COURSE RECOMMENDATION WEB APP

"User interaction and progress-based recommendation app to tailor your custom learning path."

DOCUMENTATION

Name: KUSHAGRA PATHAK

Date: 16th May to 30th May 2025

TABLE OF CONTENT

1.Executive Summary

Course Recommendation Web App is an intelligent web-based platform designed to guide learners in their educational journey by recommending relevant courses based on their interests, selected topics, and learning progress. Built with a hybrid recommendation strategy, the system enhances personalization by dynamically adapting to user behavior — specifically course completion percentages and subject preferences.

The core problem addressed by this project is the **overload of unstructured online learning resources**, which often leads to decision fatigue and inefficient learning paths. By analyzing user interaction data and leveraging machine learning techniques such as **TF-IDF vectorization** and **cosine similarity**, the system creates a personalized, skill-driven learning roadmap for each user.

The application architecture follows a modular, containerized design using **Docker**, comprising three main services:

- A **React.js frontend** for intuitive user interaction
- A **Flask backend** responsible for handling APIs and recommendation logic
- A **PostgreSQL database** to manage user data and enrollment history

The system utilizes a cleaned and preprocessed version of the **Udemy Courses Dataset**, focusing on over 3,600 online courses across different domains. Users can select courses, update their progress, and receive suggestions for advanced tools or related skills (e.g., recommending **NumPy** or **Matplotlib** after completion of a **Pandas** course).

The project demonstrates strong alignment with modern e-learning trends and can serve as a blueprint for scalable edtech solutions. It showcases the synergy of content-based filtering, progress tracking, and real-time feedback loops to support intelligent, adaptive learning pathways.

2. Introduction

In today's rapidly evolving digital learning landscape, personalized recommendations are key to enhancing user engagement and skill development. The **Course Recommendation Web App** is designed to serve as an intelligent learning companion, offering tailored course suggestions to users based on their interests, past enrollments, and completion progress.

This application bridges the gap between vast online course repositories and individual learning paths by applying machine learning and content-based filtering techniques. It not only recommends relevant upskilling content but also adapts dynamically as users interact with the system, thereby guiding them along a personalized learning roadmap.

The project combines a responsive **React.js frontend**, a robust **Flask backend**, a **PostgreSQL** database, and an integrated **ML-based recommendation engine**. It is containerized using **Docker**, making it easy to deploy and scale across various environments.

This document outlines the architecture, data pipeline, machine learning model, deployment strategy, and potential future improvements for the Course Recommendation Web App. The aim is to present a comprehensive overview of the project to both technical and non-technical stakeholders.

3. System Overview

The **Course Recommendation Web App** is an end-to-end system that intelligently recommends online courses based on user behaviour, preferences, and course completion data. It combines a content-based filtering engine with an adaptive skill roadmap to enhance lifelong learning.

3.1 System Objectives

- Provide personalized course recommendations based on a user's selected interests and progress.
- Dynamically evolve suggestions as the user advances through courses.
- Enhance skill acquisition by surfacing related tools and technologies (e.g., recommending NumPy after Pandas).
- Ensure modular, scalable, and maintainable architecture using Docker and microservices.

3.2 High-Level Architecture

The application is composed of three core services, all containerized and orchestrated via Docker:

- **Frontend (React.js)**
  A responsive user interface allowing course selection, tracking progress, and displaying recommended courses. It communicates with the backend via REST APIs.
- **Backend (Flask API Server)**
  A RESTful backend that:
  - Manages user and course enrollment data.
  - Performs similarity-based recommendation using machine learning logic.
  - Interfaces with the PostgreSQL database.
  - Serves recommendation APIs to the frontend.
- **Database (PostgreSQL)**
  Relational database to persist:
  - User data
  - Course enrollments
  - Progress percentages
  - Course metadata

These services are linked via a shared Docker network (backend_net), and the database initializes with a SQL schema on container startup.

3.3 Recommendation Engine

The engine is powered by **content-based filtering** using **TF-IDF vectorization** and **cosine similarity**. It calculates similarity scores based on:

- Course title and subject (textual feature concatenation).
- User completion percentage as a weight modifier (e.g., Pandas at 80% affects weight more than a course at 20%).

The recommendation process involves:

1. **Course Similarity Matching**: Matching TF-IDF vectors across all courses against the user's completed/active course list.
2. **Weighted Aggregation**: Weighting similarity scores by progress percentage to prioritize heavily engaged topics.
3. **Subject Clustering**: Prioritizing subjects in which the user shows deep interest, then diversifying results across other areas.
4. **Dynamic Feed Composition**: A mix of:
    - Similar courses (weighted by engagement).
    - Advanced skills/tools within the same domain.
    - Courses from different subjects for exploratory learning.

3.4 Adaptive Skill Roadmap

A bonus feature includes roadmap expansion. For example:

- If a user is nearing completion of a "Pandas" course, the system will recommend advanced or complementary tools like "NumPy" or "Matplotlib."
- This is achieved by mapping course tags and content similarities, and adjusting recommendation weights dynamically.

4. Dataset Description

The course recommendation engine is trained on a real-world dataset sourced from **Udemy**, a popular online learning platform. This dataset provides comprehensive information about thousands of courses and serves as the foundation for generating personalized recommendations.

