### VISVESVARAYA TECHNOLOGICAL UNIVERSITY

#### Jnana Sangama, Belagavi-590010



DBMS MINI PROJECT REPORT ON

### “LEARNING PROGRESS TRACKER”

Submitted in partial fulfillment for the requirements for the fifth semester

#### BACHELOR OF ENGINEERING

#### IN

**COMPUTER SCIENCE AND ENGINEERING**

For the Academic Year 2023-2024 Submitted by:

#### RACHANA B 1MV21CS076

**RAHUL JANGRA 1MV21CS078**

**SATHVIK N G 1MV21CS091**

**SHRIYA HEGDE 1MV21CS098**

Under the guidance of:

#### Dr. Suma Swamy

Professor, Department of CSE

Sir M. Visvesvaraya Institute of Technology, Bengaluru



#### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

HUNASAMARANAHALLI, BENGALURU-562157

## SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY



Krishnadevaraya Nagar, International Airport Road, Hunasmaranahalli, Bengaluru – 562157

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# CERTIFICATE

It is certified that the **DBMS Mini Project work** entitled "**LEARNING PROGRESS TRACKER**" is carried out by **RACHANA B (1MV21CS076), RAHUL JANGRA (1MV21CS078), SATHVIK N G (1MV21CS091), SHRIYA HEGDE (1MV21CS098)** bonafide students of **Sir M Visvesvaraya Institute of Technology** in partial fulfillment for the 5th semester for the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of the **Visvesvaraya Technological University, Belagavi** during the academic year **2023-2024**. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the course of Bachelor of Engineering.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name & Signature of Guide |  | Name & Signature of HOD |  | Name & Signature of Principal |
| **Dr . Suma Swamy**  Prof & Internal Guide Dept. of CSE, Sir MVIT  Bengaluru - 562157 |  | **Dr. Anitha T N** HOD, Dept. of CSE, Sir MVIT  Bengaluru - 562157 |  | **Dr. Rakesh S G**  Principal, Sir MVIT  Bengaluru – 562157 |

External Examination:

Name of Examiner Signature with Date

1)

2)

## DECLARATION

We hereby declare that the entire project work embodied in this dissertation has been carried out by us and no part has been submitted for any degree or diploma of any institution previously.

Place: Bengaluru Date:

Signature of Students:

|  |  |  |
| --- | --- | --- |
| RACHANA B (1MV21CS076) |  | RAHUL JANGRA (1MV21CS078) |
| SATHVIK N G (1MV21CS091) |  | SHRIYA HEGDE (1MV21CS098) |

## ACKNOWLEDGMENT

It gives us immense pleasure to express our sincere gratitude to the management of **Sir M. Visvesvaraya Institute of Technology,** Bengaluru for providing the opportunity and the resources to accomplish our project work in their premises.

On the path of learning, the presence of an experienced guide is indispensable and we would like to thank our guide **Dr Suma Swamy,** Professor, Department of CSE, for her invaluable help and guidance.

Heartfelt and sincere thanks to **Dr. Anitha T N,** HoD, Department of CSE, for his suggestions, constant support and encouragement.

We would also like to convey our regards to **Dr. Rakesh S G,** Principal, Sir MVIT for providing us with the infrastructure and facilities needed to develop our project.

We would also like to thank the staff of Department of Computer Science and Engineering and lab-in-charges for their co-operation and suggestions. Finally, we would like to thank all our friends for their help and suggestions without which completing this project would not have been possible.

|  |  |  |
| --- | --- | --- |
|  | RACHANA B | 1MV21CS076 |
| - | RAHUL JANGRA | 1MV21CS078 |
| - | SATHVIK N G | 1MV21CS091 |
| - | SHRIYA HEGDE | 1MV21CS098 |

## ABSTRACT

In the realm of learning and skill development, the "Learning Progress Tracker" is a user-friendly platform designed to seamlessly connect with various courses and topics, aiding learners on their educational journey. It efficiently records and analyzes users' advancement, allowing them to effortlessly track achievements, identify areas for improvement, and set personalized learning goals.

Key features include a dynamic course-topic mapping system, real-time progress updates, and an intuitive interface. Through insightful analytics, learners gain valuable insights into their strengths, weaknesses, and overall proficiency. Gamification elements and achievement milestones further boost user motivation, fostering a sense of accomplishment.

The Learning Progress Tracker is set to revolutionize education by promoting personalized learning paths, encouraging continuous improvement, and empowering individuals to unlock their full potential.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| Sl No. | Chapters | Page No. |
| **1** | **Introduction** | 1-2 |
|  | 1.1 Introduction to LPT  1.2 Need For LPT  1.3 Aim / Objectives of This Project | 1  1  2 |
| **2** | **Front End and Back End** | 3-4 |
|  | 2.1 Python (Back-End)  2.2 Django  2.3 PostgreSQL  2.4 HTML(Front-End)  2.5 CSS  2.6 JavaScript  2.7 React.js | 3  3  3  4  4  4  4 |
| **3** | **Specifications** | 5-6 |
|  | 3.1 Hardware Requirements  3.2 Software Requirements  3.3 Functional Requirements  3.4 Non-Functional Requirements | 5  5  6  6 |
| **4** | **System Design** | 7 |
|  | 4.1 UML Diagram  4.2 ER Diagram | 7  7 |
| **5** | **Conceptual Design** | 10 |
|  | 5.1 Create Commands  5.2 Insert Commands and Data  5.3 SQL Queries | 10  11  12 |
| **6** | **Frontend design, connectivity and implementation** | 15 |
|  | 6.1 Frontend Design and User Interface (UI) Description  6.1.1 Courses Module:  6.1.2 Course List Module:  6.1.3 Profile Module  6.2 Backend Implementation and Connectivity with Frontend  6.2.1 Models Explanation  6.2.2 Views Explanation  6.2.3 Connectivity with Frontend: | 15  15  15  15  16  16  17  19 |
| **7** | **Source Code** | 20-26 |
| **8** | **Results, Discussion & Screenshots** | 27 |
| **9** | **Conclusion and Scope** | 28 |
|  | **References & Weblinks** | 29 |

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Fig. No. | Description | Page No. |
| 4.1 | ER Diagram | 7 |
| 4.2 | UML DIAGRAM | 8 |
| 8.1 | Course List Page | 26 |
| 8.2 | Course Page | 26 |
| 8.3 | Profile Page | 27 |
| 8.4 | Login and Registration Page | 27 |
| 8.5 | Before Trigger | 28 |
| 8.6 | After Trigger | 28 |
| 8.7 | Print progress summary report | 28 |

**CHAPTER – 1**

**INTRODUCTION**

* 1. **INTRODUCTION TO LEARNING PROGRESS TRACKER:**

In the world of learning, our "Learning Progress Tracker" is a helpful tool that easily connects with different courses and topics. It helps learners by keeping track of how they're doing, letting them see what they've achieved, find areas to improve, and set personal learning goals. With a focus on users, a flexible course-topic system, and quick progress updates, the Learning Progress Tracker makes learning more enjoyable. It provides useful insights on strengths and weaknesses, encouraging continuous improvement and motivating learners on their unique educational paths.

