E-LEARNING PLATFORM

A MINI-PROJECT REPORT

Submitted by

NITHIN S 241801191

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

NOVEMBER 2025

BONAFIDE CERTIFICATE

Certified that this project **"E-LEARNING PLATFORM"** is the bonafide work of **"NITHIN S"** who carried out the project work under my supervision.

SIGNATURE

Dr. E.K.SUBRAMANIAN

ASSOCIATE PROFESSOR

Dept. of Artificial Intelligence and Data Science,
Rajalakshmi Engineering College
Chennai

This mini project report is submitted for the viva voce examination to be held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The purpose of this mini-project was to design and implement a fundamental **E-Learning Platform** simulating core features of a modern online course website, with a strong focus on **Database Management System (DBMS)** principles. The core objective was to effectively manage the relationships between **Instructors, Courses, Lessons, and Student Enrollments**.

The final system consists of a three-tier architecture:

1. **Data Tier (DBMS):** A **SQLite** database stores structured course information and user enrollment records, demonstrating relational database design.
2. **Application Tier (Backend):** A **Python Flask** server provides a RESTful API to initialize the database, retrieve course data, check student enrollment status, and process enrollment (purchase) requests.
3. **Presentation Tier (Frontend):** A single, fully responsive HTML/CSS/JavaScript interface allows users to sign in (mock authentication), browse courses, view details, and simulate the purchase/enrollment process using a dark, modern theme.

This project successfully demonstrates CRUD operations, foreign key constraints, and the practical application of SQL within a modern web application context.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honorable chairman **MR. S. MEGANATHAN** and the Chairperson **DR**. **M. THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal Dr. **S.N. MURUGESAN** for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head of the Department **Dr. J.M. GNANASEKAR** for being ever supporting force during our project work

We also extend our sincere and hearty thanks to our internal guide **Dr. E.K. SUBRAMANIAN,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

1. NITHIN S

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

E-Learning platforms are crucial in modern education, enabling scalable and accessible learning worldwide. These systems rely heavily on robust database management to organize complex relationships between users, courses, and educational content. This project, named **LearnVerse**, is a simulation designed to model these database functionalities

1.2 SCOPE OF THE WORK

The scope of this work is to demonstrate a functional full-stack DBMS application. It includes:

2 Designing and implementing a **relational database model** with four tables: **Users, Courses, Lessons, and Enrollments**.

3 Creating a **Python backend (Flask)** to serve course data and handle enrollment using a REST API.

Developing a **responsive frontend** that interacts with the backend to fetch and display data, simulate user sign-in, and manage the course buying and sign-out processes.

1.3 PROBLEM STATEMENT

Modern educational platforms require a database that can efficiently manage concurrent user access, maintain data integrity (e.g., ensuring only valid students can enroll in existing courses), and track transactional relationships (who bought what and when). The project addresses the need for a system that can accurately and reliably map these entities and relationships.

1.4 AIM AND OBJECTIVES OF THE PROJECT

Aim: To develop a functional DBMS mini-project showcasing essential relational database design, API interaction, and front-end development using Python, SQLite, and Flask.

Objectives:Design and implement a database schema that correctly uses Foreign Keys to link entities (e.g., Courses.instructor_id references Users.id).

1. **Implement CRUD operations for course data and enrollment management via Flask API endpoints.**
2. **Utilize the Enrollments table to track the many-to-many relationship between Students and Courses.**
3. **Create a user interface that dynamically updates based on data fetched from the backend (e.g., changing the "Buy" button to "View Course" upon successful enrollment).**

CHAPTER 2

SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS

Processor	:	Intel i5
Memory Size	:	8GB (Minimum)
HDD	:	256 TB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

Operating System	:	WINDOWS 10
Front – End	:	**HTML,TAILWIND CSS,HTML**
Back - End	:	PYTHON,SQL LITE
Language	:	python,SQL LITE

CHAPTER 3

MODULE DESCRIPTION

The application is structured into four functional areas:

1. Database Initialization (init_db in app.py)

This module is executed on server startup. It creates the four required SQL tables (Users, Courses, Lessons, Enrollments) and inserts initial mock data for instructors and courses, ensuring the platform is ready for use immediately.