4.1 Dataset Source

- **Name**: Udemy Courses Dataset
- **Provider**: Kaggle (originally scraped from Udemy)
- **License**: For educational and research purposes

4.2 Dataset Overview

- **Total Records**: 3,672 unique courses
- **Format**: CSV (Comma-Separated Values)
- **Size**: ~600 KB

4.3 Key Columns Used

The following fields from the dataset were selected for processing and model input:

Column Name	Description
course_id	Unique identifier for each course
course_title	Title of the course (used for similarity comparison)
subject	Broad subject area (e.g., Business, Web Development)
num_subscribers	Total number of users subscribed to the course
num_reviews	Number of user-submitted reviews
price	Course price in USD (used to distinguish free vs paid)
is_paid	Boolean flag indicating whether the course is paid or free
level	Skill level (e.g., All Levels, Intermediate, Beginner)
content_duration	Total length of course content in hours
published_timestamp	When the course was first published (excluded from model training

4.4 Feature Engineering

To build a meaningful similarity metric:

- A new feature, combined_features, was created by concatenating:
    - **course_title + subject**
- This field was vectorized using **TF-IDF (Term Frequency–Inverse Document Frequency)**, which emphasizes unique and meaningful terms across courses.
- Stopwords and common language fillers were removed to improve feature quality.

4.5 Data Cleaning

The dataset underwent the following preprocessing steps:

- Removal of duplicates and null values.
- Normalization of text fields (lowercasing, punctuation removal).
- Filtering to retain only relevant fields for model input.
- Conversion of content_duration and num_subscribers to numerical types.

4.6 Limitations

While the dataset is rich and varied, there are inherent constraints:

- Course content details (e.g., full syllabus, instructor ratings) are not included.
- Real-time data is not available; it represents a snapshot.
- The dataset may not reflect the most recent offerings on Udemy.

5. Machine Learning Model

This system utilizes a **Content-Based Filtering** approach to generate personalized course recommendations by analyzing course content and user interaction history. The model focuses on semantic similarity and user progress weighting to prioritize relevant upskilling paths.

5.1 Recommendation Approach

Technique Used:

- **Content-Based Filtering**
- **Cosine Similarity** on **TF-IDF Vectorized** course metadata

Unlike collaborative filtering, which relies on user-to-user or item-to-item comparisons, this model builds personalized suggestions by comparing textual features of courses and adjusting relevance based on how much a user has progressed in related topics.

5.2 Feature Extraction

A key component is the creation of a **combined feature** per course:

```
combined_features = course_title + " " + subject
```

This field serves as the primary input to the **TF-IDF Vectorizer**, which converts text data into high-dimensional numeric vectors representing word importance.

```
tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(df['combined_features'])
```

5.3 Similarity Computation

To identify related courses, the system computes **cosine similarity** between course vectors:

```
cosine_similarity(tfidf_matrix[idx], tfidf_matrix).flatten()
```

This returns a score between 0 (completely dissimilar) and 1 (identical), indicating how closely a course matches another based on content.

5.4 Weighted Recommendations Based on Progress

The system enhances basic similarity scoring by incorporating **course completion percentage**, allowing it to weigh recommendations more heavily if the user has completed more of a relevant course:

```
weighted_sim += sim_scores * completion_dict.get(course, 0)
```

The scores are then normalized to prevent bias from varying numbers of completed courses.

5.5 Subject-Level Personalization

To further personalize results:

- User's **completed courses are grouped by subject**.
- The **average completion percentage per subject** is calculated.
- Recommendations prioritize subjects where the user has shown the most engagement.

5.6 Output Generation

The final recommendation list consists of:

- Top relevant courses based on user history and course similarity
- A mix of:
    - Courses from dominant subjects
    - Courses from secondary subjects
    - Courses from unrelated but potentially interesting domains (for diversity)

All results are deduplicated and shuffled to provide variety while maintaining relevance.

5.7 Advantages

- No need for cold-start collaborative filtering
- Personalized without requiring user comparisons
- Simple, scalable, and explainable model
- Adapts in real-time as user progress is updated

6. System Workflow

This section outlines the end-to-end flow of data and logic across various components of the Course Recommendation Web App. The system integrates user interaction, backend processing, and machine learning logic to deliver personalized course recommendations.

6.1 High-Level Workflow

User → Frontend (React.js) → Backend API (Flask) → ML Model & Database (PostgreSQL) → Response → Frontend Feed

6.2 Step-by-Step Breakdown

1. User Interaction (Frontend)

- User visits the web interface.
- They select an area of interest (e.g., "Data Science").
- User enrolls in a course (e.g., "Pandas") and tracks completion progress (e.g., 80%).

2. API Communication (Backend - Flask)

- Frontend sends the selected course, subject, and progress to the backend via REST API.
- Backend stores or updates this information in the PostgreSQL database (users, enrollments tables).

3. Data Retrieval & Preprocessing

- The backend queries user data and course metadata from the PostgreSQL database.
- Course features are preprocessed using TF-IDF vectorization on combined fields (course_title + subject).

4. Recommendation Generation

- ML model computes cosine similarity scores between completed and potential courses.
- Similarity scores are weighted by the user's course completion percentage.
- Subject engagement is also considered to prioritize relevant skill paths.
- Randomization is applied for variety, and final recommendations are selected.