* 1. **NEED FOR LEARNING PROGRESS TRACKER:**

In the dynamic landscape of modern education, the need for a Learning Progress Tracker (LPT) is paramount. Learners today require a tool that goes beyond traditional approaches, providing personalized experiences and efficient learning paths. The LPT addresses these needs by offering a user-centric platform that allows individuals to track their progress, set personalized learning goals, and receive real-time updates. This not only streamlines the learning process but also caters to the diverse strengths and weaknesses of each learner, fostering a more engaging and effective educational journey. In essence, the Learning Progress Tracker becomes an indispensable companion, meeting the evolving demands of learners in their pursuit of knowledge and skill development.

* 1. **AIM AND OBJECTIVES OF THE PROJECT:**

**Aim:**

The primary aim of the Learning Progress Tracker (LPT) project is to revolutionize the educational experience by providing a user-friendly and personalized tool that empowers learners to efficiently track and enhance their individual progress across various courses and topics.

**Objectives:**

* Personalized Learning Paths: To develop a user-centric approach that tailors learning experiences based on individual preferences, strengths, and weaknesses.
* Efficient Progress Tracking: To implement a dynamic course-topic mapping system and real-time progress updates to streamline the learning process, allowing users to easily monitor their achievements and identify areas for improvement.
* User-Friendly Interface: To design and implement an intuitive and user-friendly interface that ensures easy navigation, making the Learning Progress Tracker accessible to users with varying levels of technological proficiency.
* Adaptability and Scalability: To ensure that the Learning Progress Tracker is adaptable to different educational settings and scalable to accommodate future enhancements and expansions.

**CHAPTER - 2**

**FRONTEND AND BACKEND**

**BACKEND TECHNOLOGIES:**

**2.1 PYTHON:**

Python serves as the foundational programming language for the Learning Progress Tracker project, offering versatility and readability. Renowned for its clean syntax and extensive library support, Python facilitates efficient backend development, data manipulation, and algorithm implementation, ensuring a robust and scalable foundation for the entire system.

**2.2 DJANGO:**

Django, a high-level Python web framework, plays a pivotal role in simplifying the development of complex web applications. With its built-in features for handling routing, authentication, and database interactions, Django accelerates the development process, allowing for rapid iteration and maintaining a structured, maintainable codebase.

**2.3 POSTGRESQL:**

PostgreSQL, a powerful open-source relational database management system, is chosen for its reliability and extensibility. It provides a robust storage solution for user data, progress records, and system configurations. With support for complex queries and transactions, PostgreSQL ensures data integrity and efficient retrieval, crucial for the Learning Progress Tracker's dynamic functionalities.

**FRONTEND TECHNOLOGIES:**

**2.4 HYPERTEXT MARKUP LANGUAGE (HTML):**

HTML forms the structural foundation of the Learning Progress Tracker's frontend. It defines the logical structure of web pages, encompassing headings, paragraphs, links, and other essential elements. HTML ensures a semantically meaningful presentation of content, allowing seamless integration with other technologies and facilitating the creation of an accessible and well-organized user interface.

**2.5 CASCADING STYLE SHEETS (CSS):**

CSS (Cascading Style Sheets) complements HTML by providing the styling and visual presentation of the Learning Progress Tracker. It enables the customization of colors, layouts, fonts, and overall aesthetics, ensuring a visually appealing and consistent user interface. CSS contributes to the project's responsiveness and enhances the user experience by creating an engaging and polished design. Its cascading nature allows for efficient styling, promoting maintainability and flexibility in adapting to varying screen sizes and devices.

**2.6 JAVASCRIPT:**

JavaScript, the language of the web, is instrumental in creating dynamic and interactive user interfaces. Leveraging JavaScript allows for real-time updates, responsive design elements, and asynchronous communication with the backend. Its versatility enhances the overall user experience by introducing client-side interactivity to the Learning Progress Tracker.

**2.7 REACT JS:**

React JS, a JavaScript library for building user interfaces, brings a component-based architecture to the frontend. Its declarative approach simplifies UI development, promotes code reusability, and enhances the overall project maintainability. React JS ensures a fast and interactive user experience, crucial for the Learning Progress Tracker's dynamic and personalized functionalities.

**CHAPTER - 3**

**SPECIFICATIONS**

**3.1 HARDWARE REQUIREMENTS:**

1. Server

- Processor: Dual-core processor or higher

- RAM: 4 GB or more

- Storage: 20 GB or more

- Network: 1 Gbps Ethernet or faster

2. Database Server (PostgreSQL)

- Processor: Dual-core processor or higher

- RAM: 8 GB or more

- Storage: 50 GB or more

- Network: 1 Gbps Ethernet or faster

3. Client (for accessing the application)

- Processor: Dual-core processor or higher

- RAM: 4 GB or more

- Browser: Latest versions of Chrome, Firefox, Safari, or Edge

**3.2 SOFTWARE REQUIREMENTS:**

1. Operating System

- Server: Linux (Ubuntu, CentOS) recommended

- Development: Windows, macOS, or Linux

2. Python (Back-End)

- Version 3.6 or above

- Django web framework

3. PostgreSQl

- Version 10 or above

4. Front-End

- HTML5, CSS3 for structure and style

- JavaScript for client-side interactivity

- React.js for building interactive user interface

**3.3 FUNCTIONAL REQUIREMENTS:**

1. User Authentication

- Users should be able to register, log in, and log out.

2. Learning Tracker

- Users can track their learning progress.

- Ability to add, edit, and delete learning activities.

3. Dashboard

- Personalized dashboard displaying learning progress and achievements.

4. Reporting

- Generate reports on learning activities and progress.

5. User Roles

- Different roles (admin, regular user) with corresponding permissions.

6. Search and Filter

- Users can search and filter their learning activities.

**3.4 NON-FUNCTIONAL REQUIREMENTS:**

1. Performance

- The system should handle a concurrent user load of at least 100 users.

2. Security

- Use HTTPS for secure communication.

- Implement secure coding practices to prevent common web vulnerabilities.

3. Usability

- The user interface should be intuitive and user-friendly.

4. Compatibility

- The application should work on major browsers (Chrome, Firefox, Safari, Edge).

5. Documentation

- Well-documented codebase.

- User and administrator manuals.

6. Testing

- Comprehensive unit tests, integration tests, and user acceptance tests.

