE u.username = 'test_enroll_student' AND c.course_name = 'DevOps';
Empty set (0.01 sec)
```

#### 4.4.6 Instructor View

The interface is similar to the administrator but limited to courses assigned to the instructor.

EduSys [Courses](#) [Logout](#)

### My Courses

Course ID	Course Name	Department Name	Location	Schedule	Semester	Availability
1	DevOps	CECS	Online	F	Fall 2033	25
202	Test_add_student_instructor	CECS	C405	M	Fall 2025	100

## Course Detail - DevOps

### Course Details

**Course ID:** 1

**Course Name:** DevOps

**Department ID:** 1

**Instructor ID:** 3202

**Instructor Name:** test\_register

**Location:** Online

**Schedule:** F

**Semester:** Fall 2033

**Availability:** 25

Add New Student

### Enrolled Students

ID	Email	Username	Department	Action
214	student_013@gmail.com	student_013	CAS	<a href="#">Remove</a>
525	student_324@example.com	student_324	CAS	<a href="#">Remove</a>
527	student_326@hotmail.com	student_326	CAS	<a href="#">Remove</a>
863	student_662@gmail.com	student_662	CAS	<a href="#">Remove</a>
878	student_677@gmail.com	student_677	CAS	<a href="#">Remove</a>
1048	student_847@yahoo.com	student_847	CAS	<a href="#">Remove</a>
204	student_003@gmail.com	student_003	CBM	<a href="#">Remove</a>
290	student_089@hotmail.com	student_089	CBM	<a href="#">Remove</a>



#### 4.4.7 Course View and Filtering

EduSys

CoursesLogout

### Courses

Search by Course Name

Filter by Department

Filter by Schedule

Apply Filters

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability
1	DevOps	CBM	14	Online	R	Fall 2033	57
2	Database Systems	CECS	148	Room 103	T	Spring 2031	68
3	Artificial Intelligence	CHS	168	Online	MF	Spring 2039	37
4	Algorithms	CBM	70	Online	MF	Spring 2022	87
5	Mobile App Development	CAS	103	Room 101	F	Summer 2031	86
6	DevOps	guest	34	Room 103	MTR	Spring 2027	89
7	DevOps	guest	46	Room 103	T	Summer 2035	72
8	Artificial Intelligence	CBM	91	Room 101	T	Summer 2034	81
9	Cloud Computing	CHS	122	Room 103	W	Spring 2028	79
10	Operating Systems	CAS	73	Online	TR	Spring 2029	29

Previous

1

2

...

20

Next

##### 4.4.7.1 Filter by Name

EduSys

CoursesLogout

### Courses

Search by Course Name

Filter by Department

Filter by Schedule

Apply Filters

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability
4	Algorithms	CBM	70	Online	MF	Spring 2022	35
38	Algorithms	CECS	146	Online	W	Fall 2026	-
50	Algorithms	CAS	134	Room 101	TW	Spring 2039	13
64	Algorithms	CBM	51	Room 103	MW	Spring 2021	30
79	Algorithms	CAS	174	Room 105	M	Summer 2030	-
86	Algorithms	CBM	115	Room 105	T	Fall 2037	-
92	Algorithms	guest	63	Room 103	T	Summer 2023	2
101	Algorithms	CBM	30	Room 104	F	Summer 2022	-

##### 4.4.7.2 Filter by Department

## Courses

Search by Course Name

Filter by Department

CHS ▾

Filter by Schedule ▾

Apply Filters

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability
20	DevOps	CHS	172	Room 103	MTR	Spring 2030	-
26	Mobile App Development	CHS	2	Room 104	M	Spring 2025	-
29	DevOps	CHS	34	Room 101	T	Spring 2038	-
35	DevOps	CHS	66	Room 102	MTR	Fall 2025	12
48	Mobile App Development	CHS	97	Room 101	MF	Fall 2038	-
72	Web Development	CHS	50	Online	MF	Summer 2035	-
110	Mobile App Development	CHS	134	Room 103	MW	Spring 2034	-
123	DevOps	CHS	133	Online	MTR	Fall 2040	-
126	Web Development	CHS	139	Room 101	T	Summer 2038	2
160	Web Development	CHS	103	Room 101	F	Spring 2020	-

### 4.4.7.3 Filter by Schedule

## Courses

Search by Course Name

Filter by Department

All Departments ▾

Filter by Schedule ▾

Apply Filters

ID	Course Name	Department	Instructor ID	Location	Schedule	Semester	Availability
1	DevOps	CBM	14	Online	R	Fall 2033	-
18	Machine Learning	CHS	189	Room 102	R	Summer 2020	-
37	Machine Learning	CHS	110	Online	R	Summer 2024	-
44	Networking	CHS	102	Room 102	R	Spring 2034	-
56	Software Engineering	CECS	91	Room 105	R	Summer 2032	-
58	Web Development	CAS	139	Room 101	R	Spring 2024	22
61	Operating Systems	CECS	83	Room 103	R	Fall 2038	26
67	Mobile App Development	guest	181	Room 105	R	Spring 2035	17
75	DevOps	CECS	69	Room 104	R	Summer 2020	13
76	Data Structures	CECS	157	Room 102	R	Summer 2021	15

## 5 Presentation & Material

- [Presentation's slide deck](#)
- [Final report](#)
- Source code: <https://github.com/Thanh-WuTan/edu-flask-db-project>