2. Course Management API (Backend)

This is the heart of the REST API, handling requests from the frontend:

- /api/courses (GET): Retrieves the list of all courses and their instructors using a **SQL JOIN** operation.
- /api/course/<id> (GET): Retrieves detailed information, including all associated lessons, and checks the user's **Enrollment Status** against the Enrollments table.

3. Enrollment and Payment Module (Backend and Frontend)

This module handles the transactional aspect of the E-Learning platform:

- **Frontend:** Prompts the user to **Sign In**, uses a **Mock Payment Modal** for purchase simulation, and calls the backend.
- /api/enroll (POST): This API endpoint receives the user_id and course_id. It performs a final check for existing enrollment and then **inserts a new record** into the **Enrollments** table (a crucial DBMS transaction).

4. Frontend Interface (index.html)

The single-page web interface, styled with a dark theme (Tailwind CSS), manages the views and user interaction:

- **Authentication:** Handles mock sign-in and sign-out, setting the global MOCK_USER_ID.
- **Dashboard View:** Displays dynamic course cards.
- **Detail View:** Shows course metadata and lessons, and controls the state of the "Buy/View Course" button based on the API response.

CHAPTER 4

```
import sqlite3

import json

from flask import Flask, jsonify, redirect, request # Added 'request'

from flask_cors import CORS



# --- FLASK APP SETUP ---

app = Flask(__name__)

CORS(app)



DATABASE = 'e_learning.db'



def get_db_connection():

    """Connects to the SQLite database and sets up row factory for dict-like rows."""

    conn = sqlite3.connect(DATABASE)

    conn.row_factory = sqlite3.Row
```

```
    return conn


def init_db():
    """Initializes the database schema and populates it with dummy data."""
    print("Initializing database...")
    conn = get_db_connection()
    cursor = conn.cursor()

    # 1. Users Table (DBMS Entity)
    # Role 'I' for Instructor, 'S' for Student
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Users (
            id INTEGER PRIMARY KEY,
            name TEXT NOT NULL,
            email TEXT UNIQUE NOT NULL,
            role TEXT NOT NULL
```

```
    );
"""")


# 2. Courses Table (DBMS Entity)
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Courses (
        id INTEGER PRIMARY KEY,
        title TEXT NOT NULL,
        description TEXT,
        instructor_id INTEGER,
        FOREIGN KEY (instructor_id) REFERENCES Users(id)
    );
""")


# 3. Lessons Table (DBMS Entity) - Strong relationship with Courses
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Lessons (
```

```
        id INTEGER PRIMARY KEY,
        course_id INTEGER NOT NULL,
        title TEXT NOT NULL,
        content TEXT,
        FOREIGN KEY (course_id) REFERENCES Courses(id)
    );
""")

# 4. Enrollments Table (DBMS Relationship)
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Enrollments (
        user_id INTEGER NOT NULL,
        course_id INTEGER NOT NULL,
        enroll_date DATE DEFAULT CURRENT_DATE,
        PRIMARY KEY (user_id, course_id),
        FOREIGN KEY (user_id) REFERENCES Users(id),
        FOREIGN KEY (course_id) REFERENCES Courses(id)
```

```
    );
"""")

# --- Sample Data Insertion ---

# Insert Users (Added student user ID 4 for enrollment testing)
cursor.execute("INSERT OR IGNORE INTO Users (id, name, email, role) VALUES (?, ?, ?, ?)", (1, 'Dr. Aris Patel', 'aris@plat.edu', 'I'))
cursor.execute("INSERT OR IGNORE INTO Users (id, name, email, role) VALUES (?, ?, ?, ?)", (2, 'Prof. Lin Wang', 'lin@plat.edu', 'I'))
cursor.execute("INSERT OR IGNORE INTO Users (id, name, email, role) VALUES (?, ?, ?, ?)", (3, 'Dr. Anya Sharma', 'anya@plat.edu', 'I'))
cursor.execute("INSERT OR IGNORE INTO Users (id, name, email, role) VALUES (?, ?, ?, ?)", (4, 'Mock Student User', 'student@plat.edu', 'S'))

# Insert Courses
cursor.execute("INSERT OR IGNORE INTO Courses (id, title, description, instructor_id) VALUES (?, ?, ?, ?)",
```