7. Deployment

- Easy deployment process for updates and new features

**CHAPTER – 4**

**SYSTEM DESIGN**

**4.1 UML DIAGRAM:**

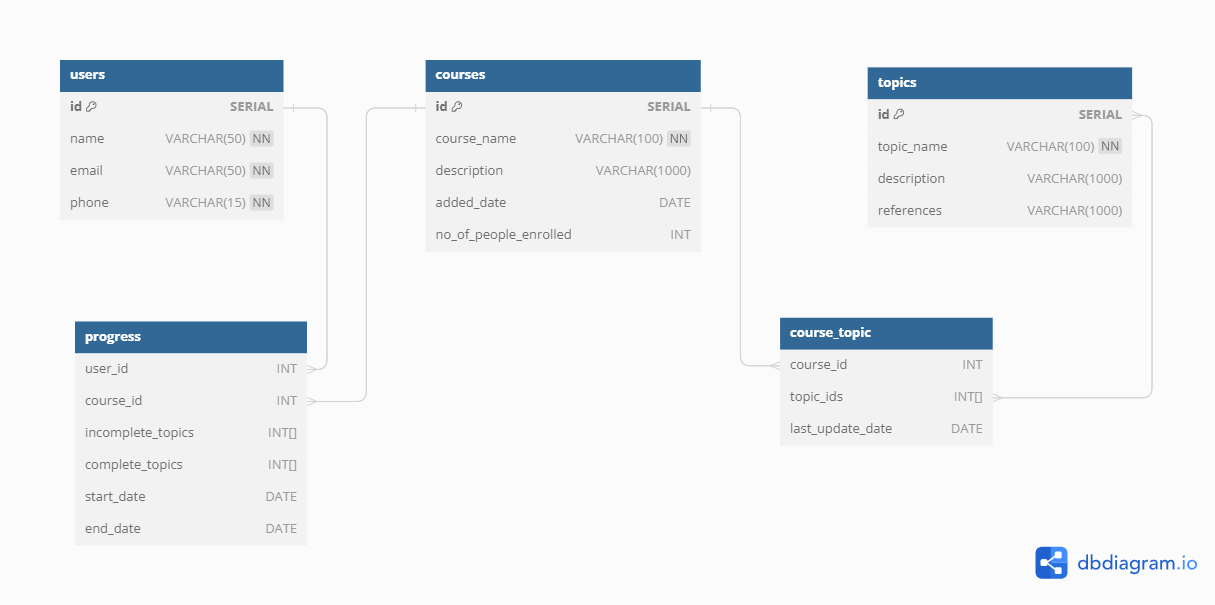
****

Fig 4.1 UML Diagram for Learning Progress Tracker

The UML diagram provided was created using the tool **dbdiagram.io**

* **Users Table**
* Each user has a unique identifier ("id"), a name, an email, and a phone number.
* **Courses Table**
* Each course is identified by a unique identifier ("id").
* It has a name, an optional description, an added date, and a count of people enrolled (with a default of 0).
* **Topics Table**
* Each topic is identified by a unique identifier ("id").
* It has a name, an optional description, links, and references.
* **Course-Topic Table**
* Courses and topics are related through a separate table called "course\_topic."
* This table keeps track of which courses include which topics.
* It has a reference to the course's id ("course\_id") and the topic's id ("topic\_id").
* Additionally, there is a field for the last update date.
* **Progress Table**
* Tracks the progress of users in courses.
* It has a reference to the user's id ("user\_id") and the course's id ("course\_id").
* There are arrays to keep track of completed and incomplete topics.
* It also records the start and end dates for a user's progress in a course.

**4.2 ENTITY – RELATIONSHIP (ER) DIAGRAM:**

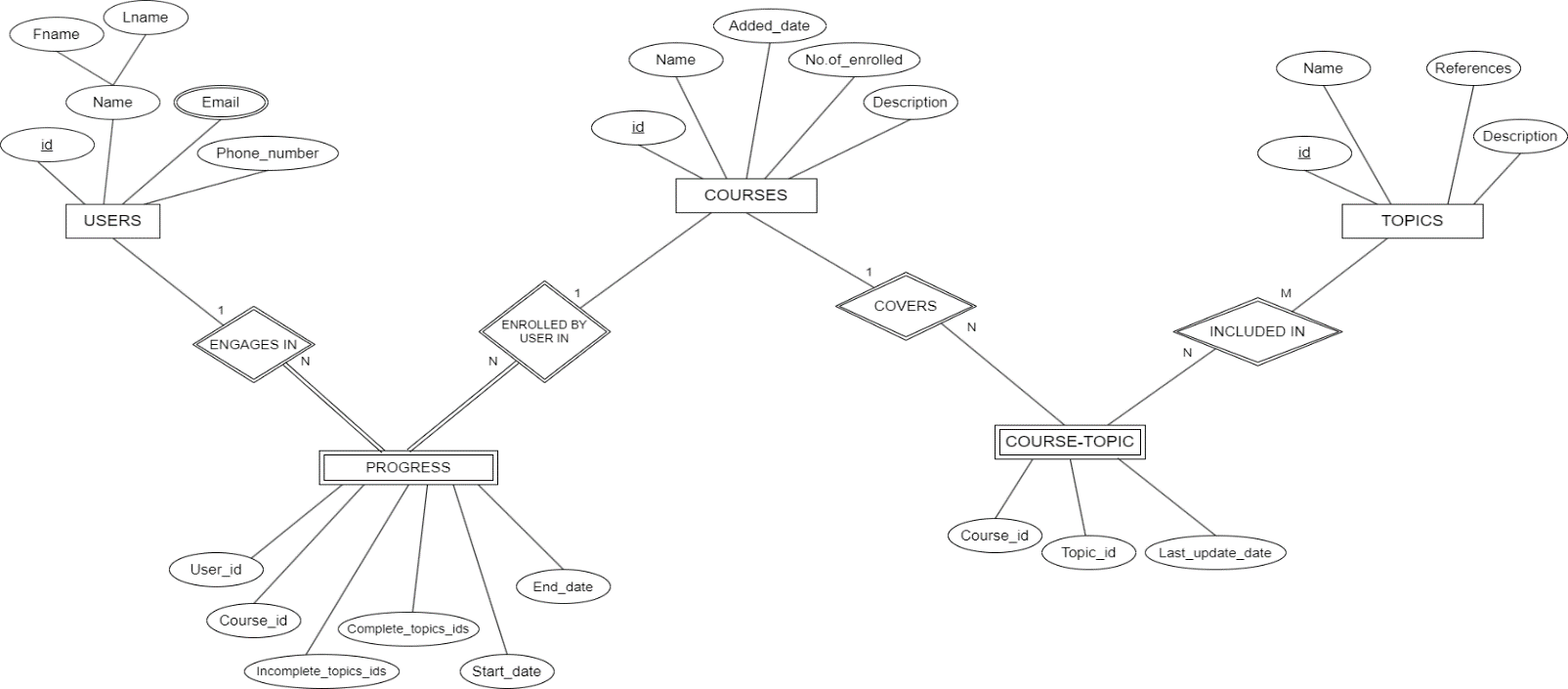


Fig 4.2 ER Diagram for Learning Progress Tracker

The ER diagram provided was created using the tool **draw.io**

|  |  |
| --- | --- |
| 1. **Users Entity:**  * Type: Strong Entity * Attributes: * "id" (Primary Key, Serial) * "name" (Unique, Not Null) * "email" (Not Null) * "phone" (Not Null) | 1. **Courses Entity:**  * Type: Strong Entity * Attributes: * "id" (Primary Key, Serial) * "course\_name" (Not Null) * "description" * "added\_date" (Date) * "no\_of\_people\_enrolled" (Default: 0) |
| 1. **Topics Entity:**  * Type: Strong Entity * Attributes: * "id" (Primary Key, Serial) * "topic\_name" (Not Null) * "description" * "links" * "references" | 1. **Progress Entity:**  * Type: Weak Entity * Attributes: * "user\_id" (Foreign Key referencing "users"."id") * "course\_id" (Foreign Key referencing "courses"."id") * "incomplete\_topics" (Array of integers, default: empty array) * "complete\_topics" (Array of integers, default: empty array) * "start\_date" (Date) * "end\_date" (Date) |
| 1. **Course-Topic Entity:**  * Type: Weak Entity * Attributes: * "course\_id" (Foreign Key referencing "courses"."id") * "topic\_id" (Foreign Key referencing "topics"."id") * "last\_update\_date" (Date) |  |

**Relationships:**

1. **Users-Progress Relationship:**

* **Type**: Identifying Relationship
* **Cardinality Ratio**: 1:N (One user can have progress in multiple courses, and each course can have progress tracked by multiple users).
* **Participation Constraints**: Partial Participation on Users side (Not every user must have progress).Total Participation on Progress side (Every progress must be associated with a user).