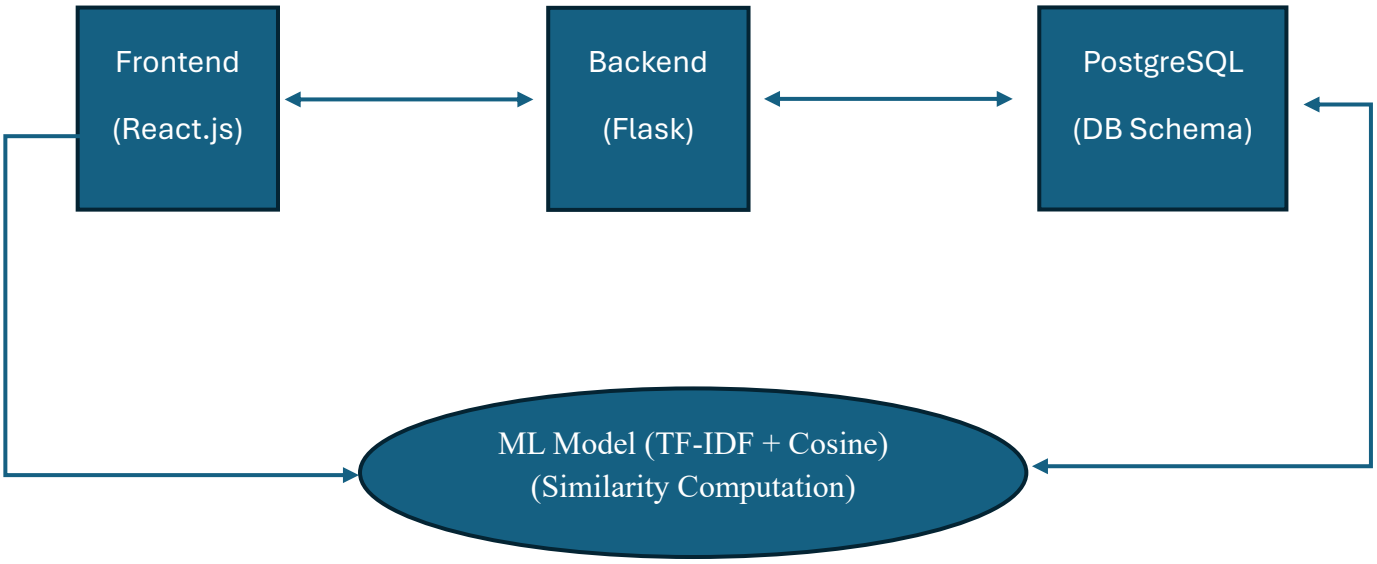
5. Returning Results

- The backend returns a ranked list of **12 recommended courses**.
- The frontend displays this list in a feed-style UI with course title, subject, duration, reviews, and more.

6.3 Real-Time Adaptation

- Every time a user updates their course progress, the system recalculates weights and updates the recommendation feed.
- This creates an **adaptive skill roadmap** that evolves as the user progresses.

6.4 Component Interaction Diagram



(Component Interaction Diagram)

7. Docker & Deployment

This section outlines the containerization and deployment strategy used to run the Course Recommendation Web App seamlessly across environments. The application is containerized using Docker and orchestrated using Docker Compose, making it easy to develop, test, and deploy.

7.1 Architecture Overview

The application is composed of three main services, each running in its own Docker container:

1. **Frontend** – React.js-based UI
2. **Backend** – Flask API with machine learning logic
3. **Database** – PostgreSQL for persistent storage

7.2 Docker Compose Structure

Here's a breakdown of the docker-compose.yml setup:

```
version: '3.8'


services:
  db:
    image: postgres:15
    container_name: postgres_container
    restart: always
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypass
      POSTGRES_DB: coursedb
    volumes:
      - db_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"
```

```
  networks:
    - backend_net

backend:
  build: ./Back
  container_name: backend_container
  depends_on:
    - db
  environment:
    DB_HOST: db
    DB_PORT: 5432
    DB_NAME: coursedb
    DB_USER: myuser
    DB_PASS: mypass
  ports:
    - "5000:5000"
  networks:
    - backend_net

frontend:
  build: ./Front
  container_name: frontend_container
  depends_on:
    - backend
  ports:
    - "3000:3000"
  networks:
    - backend_net
```

```
volumes:
  db_data:


networks:
  backend_net:
```

7.3 Steps to Deploy Locally

Prerequisites:

- Docker installed (Docker Desktop or Docker Engine)
- Docker Compose installed (v2 or above)

1. Clone the Repository:

git clone https://github.com/IcodeG00D/Course-Recommendation-Web-App.git

cd Course-Recommendation-Web-App

2. Launch the Application:

docker-compose up --build

🟢 The following services will start:

- React Frontend → http://localhost:3000
- Flask Backend → http://localhost:5000
- PostgreSQL → accessible on port 5432 (internal use)

8. Installation & Usage

This section guides you through setting up the project on your local machine and provides instructions for using the application effectively. Whether you're a developer looking to contribute or a user interested in exploring personalized course recommendations, this guide has you covered.