(101, 'Introduction to SQL and Relational Databases', 'Learn the fundamental concepts of database management, normalization, and SQL querying.', 1))

cursor.execute("INSERT OR IGNORE INTO Courses (id, title, description, instructor_id) VALUES (?, ?, ?, ?)",

(102, 'Python Web Development with Flask', 'A practical guide to building REST APIs using the Flask framework.', 2))

cursor.execute("INSERT OR IGNORE INTO Courses (id, title, description, instructor_id) VALUES (?, ?, ?, ?)",

(103, 'Data Structures & Algorithms in Python', 'Master core DSA concepts including trees, graphs, and dynamic programming for coding interviews.', 1))

cursor.execute("INSERT OR IGNORE INTO Courses (id, title, description, instructor_id) VALUES (?, ?, ?, ?)",

(104, 'Advanced Cloud Computing (AWS Focus)', 'Explore serverless architecture, microservices, and large-scale deployment using AWS.', 3))

Insert Lessons

cursor.execute("INSERT OR IGNORE INTO Lessons (course_id, title, content) VALUES (?, ?, ?)", (101, 'What is a Database?', 'A database is an organized collection of data...'))

```
    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (101, 'SQL Basic Queries', 'SELECT,
FROM, and WHERE clauses are the foundation of SQL.'))

    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (102, 'Setting up the Flask Project',
'Initialize your Python environment and install Flask.'))

    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (102, 'Creating API Endpoints',
'Define routes to handle GET and POST requests.'))

    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (103, 'Big O Notation and Time
Complexity', 'Understanding how to measure algorithm efficiency.'))

    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (103, 'Introduction to Binary Search
Trees', 'Balanced vs. unbalanced trees and common operations.'))

    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (104, 'Serverless Functions (Lambda)',
'Building and deploying FaaS functions.'))

    cursor.execute("INSERT OR IGNORE INTO Lessons (course_id,
title, content) VALUES (?, ?, ?)", (104, 'Containerization with Docker
and ECS', 'Packaging applications for scalable cloud deployment.'))
```

```
    conn.commit()
    conn.close()
    print("Database initialized successfully.")

# Initialize database when the app starts
init_db()

# --- API ENDPOINTS ---

@app.route('/', methods=['GET'])
def index_redirect():
    """Redirects the root URL to the courses API endpoint."""
    return redirect('/api/courses')

def dict_factory(cursor, row):
    d = {}
    for idx, col in enumerate(cursor.description):
```

```
        d[col[0]] = row[idx]
    return d


@app.route('/api/courses', methods=['GET'])
def get_courses():
    """API endpoint to fetch a list of all courses."""
    conn = get_db_connection()
    conn.row_factory = dict_factory
    courses = conn.execute("""
        SELECT
            C.id, C.title, C.description, U.name AS instructor_name
        FROM
            Courses C
        JOIN
            Users U ON C.instructor_id = U.id
    """).fetchall()
    conn.close()
```

```
    course_list = [dict(row) for row in courses]
    return jsonify(course_list)


@app.route('/api/course/<int:course_id>', methods=['GET'])
def get_course_detail(course_id):
    """
    API endpoint to fetch detailed course information, its lessons,
    and check enrollment status for a given user.
    """
    conn = get_db_connection()
    conn.row_factory = dict_factory

    # Get user_id from query parameters (used by frontend to check status)
    user_id = request.args.get('user_id', type=int)
```

```
# Fetch Course Details
course = conn.execute("""
    SELECT
        C.id, C.title, C.description, U.name AS instructor_name
    FROM
        Courses C
    JOIN
        Users U ON C.instructor_id = U.id
    WHERE C.id = ?
""", (course_id,)).fetchone()

if course is None:
    conn.close()
    return jsonify({"error": "Course not found"}), 404

# Check Enrollment Status (DBMS Query)
is_enrolled = False
```

```
if user_id:
    enrollment = conn.execute("""
        SELECT 1 FROM Enrollments
        WHERE user_id = ? AND course_id = ?
    """, (user_id, course_id)).fetchone()
    if enrollment:
        is_enrolled = True

# Fetch Lessons for the Course
lessons = conn.execute("""
    SELECT id, title, content
    FROM Lessons
    WHERE course_id = ?
    ORDER BY id
""", (course_id,)).fetchall()

conn.close()
```

```
    course_data = dict(course)
    course_data['lessons'] = [dict(row) for row in lessons]
    course_data['is_enrolled'] = is_enrolled # Add enrollment status to
response

    return jsonify(course_data)

@app.route('/api/enroll', methods=['POST'])
def enroll_user():
    """API endpoint to handle user enrollment into a course (the 'buy'
action)."""
    data = request.get_json()
    user_id = data.get('user_id')
    course_id = data.get('course_id')

    if not all([user_id, course_id]):
        return jsonify({"error": "Missing user_id or course_id"}), 400
```

```
    conn = get_db_connection()
    try:
        # Check if already enrolled
        check = conn.execute("SELECT 1 FROM Enrollments WHERE user_id = ? AND course_id = ?", (user_id, course_id)).fetchone()
        if check:
            conn.close()
            return jsonify({"message": "Already enrolled"}), 200

        # Perform the Enrollment (DBMS Insert)
        conn.execute("INSERT INTO Enrollments (user_id, course_id) VALUES (?, ?)", (user_id, course_id))
        conn.commit()
        return jsonify({"message": "Enrollment successful!"}), 201

    except sqlite3.IntegrityError:
```

```
        return jsonify({"error": "Enrollment failed due to database
constraint"}), 409
    except Exception as e:
        return jsonify({"error": str(e)}), 500
    finally:
        conn.close()


if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1', port=5000)
```