1. **Courses-Progress Relationship:**

* **Type**: Identifying Relationship
* **Cardinality Ratio**: 1:N (One course can have progress tracked by multiple users, and each user can have progress in multiple courses).
* **Participation Constraints**: Partial Participation on Courses side (Not every course must have progress tracked).Total Participation on Progress side (Every progress must be associated with a course).

1. **Courses-Course-Topic Relationship:**

* **Type**: Identifying Relationship
* **Cardinality Ratio**: 1:N (One course can have multiple topics, and each topic can be associated with multiple courses).
* **Participation Constraints**: Partial Participation on Courses side (Not every course must be associated with a topic).Partial Participation on Course-Topic side (Not every topic must be associated with a course).

1. **Topics-Course-Topic Relationship:**

* **Type**: Identifying Relationship
* **Cardinality Ratio:** M:N (One topic can be associated with multiple courses, and each course can have multiple topics).
* **Participation Constraints**: Partial Participation on Topics side (Not every topic must be associated with a course).Partial Participation on Course-Topic side (Not every course must be associated with a topic).

**CHAPTER – 5**

**CONCEPTUAL DESIGN**

**5.1 CREATE COMMANDS:**

-- Create user table

CREATE TABLE USERS (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL,

email VARCHAR(50) UNIQUE NOT NULL,

phone VARCHAR(15) UNIQUE NOT NULL

);

-- Create course table

CREATE TABLE COURSES (

id SERIAL PRIMARY KEY,

course\_name VARCHAR(100) NOT NULL,

description VARCHAR(1000),

added\_date DATE,

no\_of\_people\_enrolled INT DEFAULT 0

);

-- Create topic table

CREATE TABLE TOPICS (c

id SERIAL PRIMARY KEY,

topic\_name VARCHAR(100) NOT NULL,

description VARCHAR(1000),

reference VARCHAR(1000)

);

-- Create course\_topic table with explicit primary and foreign keys

CREATE TABLE COURSE\_TOPIC (

course\_id INT REFERENCES courses(id) ON DELETE CASCADE,

topic\_ids INT[] DEFAULT '{}'::INT[],

last\_update\_date DATE

);

-- Create progress table with explicit primary and foreign keys

CREATE TABLE PROGRESS (

user\_id INT REFERENCES users(id) ON DELETE CASCADE,

course\_id INT REFERENCES courses(id) ON DELETE CASCADE,

incomplete\_topics INT[] DEFAULT '{}'::INT[],

complete\_topics INT[] DEFAULT '{}'::INT[],

start\_date DATE,

end\_date DATE

)

**5.2 INSERT COMMANDS AND DATA**

-- Inserting users

INSERT INTO USERS (name, email, phone) VALUES

('Rahul Jangra', 'rahul@example.com', '123-456-7890'),

('Sathvik N G', 'sathvik@example.com', '987-654-3210');

-- Inserting courses

INSERT INTO COURSES (course\_name, description, added\_date) VALUES

('Python', 'Learn Python programming language.', '2024-02-19'),

('SQL', 'Learn SQL for database management.', '2024-02-19');

-- Inserting topics

INSERT INTO TOPICS (topic\_name, description, reference) VALUES

('Basic Syntax', 'Introduction to Python syntax.', 'https://docs.python.org/3/tutorial/introduction.html'),

('Variables and Datatypes', 'Understanding variables and data types in Python.', 'https://realpython.com/python-variables/'),

('Loops and Conditionals', 'Learn about loops and conditional statements in Python.', 'https://realpython.com/python-conditional-statements/'),

('Functions', 'Introduction to functions in Python.', 'https://realpython.com/defining-your-own-python-function/'),

('Object Oriented Programming', 'Understanding OOP concepts in Python.', 'https://realpython.com/python3-object-oriented-programming/'),

('Data Definition Language', 'Learn SQL DDL commands.', 'https://www.postgresql.org/docs/current/ddl.html'),

('Data Manipulation Language', 'Learn SQL DML commands.', 'https://www.postgresql.org/docs/current/dml.html'),

('Joins', 'Understanding SQL join operations.', 'https://www.w3schools.com/sql/sql\_join.asp'),

('Aggregate Functions', 'Learn about SQL aggregate functions.', 'https://www.postgresql.org/docs/current/functions-aggregate.html');

-- Inserting data into COURSE\_TOPIC for Python

INSERT INTO COURSE\_TOPIC (course\_id, topic\_ids, last\_update\_date) VALUES

(1, '{1,2,3,4,5}', CURRENT\_DATE);

-- Inserting data into COURSE\_TOPIC for SQL

INSERT INTO COURSE\_TOPIC (course\_id, topic\_ids, last\_update\_date) VALUES

(2, '{6,7,8,9}', CURRENT\_DATE);

INSERT INTO PROGRESS (user\_id, course\_id, incomplete\_topics, complete\_topics, start\_date, end\_date) VALUES

(1, 1, '{1, 2}', '{3, 4, 5}', '2024-02-01', '2024-02-15'),

(2, 1, '{4, 5}', '{1, 2, 3}', '2024-01-20', '2024-02-10'),

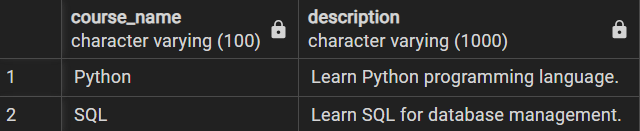
(1, 2, '{6, 9}', '{7, 8}', '2024-02-05', '2024-02-18');

**5.3 SQL QUERIES**

1. **Retrieve all the courses with their names and description**

SELECT course name, description

FROM COURSES;



1. **Retrieve all topics from a particular course**

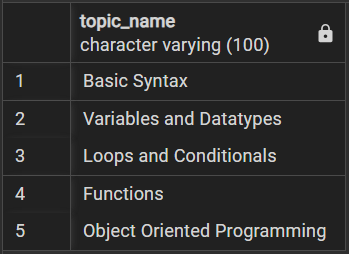
SELECT t.topic\_name

FROM COURSE\_TOPIC ct

JOIN COURSES c ON ct.course\_id = c.id

JOIN TOPICS t ON t.id = ANY(ct.topic\_ids)

WHERE c.course\_name = ‘Python’;



1. **Retrieve details of the progress made by a particular user in each enrolled course, including the number of completed topics, total topics in each course, and the percentage of completion.**

SELECT

u.name AS username,

c.course\_name,

p.complete\_topics AS completed\_topics,

ARRAY\_LENGTH(p.complete\_topics, 1) AS completed\_topics\_count,

ARRAY\_LENGTH(ct.topic\_ids, 1) AS total\_topics,

ROUND((ARRAY\_LENGTH(p.complete\_topics, 1) \* 100.0) / ARRAY\_LENGTH(ct.topic\_ids, 1), 2) AS completion\_percentage