8.1 Prerequisites

Ensure the following software is installed on your machine:

- Docker
- Docker Compose
- Git (for cloning the repository)
- Modern web browser (e.g., Chrome, Firefox)

8.2 Cloning the Repository

```
git clone https://github.com/IcodeG00D/Course-Recommendation-Web-App.git

cd Course-Recommendation-Web-App
```

8.3 Environment Configuration

The system uses environment variables to connect to the database and run services. These are defined in the docker-compose.yml file:

```
DB_HOST: db

DB_PORT: 5432

DB_NAME: coursedb

DB_USER: myuser

DB_PASS: mypass
```

You may also create a .env file for overriding sensitive variables during production deployment.

8.4 Running the Application

To build and run all services together:

docker-compose up --build

- **Frontend** will be available at: http://localhost:3000
- **Backend API** will be running at: http://localhost:5000
- **PostgreSQL** is available internally at port 5432

To stop the application:

docker-compose down

8.5 Usage Workflow

1. **Start the Application**: Open http://localhost:3000 in a browser.
2. **Create a User**: Use the UI to register a new username.
3. **Select a Course**: Pick a course of interest (e.g., "Pandas").
4. **Track Progress**: Indicate how much of the course you've completed.
5. **View Recommendations**: Based on your progress, related skills (e.g., NumPy, Matplotlib) and advanced topics will appear in your recommendation feed.
6. **Repeat & Personalize**: As you interact more with the platform, recommendations become smarter and better tailored to your interests.

8.6 Admin & Developer Tips

- To check logs:

```
docker logs backend_container
docker logs frontend_container
```

- To access PostgreSQL shell:

```
docker exec -it postgres_container psql -U myuser -d coursedb
```

9. API Documentation

This section outlines the complete list of backend API endpoints provided by the Flask application for the Course Recommendation Web App. These endpoints handle user management, course enrollments, progress tracking, subject/course data retrieval, and recommendation logic.

Base URL

http://localhost:5000/

9.1 POST /store_user

Registers a new user.

Request Body:

```
{
 "name": "john_doe"
}
```

Response:

```
{
 "message": "User stored successfully"
}
```

9.2 POST /enroll_course

Enrolls a user into a course.

- **Request Body:**

```
{
 "user_name": "john_doe",
 "course_name": "Pandas for Data Analysis",
 "c_subject": "Data Science"
}
```

Response:

```
{
 "message": "Course enrolled successfully"
}
```

9.3 DELETE /unenroll_course

Removes an enrolled course for a user.

- **Request Body:**

```
{
  "user_name": "john_doe",
  "course_name": "Pandas for Data Analysis"
}
```

Response:

```
{
  "message": "Enrollment deleted successfully"
}
```

9.4 POST /update_progress

Updates course completion percentage for a specific enrollment.

- **Request Body:**

```
{
  "user_name": "john_doe",
  "course_name": "Pandas for Data Analysis",
  "progress": 80
}
```

Response:

```
{
  "message": "Progress updated successfully"
}
```

9.5 GET /enrollments/<user_name>

Fetches all enrollments for a given user.

- **Response:**


```
[
  {
    "course_title": "Pandas for Data Analysis",
    "subject": "Data Science",
    "progress": 80
  }
]
```

9.6 GET /merged-enrollments/<user_name>

Returns detailed enrollment information (with user and course metadata).

- **Response:**


```
[
  {
    "id": 1,
    "name": "john_doe",
    "course": "Pandas for Data Analysis",
    "subject": "Data Science",
    "progress": 80
  }
]
```

9.7 POST /recommend

Generates personalized course recommendations based on the active course and user's progress.

- **Request Body:**

```
{
 "user_name": "john_doe",
 "active_course_title": "Pandas for Data Analysis",
 "completion_percent": {
  "Pandas for Data Analysis": 80
 }
}
```

Response:

```
{
 "recommendations": [
  {
   "course_title": "NumPy for Beginners",
   "score": 0.87
  },
  ...
 ]
}
```

9.8 GET /subjects

Retrieves all unique course subjects from the dataset.

- **Response:**
- {
- "subjects": [
- "Data Science",
- "Business",

- "Web Development"
-]
- }

9.9 GET /top-courses/<subject>

Returns the top 20 most subscribed courses under a specific subject.

Example:
/top-courses/Data Science

Response:


```
{
 "top_courses": [
  {
   "course_title": "Pandas for Data Analysis",
   "published_timestamp": "2019-05-10T00:00:00Z",
   "num_reviews": 1200,
   "num_subscribers": 45000,
   "content_duration": 4.5,
   "subject": "Data Science"
  },
  ...
 ]
}
```


Status Codes Used

- 200 OK – Successful response.
- 201 Created – Resource created successfully.
- 400 Bad Request – Missing or invalid data.
- 404 Not Found – Resource not found (e.g., user or course).
- 500 Internal Server Error – Server-side error.