CHAPTER 5

SCREEN SHOTS

Fig 5.1 Login page

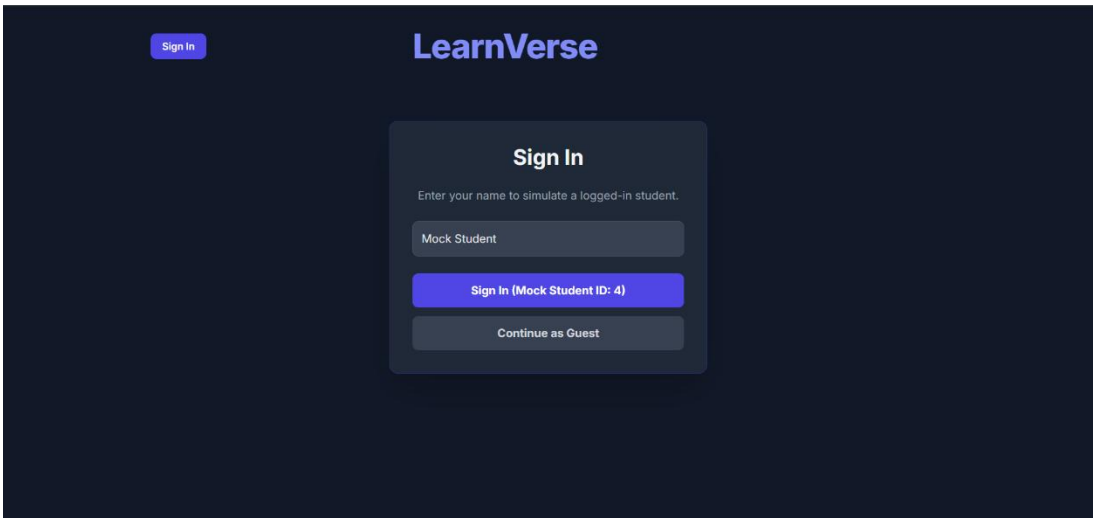

Fig 5.2 Course details

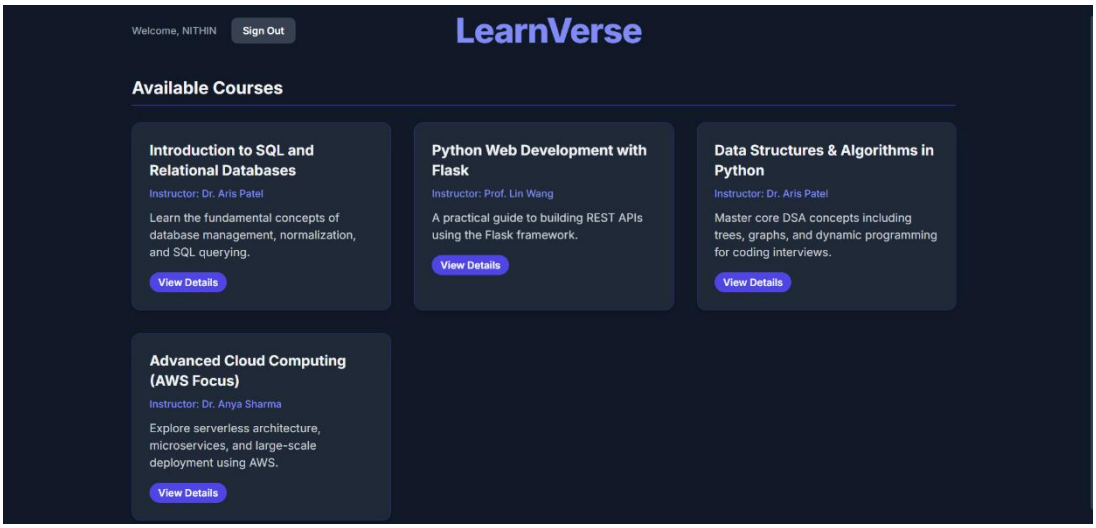

Fig 5.3 Payment Page

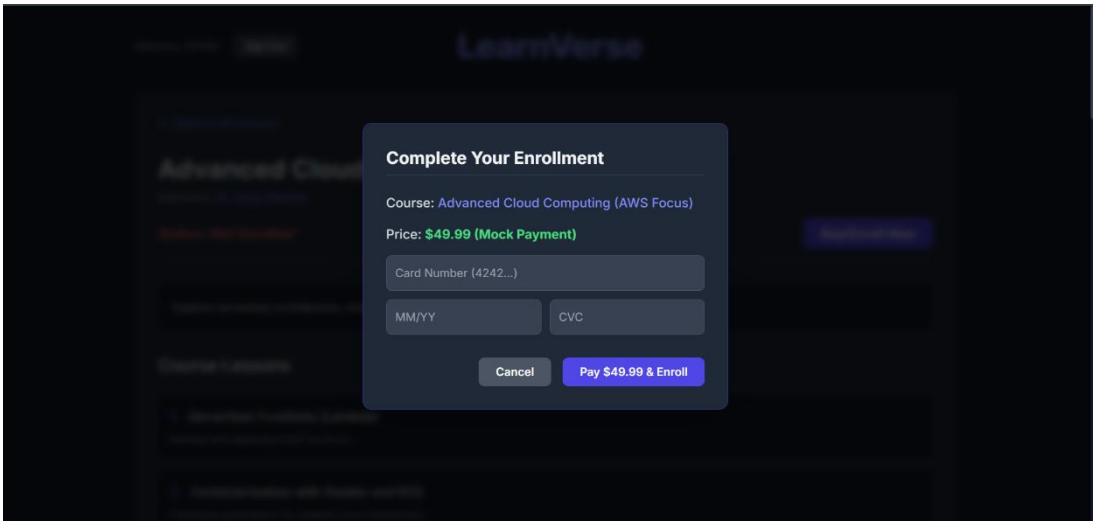

Fig 5.4 Booking creation

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

The **LearnVerse E-Learning Platform** mini-project successfully fulfills the primary objectives of a DBMS-focused application. We established a stable, relational database using **SQLite**, demonstrating key concepts like entity mapping and relationship management through the **Enrollments** table. The **Python Flask** backend acts as a robust data layer, handling **SQL queries**, transactional **INSERTS**, and providing structured data via API. The modern, responsive frontend provides a tangible interface for the user to interact with the underlying database logic, such as dynamically changing the button state from "Buy" to "View Course" based on enrollment status stored in the DBMS.

Future Enhancement

1. **Full User Authentication:** Implement a secure system (e.g., using a proper JWT token system) instead of the current mock login to manage multiple student accounts.

2. **Instructor Dashboard:** Create a separate view allowing instructors to add, edit, or delete their courses and lessons (full CRUD implementation for the Courses and Lessons tables).

3. **Progress Tracking:** Add a new table (Progress) to track student completion of lessons, adding another relational layer (e.g., user_id, lesson_id, completion_status).

REFERENCES

1. **SQL and SQLite:** https://www.w3schools.com/sql/
2. **Python Flask Framework:** https://flask.palletsprojects.com/
3. **Frontend Styling (Tailwind CSS):** https://tailwindcss.com/docs
4. **Python Language Reference:** https://www.learnpython.org