FROM

USERS u

JOIN

PROGRESS p ON u.id = p.user\_id

JOIN

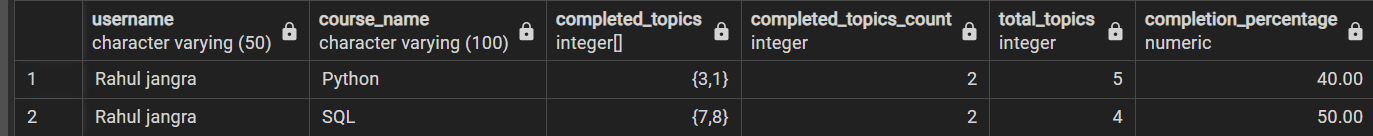
COURSES c ON p.course\_id = c.id

JOIN

COURSE\_TOPIC ct ON c.id = ct.course\_id

WHERE

u.name = {user\_name



1. **Update the user progress upon completion of a particular topic from a course they are enrolled in.**

CREATE OR REPLACE FUNCTION update\_progress\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'UPDATE' THEN

UPDATE progress

SET

complete\_topics = array\_append(progress.complete\_topics, NEW.completed\_topic\_id),

incomplete\_topics = array\_remove(progress.incomplete\_topics, NEW.completed\_topic\_id)

WHERE

user\_id = NEW.user\_id

AND course\_id = NEW.course\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_progress

AFTER UPDATE ON progress

FOR EACH ROW

EXECUTE FUNCTION update\_progress\_trigger();

Query returned successfully in 119 msec.

1. **Register a new user**

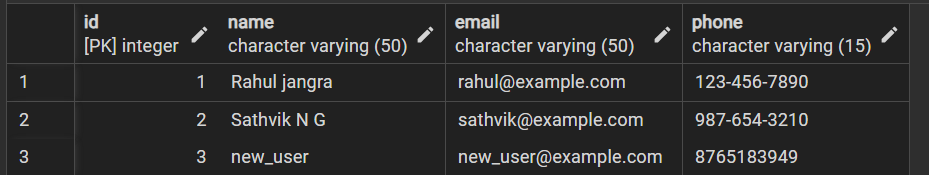
INSERT INTO USERS

VALUES (1,'new\_user', 'new\_user@example.com',8765183949);

Output:

INSERT 0 1

Query returned successfully in 104 msec.



1. **Retrieve the most popular courses based on the number of enrollments:**

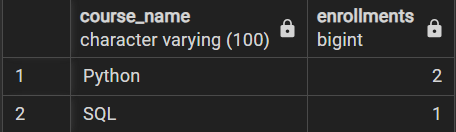
SELECT c.course\_name, COUNT(p.user\_id) AS enrollments

FROM courses c

LEFT JOIN progress p ON c.id = p.course\_id

GROUP BY c.course\_name

ORDER BY enrollments DESC;



1. **Procedure:**

DELIMITER //

CREATE PROCEDURE UpdateUserData(

IN user\_id INT,

IN new\_username VARCHAR(255),

IN new\_email VARCHAR(255)

)

BEGIN

-- Check if user exists

DECLARE user\_exists INT;

SELECT COUNT(\*) INTO user\_exists FROM users WHERE user\_id = user\_id;

-- If user exists, update user data

IF user\_exists > 0 THEN

UPDATE users

SET

username = new\_username,

email = new\_email

WHERE user\_id = user\_id;

ELSE

-- Handle the case where the user does not exist (optional)

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'User does not exist';

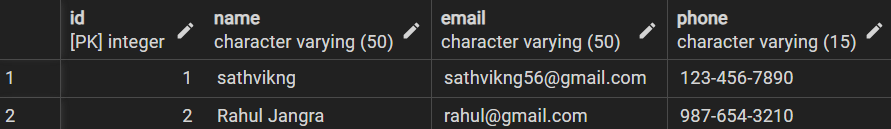
END IF;

END //

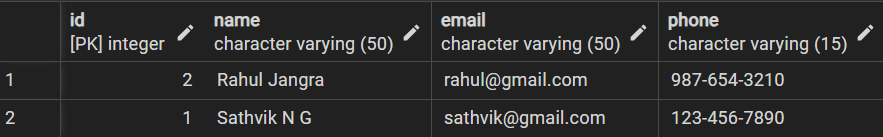
DELIMITER;

OUTPUT:

The `UpdateUserData` stored procedure takes input parameters—`user\_id`, `new\_username`, and `new\_email`. It checks if the user exists in the `users` table, updating the user's `username` and `email` if found. Optionally, it signals an error if the user doesn't exist.



Before procedure

After procedure

**CHAPTER -6**

**FRONTEND DESIGN, CONNECTIVITY AND IMPLEMENTATION**

**6.1 FRONTEND DESIGN AND USER INTERFACE(UI) DESCRIPTION:**

**6.1.1 Courses Module:**

This module contains templates for displaying individual courses. Each course template displays the topics within the course, along with checkboxes next to each topic. These checkboxes allow users to mark topics as completed, thereby increasing the course completion percentage. This interactive feature provides users with a visual representation of their progress within each course.

Functionality:

* Display Topics: The templates dynamically display the topics within each course, allowing users to navigate through the course content.
* Topic Completion: Checkboxes next to each topic enable users to mark topics as completed, updating the course completion percentage accordingly.
* Save Progress: Users can save their progress within a course, ensuring that their completion status is retained across sessions.
* View References: Links to additional references related to the course provide users with supplementary resources to enhance their learning experience.

**6.1.2 Course List Module:**

This Module contains templates for listing all available courses. Each course is accompanied by a brief description and links to view the topics within the course. This listing allows users to explore and discover courses of interest.

Functionality:

* Course Listing: Templates dynamically list all available courses, providing users with an overview of the course offerings.
* Description: A brief description accompanying each course gives users insight into the content and objectives of the course.
* View Topics: Links to view topics within each course allow users to delve deeper into the course curriculum and make informed decisions about enrollment.

**6.1.3 Profile Module:**

This Module contains templates for displaying user profiles. Each profile template showcases the user's username, profile photo, and a list of courses in which the user is enrolled. Additionally, the template displays the completion percentage for each course, allowing users to track their progress.

Functionality:

* User Information: The template prominently displays the user's username and profile photo, providing personalization and visual identification.
* Enrolled Courses: A list of enrolled courses allows users to see which courses they are currently participating in.
* Completion Percentage: The completion percentage for each course informs users about their progress and encourages them to continue their learning journey

**6.2 BACKEND IMPLEMENTION AND CONNECTIVITY WITH FRONTEND**

**6.2.1 Models Explanation**

1. User Model

Fields:

- Their\_ID: Primary key for the user.

- Their\_Username: Username of the user.

- Their\_Name: Full name of the user.

- Their\_Email: Email address of the user.

- Their\_Password: Password of the user.

- Their\_Join\_Date: Date and time when the user joined.

- Their\_last\_login: Date and time of the user's last login.

- Their\_User\_image: ImageField for storing user images.

Methods:

- \_\_str\_\_(): Returns the string representation of the user, which includes their ID, username, and name.

- \_\_list\_\_(): Returns a list representation of the user containing their ID, username, name, email, join date, and user image.