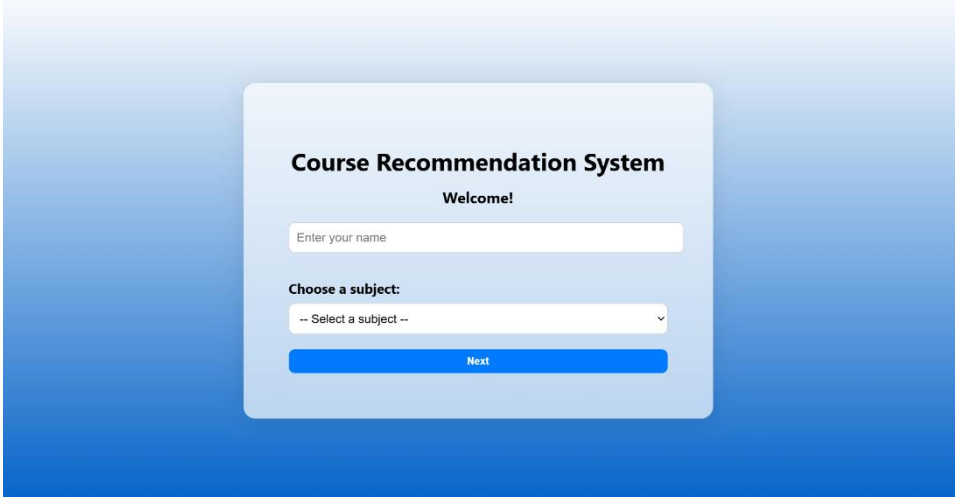
10. UI Screenshots

This section showcases visual representations of the frontend interface built using **React.js**. These screenshots provide insight into the core functionalities and user experience of the Course Recommendation Web App.

---

1. Home Page – Subject Selection

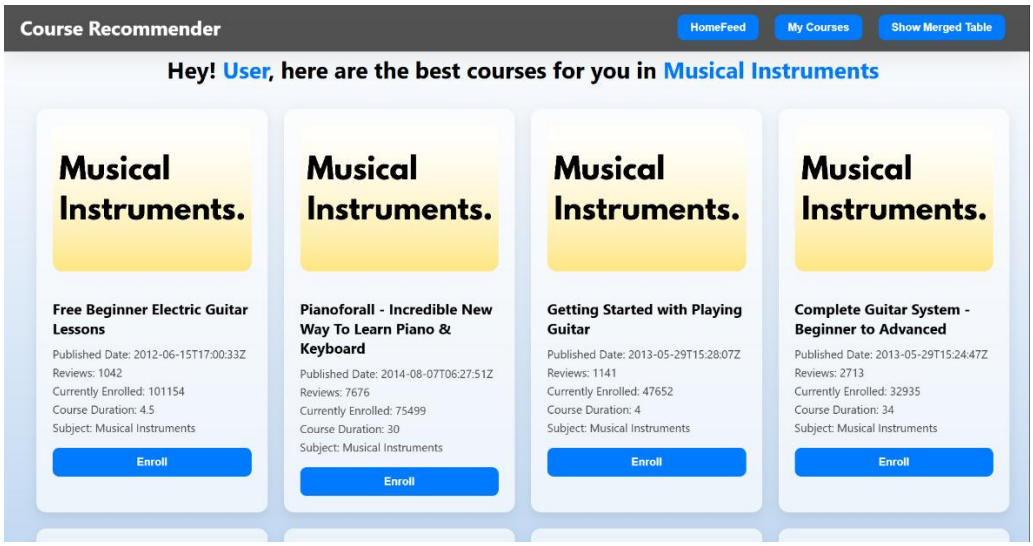
- Users can browse various subjects such as *Business Finanace*, *Web Development*, etc.
- Clicking on a subject fetches top trending courses via the /top-courses/<subject> API.



---

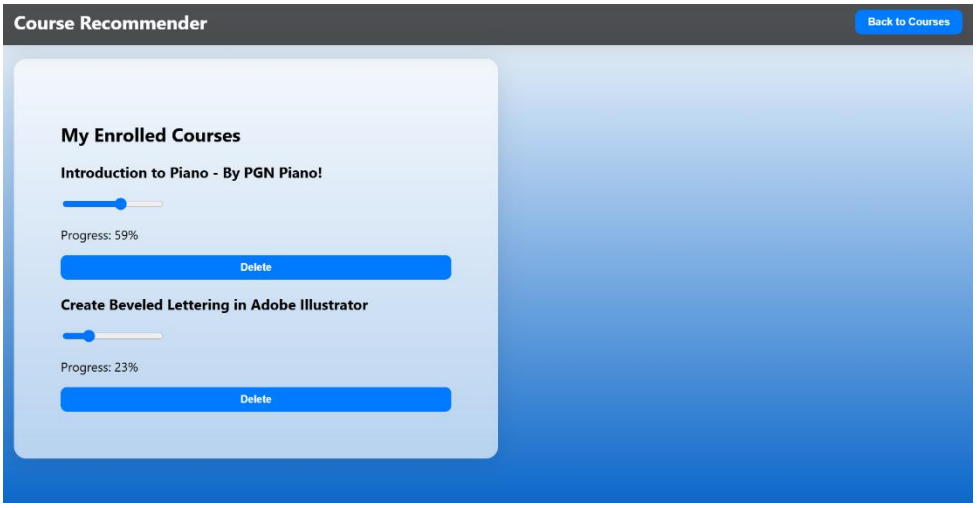
2. Course List View

- Displays a dynamic list of courses with metadata (duration, reviews, subscribers).
- Courses can be added to the user's profile via enrollment.