- check\_password(): Checks if the provided password matches the user's stored password.

2. Courses Model

Fields:

- Course\_ID: Primary key for the course.

- Course\_name: Name of the course.

- Date\_added: Date and time when the course was added.

- No\_of\_enrolment: Number of users enrolled in the course.

- Course\_description: Description of the course.

- Image\_link: Link to the course image.

Methods:

- \_\_str\_\_(): Returns the string representation of the course, which includes the course ID and name.

- \_\_list\_\_(): Returns a list representation of the course.

3. Topics Model

Fields:

- Topic\_ID: Primary key for the topic.

- Topic\_name: Name of the topic.

- Topic\_description: Description of the topic.

- Last\_updated: Date and time when the topic was last updated.

Methods:

- \_\_str\_\_(): Returns the string representation of the topic, which includes the topic ID and name.

- \_\_list\_\_(): Returns a list representation of the topic.

4. CTtable Model

- Fields:

- Course\_ID: ForeignKey to the Courses model, indicating the associated course.

- Topics\_IDs: Comma-separated string field storing IDs of related topics.

- Topic\_Sources: Field to store sources related to the topics.

- Methods:

- \_\_str\_\_(): Returns the string representation of the CTtable, which is the associated course ID.

- \_\_list\_\_(): Returns a list representation of the CTtable.

5. Progress Model

-Fields:

- User\_ID: ForeignKey to the Profile model, indicating the associated user.

- Course\_ID: ForeignKey to the Courses model, indicating the associated course.

- Completed\_topic\_IDs: Comma-separated string field storing IDs of completed topics.

- Incompleted\_topic\_IDs: Comma-separated string field storing IDs of incompleted topics.

- Start\_date: Date and time when the progress started.

- Last\_update\_date: Date and time when the progress was last updated.

- Methods:

- \_\_str\_\_(): Returns the string representation of the progress, which includes the user ID and course ID.

- \_\_list\_\_(): Returns a list representation of the progress.

**6.2.2 Views Explanation**

In Django, the views.py file contains functions or classes that handle HTTP requests and generate responses. Views encapsulate the business logic, process requests, interact with models, and render responses, often using templates. They also handle authentication, authorization, error handling, and middleware integration. Overall, views.py defines how the application interacts with clients and processes requests. Below are views used in our project

1. Index View:

- Description: This view handles the index page of the application.

- Features:

- Retrieves user data if the user is authenticated.

- Allows users to create a profile.

- Checks if a profile is created for the user.

- Usage: Renders the index page with user data and profile information if available.

2. Register View:

- Description: This view handles user registration.

- Features:

- Validates user registration form data.

- Saves user registration data.

- Sends a welcome email to the registered user.

- Usage: Renders the registration page and processes user registration requests.

3. Login View:

- Description: This view handles user authentication and login.

- Features:

- Authenticates user credentials.

- Redirects users to the index page upon successful login.

- Usage: Renders the login page and processes user login requests.

4. About View:

- Description: This view renders the about us page of the application.

- Usage: Renders the about us page with relevant information about the application.

5. Course List View:

- Description: This view lists all available courses.

- Features:

- Retrieves course data from the database.

- Limits course descriptions to 100 characters.

- Usage: Renders the course list page with course data.

6. Profile View:

- Description: This view displays user profiles.

- Features:

- Retrieves user profile data if the user is authenticated.

- Retrieves progress data for the user's enrolled courses.

- Usage: Renders the profile page with user profile and progress information.

7. Course View:

- Description: This view displays details of a specific course.

- Features:

- Retrieves course data and associated topics from the database.

- Allows enrolled users to save progress and enroll in courses.

- Usage: Renders the course page with course details and topic information.

These views collectively handle user authentication, registration, profile management, course listing, and course details display in the application. They ensure smooth navigation and interaction for users accessing different sections of the platform.

**6.2.3 Connectivity with Frontend:**

In a Django project, the frontend is connected with the backend through a combination of HTML templates, views, and URL configurations.

HTML Templates: HTML templates are used to define the structure and content of the frontend pages. These templates can include placeholders for dynamic data, which are populated by the backend before being rendered in the browser. Django's templating engine allows for the insertion of dynamic content using template variables and template tags.

Views: Views in Django are Python functions or class-based views that handle incoming HTTP requests and return HTTP responses. Views fetch data from the backend models or perform business logic, and then pass this data to the frontend templates for rendering. Views can also handle form submissions, process user input, and perform other backend tasks.

URL Configurations: URL configurations define the mapping between URLs and view functions. In Django, the urlpatterns list in the urls.py file maps URL patterns to corresponding view functions. When a user accesses a URL in the frontend, Django's URL resolver matches the URL pattern to a view function, which then processes the request and returns an HTTP response.

Data Passing: Data is passed between the backend and frontend through the context dictionary in views. In Django views, data retrieved from the backend models or processed by the backend logic is added to the context dictionary and passed to the frontend templates. The frontend templates use this data to dynamically generate HTML content that is rendered in the browser.

Form Handling: In web applications, forms are used to collect user input and submit data to the backend. Django provides form handling functionality through its forms.py module, allowing developers to define HTML forms in templates and handle form submissions in views. Form data submitted by users is processed in the backend, validated, and then stored or used for further processing.

**CHAPTER – 7**

**SOURCE CODE**

**models.py**

from django.db import models

from django.contrib.auth.models import AbstractUser, BaseUserManager, PermissionsMixin

from django.contrib.postgres.fields import ArrayField

from django.db import models

import datetime

from django import forms

def def\_img():

return "https://static.wikia.nocookie.net/solo-leveling/images/8/8b/Jinwoo4.jpg/revision/latest?cb=20210803222649"

**#USER Model**

class TrackerUser(models.Model):

Their\_ID = models.CharField(primary\_key = True,max\_length = 10)

Their\_Username = models.CharField(max\_length=100,default="(^o^)")

Their\_Name = models.CharField(max\_length = 100,null=True)

Their\_Email = models.EmailField(null=True)

Their\_Password = models.CharField(max\_length=20, default = "0000")

Their\_Join\_Date = models.DateTimeField(auto\_now = True)

Their\_last\_login = models.DateTimeField(auto\_now=True)

# Their\_Login\_map = models.CharField(max\_length=10000,null=True)

Their\_User\_image = models.ImageField(upload\_to='user\_images/', default='static/images/default.png') # ImageField for storing user images

def \_\_str\_\_(self) -> str:

return f"{self.Their\_ID} > {self.Their\_Username} = {self.Their\_Name}"

def \_\_list\_\_(self) -> list:

return [self.Their\_ID,self.Their\_Username,self.Their\_Name,self.Their\_Email,self.Their\_Join\_Date,self.Their\_User\_image]

def check\_password(self,pword) -> bool:

return self.Their\_Password == pword

**# COURSES Model**

class Courses(models.Model):

Course\_ID = models.CharField(primary\_key = True, max\_length=10)

Course\_name = models.CharField(max\_length=50, unique=True,default="(\*\_\*)")

Date\_added = models.DateTimeField(auto\_now=True)

No\_of\_enrolment = models.IntegerField(default=0)

Course\_description = models.CharField(max\_length = 1200,null=True)

Image\_link = models.CharField(max\_length=150,default=def\_img)

def \_\_str\_\_(self) -> str:

return f"{self.Course\_ID} : {self.Course\_name}"

def \_\_list\_\_(self) -> list:

return [self.Course\_ID,self.Course\_name,self.Date\_added,self.No\_of\_enrolment,self.Course\_description,self.Image\_link]

**#TOPICS Model**

class Topics(models.Model):

Topic\_ID = models.CharField(primary\_key = True, max\_length=10)

Topic\_name = models.CharField(max\_length=50,default="(\*-\*)")

Topic\_description = models.CharField(max\_length=1200,null=True)

Last\_updated = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self) -> str:

return f"{self.Topic\_ID} : {self.Topic\_name}"

def \_\_list\_\_(self) -> list:

return [self.Topic\_ID,self.Topic\_name,self.Topic\_description,self.Last\_updated]

**#COURSE\_TOPIC Model**

class CTtable(models.Model):

Course\_ID = models.ForeignKey('Courses', related\_name = "ct\_cid", on\_delete = models.CASCADE)

Topics\_ID = models.ForeignKey('Topics', related\_name = "ct\_tid", on\_delete = models.CASCADE, default = "1")

# Course\_rating = models.IntegerField(default=0)

# Recommended\_Time = models.IntegerField(default=0)

Topic\_Sources = models.CharField(max\_length=1000,null=True)

def \_\_str\_\_(self) -> str:

return f"{self.Course\_ID}"

def \_\_list\_\_(self) -> list:

return [self.Course\_ID,self.Topics\_ID,self.Topic\_Sources]

**#PROGRESS Model**

class Progress(models.Model):

User\_ID = models.ForeignKey("TrackerUser", related\_name = "prog\_uid", on\_delete = models.CASCADE)

Course\_ID = models.ForeignKey('Courses', related\_name = "prog\_cid", on\_delete = models.CASCADE)

Completed\_topics = ArrayField(models.CharField(max\_length=100), default=list, blank=True)

Incomplete\_topics = ArrayField(models.CharField(max\_length=100), default=list, blank=True)

Start\_date = models.DateTimeField(auto\_now=True)

Finish\_date = models.DateTimeField(auto\_now=True) # if Finish date is same as Start\_date // display as unfinished

def \_\_str\_\_(self) -> str:

return f"{self.User\_ID} : {self.Course\_ID}"

def \_\_list\_\_(self) -> list:

return [self.User\_ID,self.Course\_ID,self.Completed\_topics,self.Incomplete\_topics,self.Start\_date,self.Finish\_date]

def save(self, \*args, \*\*kwargs):

# Increase the number of enrollments for the course

self.Course\_ID.No\_of\_enrolment += 1

self.Course\_ID.save() # Save the course with updated enrollment count

super(Progress, self).save(\*args, \*\*kwargs)

**views.py**

from django.shortcuts import render, redirect

from django.contrib import messages

from django.contrib.auth import authenticate, login

from django.contrib.auth.decorators import login\_required

from django.contrib.auth.forms import AuthenticationForm

from .forms import UserRegisterForm

from django.core.mail import send\_mail

from django.core.mail import EmailMultiAlternatives

from django.template.loader import get\_template

from django.template import Context

from django.contrib.auth.models import User

from .models import \*

**# COURSE\_LIST Module:**

def course\_list(request):

courseData = [x.\_\_dict\_\_ for x in Courses.objects.all()]

for el in courseData:

el['Course\_description'] = el['Course\_description'][:100]

return render(request, 'course\_list/courseListPage.html' ,{

"courseData":courseData

**# PROFILE Module**

def pprofile(request,usernname):

profiledata = ""

if request.user.is\_authenticated:

# print("Working")

# try:

userdata = User.objects.get(username=usernname)

profiledata = profile.objects.get(uid = userdata).\_\_dict\_\_

#---------------------------------------------------

#---------------------------------------------------

# If the user is loggeed in bring all his progress data

progressData = [x.\_\_dict\_\_ for x in Prograss.objects.filter(User\_ID = userdata)]

if userdata:

userdata = userdata.\_\_dict\_\_

for prog in progressData:

prog['coursename'] = Courses.objects.get(Course\_ID = prog['Course\_ID\_id']).\_\_dict\_\_['Course\_name']

prog['percentage'] = "%.2f" % (100\*len([x for x in prog['Completed\_topic\_IDs'].split(',') if '0' <= x <= '9999'])/len([x for x in prog['Incompleted\_topic\_IDs'].split(',') if '0' <= x <= '999' ]))

print([x for x in prog['Completed\_topic\_IDs'].split(',') if '0' <= x <= '9999'],[x for x in prog['Incompleted\_topic\_IDs'].split(',') if '0' <= x <= '999' ])

return render(request, 'profile/profile.html' ,{

"userdata" : userdata,

"profiledata":profiledata,

"progressData":progressData

})

# except :

messages.info(request, "User does not exist")

return redirect('index')

else:

return redirect('login')

**# COURSE Module**

def course(request, courseid):

course = Courses.objects.get(Course\_ID = courseid).\_\_dict\_\_

print([tid for tid in CTtable.objects.get(Course\_ID=course['Course\_ID']).\_\_dict\_\_['Topics\_IDs'].split(',') if '-10' <= tid <= '9999' ])

topicdata = [Topics.objects.get(Topic\_ID=tid).\_\_dict\_\_ for tid in str(CTtable.objects.get(Course\_ID=course['Course\_ID']).\_\_dict\_\_['Topics\_IDs']).split(',') if '-10' <= tid <= '99999' ]

enrollementStatus = False

progressData = ""

if request.user.is\_authenticated:

# ---------------------------------------------------

# Only if the user is logged in and enrolled in the course

# Following id to save the progress

if request.method == "POST":

print(request.POST)

if request.POST.get("enroll-button", 0) == 'Enroll':

incmplt\_tpc = "".join([str(x["Topic\_ID"]) + "," for x in topicdata])

print(incmplt\_tpc)

p = Prograss(User\_ID = request.user ,Course\_ID = Courses.objects.get(Course\_ID = courseid) ,Completed\_topic\_IDs = "0",Incompleted\_topic\_IDs=incmplt\_tpc)

print("saved")

p.save()

elif request.POST.get("save-progress-button", 0) == "SaveProgress" :

print(request.POST.get("enroll-button", 0))

completedTopics = ""

for i in range(50):

if 'on' in request.POST.get(f"topic\_{i}\_done", ''):

completedTopics += str(i)+","

# MyModel.objects.filter(pk=some\_value).update(field1='some value')

for prog in [x for x in Prograss.objects.filter(User\_ID = request.user.id)]:

if prog.\_\_dict\_\_['Course\_ID\_id'] == courseid:

prog.\_\_dict\_\_["Completed\_topic\_IDs"] = completedTopic

print(prog)

prog.save()

progressDatas = [x.\_\_dict\_\_ for x in Prograss.objects.filter(User\_ID = request.user.id)]

for val in progressDatas:

if str(val['Course\_ID\_id']) == course['Course\_ID']:

progressData = val

# print(topicdata)

enrollementStatus = True

for tdata in topicdata:

tdata["Completed"] = True if tdata['Topic\_ID'] in "".join(val['Completed\_topic\_IDs']).split(',') else False

return render(request, 'courses/general.html', {

"courseData":course,

"topicdatas\_basic":topicdata,

"topicdatas\_inter":"",

"topicdatas\_adv":"",

"isEnrolled":enrollementStatus,

"progressData":progressData

})

**CHAPTER – 8**

**RESULT, DISCUSSION & SCREENSHOTS**

* **Course List Page:**

This page displays a list of courses with brief descriptions. It retrieves course data from the database, truncates descriptions for brevity, and renders the information on the 'courseListPage.html' template.