3. User Dashboard – Enrollments

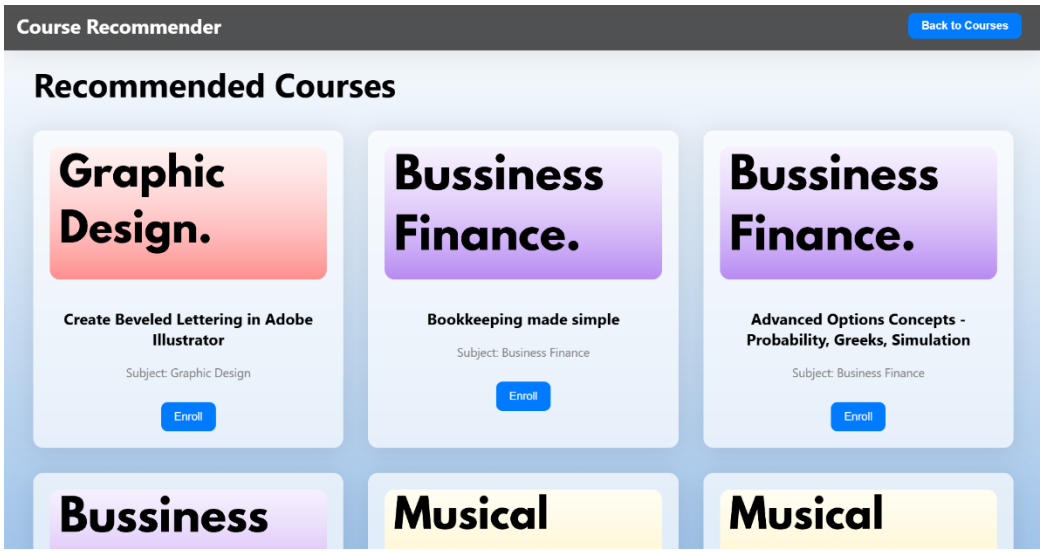
- Logged-in users can view their active enrollments.
- Each enrolled course shows progress and allows updates via sliders or dropdowns.
- Enrollments are fetched using /merged-enrollments/<user_name>.



---

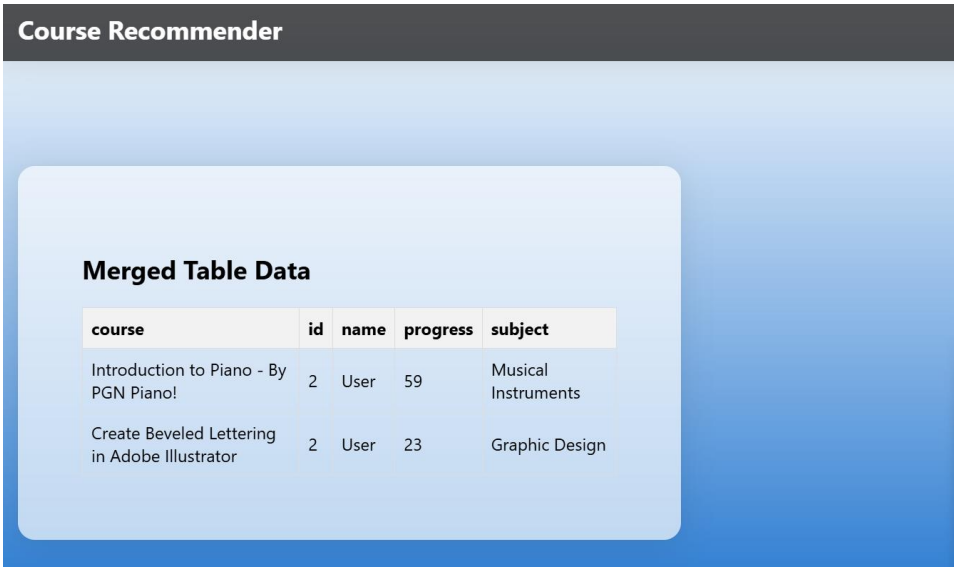
4. Recommendation Feed

- As the user completes a course (e.g., > 80%), related technologies or next-level skills (e.g., *NumPy*, *Matplotlib*) are recommended using /recommend.



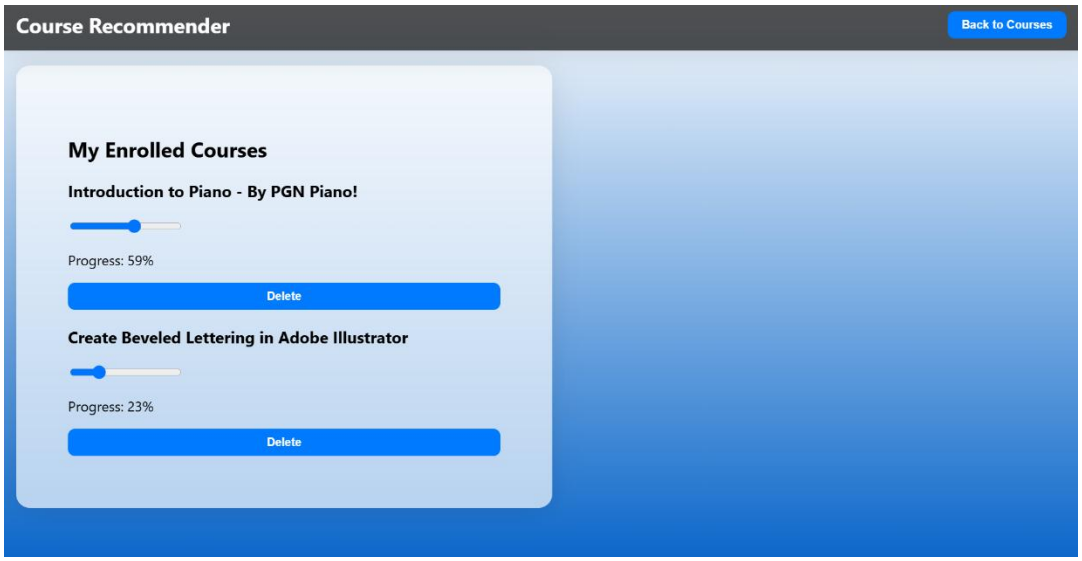
📌 5. Progress Tracker

- Interactive progress tracker allows users to update course completion.
- Backend updates through /update_progress.



📌 6. Enroll/Unenroll Controls

- UI buttons to enroll in or remove a course.
- Integrates seamlessly with /enroll_course and /unenroll_course.



These screens represent the main components of the UI. For a full demo, clone and run the application locally as described in the installation section.

11. Testing & Validation

Effective testing was carried out to ensure that the **Course Recommendation Web App** functions reliably across all components—backend, frontend, and database layers. Testing focused on correctness, integration, and stability of the overall system.

---

1. Backend Testing

Approach Used:

- Manual testing via frontend interface.
- Terminal-based testing using curl and console logging.
- Code-level assertions and print-debugging to trace request/response behavior.

Tested API Endpoints:

Endpoint	Method	Status	Purpose
/store_user	POST	OK	Stores user, handles duplicates gracefully
/recommend	POST	OK	Returns course recommendations based on progress
/top-courses/<subject>	GET	OK	Fetches top 20 popular courses per subject
/enroll_course	POST	OK	Records user's enrolled course
/unenroll_course	DELETE	OK	Removes enrollment based on username and course
/update_progress	POST	OK	Updates user progress for a course
/enrollments/<user_name>	GET	OK	Returns all course enrollments for a user
/merged-enrollments/<user_name>	GET	OK	Combines user and enrollment details for tracking

Validation Notes:

- Each API tested via live interaction with the frontend.
- Console outputs (e.g., debug logs) confirmed correct data flows.
- Error messages and HTTP status codes were verified for missing/invalid inputs.

---

2. Frontend Testing

Approach Used:

- Manual testing by simulating user interactions:
  - Registering user
  - Selecting subjects and enrolling in courses
  - Updating course progress
  - Viewing dynamic recommendations
- Verified that UI dynamically updated based on backend responses.

Validation Checks:

- Component state updates triggered correct API calls.
- Loading indicators and error feedback appeared as expected.
- Recommendation logic reflected course completion changes.

---

3. Dockerized Integration Testing

Environment Setup:

- Used Docker Compose to launch frontend, backend, and PostgreSQL containers.
- Verified container communication via Docker network.
- Ensured persistent database state using named volume.

Validation Highlights:

- Frontend could successfully connect to backend APIs within Docker.
- Environment variables were passed correctly across containers.
- Recommendation system worked end-to-end in containerized setup.

---

4. Edge Case Testing

Case	Result
Missing active_course_title in /recommend	Returns 400 Bad Request
Enroll request without user	Returns 404 Not Found
Update progress for non-enrolled course	Returns error message

Case	Result
Duplicate user entry	Gracefully handled
Unenroll non-existent course	Returns descriptive error

Summary

- **Testing Scope**: All critical paths and user flows verified.
- **Tools Used**: Browser, Terminal (curl/print logs), Docker logs.
- **Testing Confidence**: All core features functional in real-use simulation.
- **Status**: App is stable and ready for deployment in a controlled environment.

12. Future Improvements

While the **Course Recommendation Web App** delivers personalized learning experiences using course metadata and user engagement, there are several enhancements that could significantly improve the intelligence, scalability, and usability of the platform.

---

1. Enhanced Database Structure

- **Current Limitation**: The database only stores basic user info, enrollment, and progress.
- **Improvement Goal**: Expand the schema to include:
    - Course levels (beginner, intermediate, advanced)
    - Tag-based metadata (e.g., machine learning, visualization)
    - Skill tracking per user
    - Certification history
- **Benefit**: Supports more intelligent filtering and targeted course recommendations.

---

2. Roadmap-Based Learning Paths

- **Concept**: Allow users to choose a **learning goal or domain** (e.g., "Become a Data Analyst").
- **Implementation**:
    - Curate course sequences based on industry skills.
    - Provide milestone checkpoints and estimated timelines.
- **Outcome**: Transforms the app into a **career-oriented skill roadmap engine**, not just a recommender.

3. Hierarchical, Level-Based Recommendations

- **Current Logic**: Recommends similar courses using cosine similarity on course metadata.
- **Enhanced Flow**:
    - Track course **levels** (Basic, Intermediate, Advanced).
    - Use current completion level to **intelligently progress** user to the next challenge.

    - Example: If a user completes a "Basic Pandas" course → recommend "Intermediate Pandas", followed by "Advanced Data Manipulation with Pandas".

- **Result**: Promotes structured skill growth and avoids repeated beginner-level recommendations.

4. Certificate Generation

- **Feature**: Automatically generate **completion certificates** for users upon reaching a certain progress threshold (e.g., 90%+).
- **Technical Additions**:
    - PDF generation on the backend using libraries like ReportLab or WeasyPrint.
    - Store/download links in database.
    - Certificate validation page (e.g., /verify-certificate?user=xyz).
- **User Value**: Boosts motivation and gamifies learning. Certificates can be shared or added to resumes/LinkedIn.

5. Real-Time Feedback and User Reviews

- Add an optional review system:
    - Users rate and review courses after completion.
    - Help others choose more effective content.
    - Enhance recommendation engine with collaborative filtering down the line.

6. Admin Dashboard

- Admin-level UI for managing:
    - Course data (add/edit/remove)
    - Track active users, completion rates
    - Generate reports
- Adds transparency and operational control to the system.

7. Analytics and Usage Insights

- Show user dashboards with:
    - Weekly learning time
    - Courses completed vs. enrolled
    - Skill progression graphs
- Encourages habit-forming behavior by visualizing growth.

13. Conclusion

The **Course Recommendation Web App** is a modular, scalable, and intelligent platform designed to guide users through personalized learning journeys. By leveraging a hybrid recommendation system combining content-based filtering and user interaction history, the platform dynamically adjusts to each user's progress, interests, and goals.

The system's core strength lies in its seamless integration of modern technologies — from a responsive React.js frontend and Flask-based API backend to a robust PostgreSQL database and Dockerized microservices architecture. This setup ensures high portability, maintainability, and ease of deployment across development and production environments.

Throughout development, key focus areas included:

- Delivering relevant and level-appropriate course suggestions.
- Enabling interactive course progress tracking and adaptive feedback.
- Laying the foundation for skill-based roadmaps, certification, and real-time recommendations.

With proposed future enhancements such as level progression logic, certificate issuance, career-path roadmaps, and user analytics, this platform can evolve into a fully-fledged **learning assistant and career development tool**.

Ultimately, this project demonstrates how data-driven solutions can make education more engaging, personalized, and goal-oriented—aligning learning paths with real-world skills and aspirations.

14. References

This project utilizes multiple resources, datasets, libraries, and tools. Below are the key references that informed the design, implementation, and deployment of the Course Recommendation Web App:

- **Dataset:**
  - Udemy Courses Dataset — Andrew Mvd, Kaggle https://www.kaggle.com/datasets/andrewmvd/udemy-courses
- **Machine Learning & NLP:**
  - Scikit-learn: Machine Learning in Python — https://scikit-learn.org
  - TF-IDF Vectorization and Cosine Similarity Concepts — Manning, Raghavan, Schütze, *Introduction to Information Retrieval*, 2008
- **Frameworks & Libraries:**
  - Flask — Micro web framework for Python https://flask.palletsprojects.com
  - React.js — Frontend JavaScript library https://reactjs.org
  - Pandas & NumPy — Data manipulation and scientific computing https://pandas.pydata.org, https://numpy.org
  - psycopg2 — PostgreSQL adapter for Python https://www.psycopg.org
- **Database & Deployment:**
  - PostgreSQL — Open-source relational database https://www.postgresql.org
  - Docker — Containerization platform https://www.docker.com
- **Additional References:**
  - CORS Implementation in Flask — Flask-CORS Documentation https://flask-cors.readthedocs.io

15. Appendix

A. Database Schema

```
CREATE TABLE IF NOT EXISTS users (
   id SERIAL PRIMARY KEY,
   user_name VARCHAR(100)
);


CREATE TABLE enrollments (
   id SERIAL PRIMARY KEY,
   user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
   course VARCHAR(100),
   c_subject VARCHAR(100),
   progress INTEGER DEFAULT 0
);
```

B. Sample Environment Variables

```
DB_HOST=db
DB_PORT=5432
DB_NAME=coursedb
DB_USER=myuser
DB_PASS=mypass
```

C. Docker Compose Configuration (excerpt)

```
services:
  db:
    image: postgres:15
    container_name: postgres_container
    restart: always
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypass
```

```
    POSTGRES_DB: coursedb
  volumes:
    - db_data:/var/lib/postgresql/data
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql
  ports:
    - "5432:5432"
  networks:
    - backend_net

backend:
  build: ./Back
  container_name: backend_container
  depends_on:
    - db
  environment:
    DB_HOST: db
    DB_PORT: 5432
    DB_NAME: coursedb
    DB_USER: myuser
    DB_PASS: mypass
  ports:
    - "5000:5000"
  networks:
    - backend_net

frontend:
  build: ./Front
  container_name: frontend_container
  ports:
```

```
    - "3000:3000"
  depends_on:
    - backend
  networks:
    - backend_net
volumes:
 db_data:


networks:
 backend_net:
```

D. Example API Request Payload (POST /recommend)

```
{
 "user_name": "johndoe",
 "active_course_title": "Pandas for Data Analysis",
 "completion_percent": {
  "Pandas for Data Analysis": 80,
  "Intro to Python": 100
 }
}
```

E. Common Error Responses

HTTP Code	Error Message	Possible Cause
400	Missing data	Required fields not provided
404	User not found	User does not exist in database
500	Server error / Database error	Internal server or DB connection failure

----------THANK YOU----------