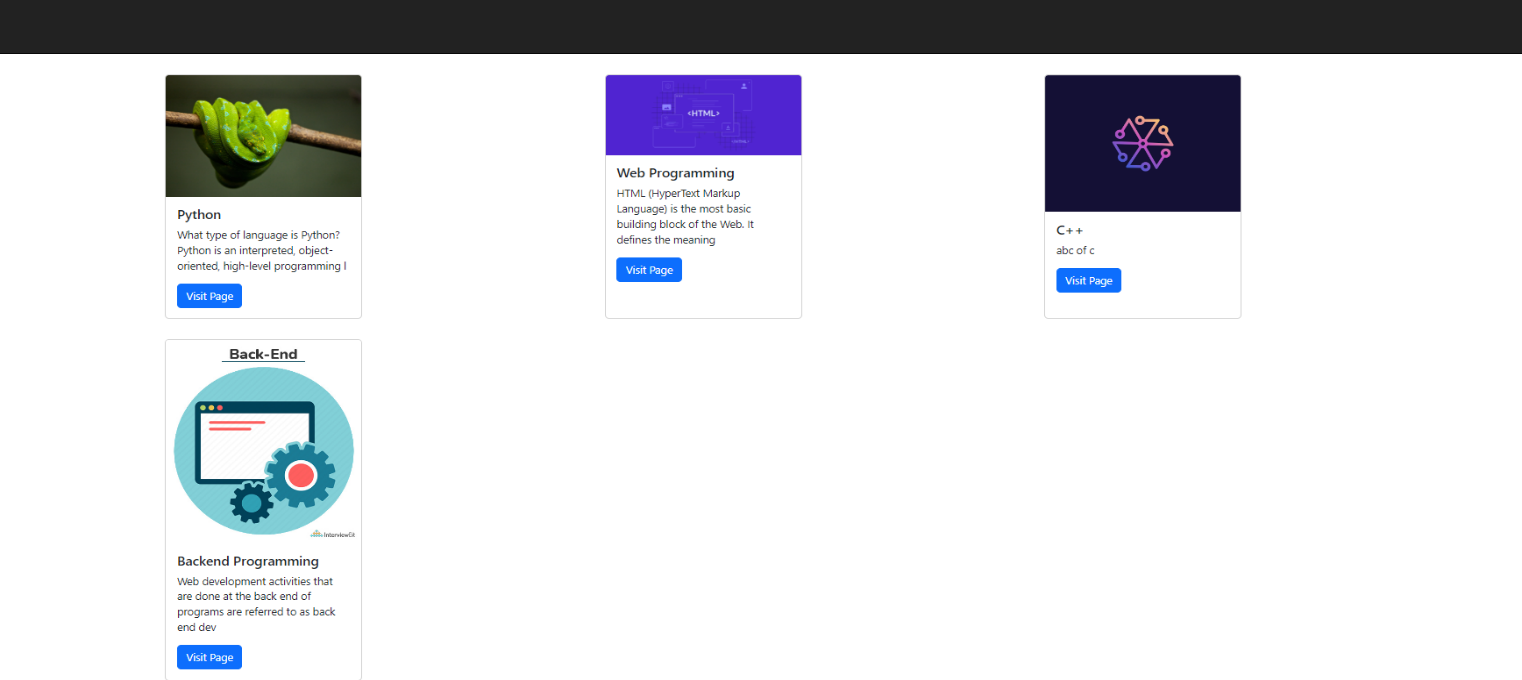


Fig 8.1 Course List Page

* **Course Page:**

Accessible only to logged-in users, this page presents details about a specific course, including topics and enrollment status. Users can enroll, save progress, and view their progress data. The template 'general.html' dynamically displays relevant course information.

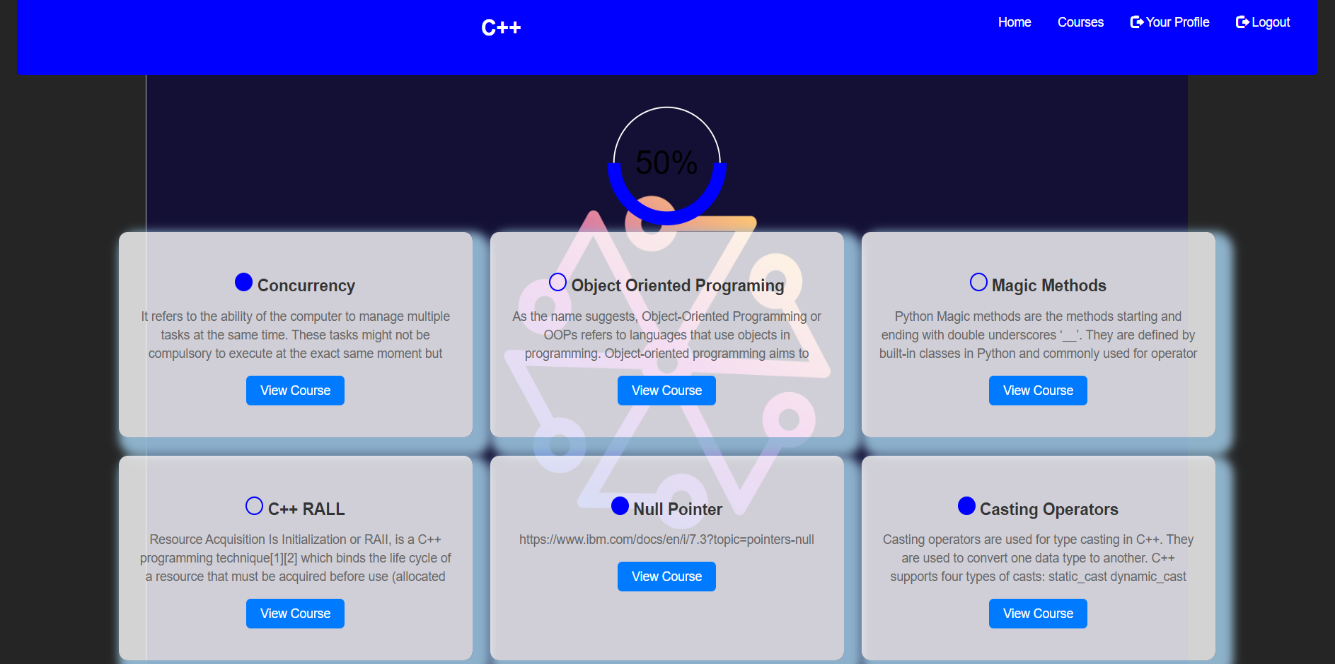


Fig 8.2 Course Page

* **Profile Page**:

Users can view their profiles, including personal and progress data. If authenticated, the page retrieves user and progress information from the database. The 'profile.html' template displays user data and progress details.

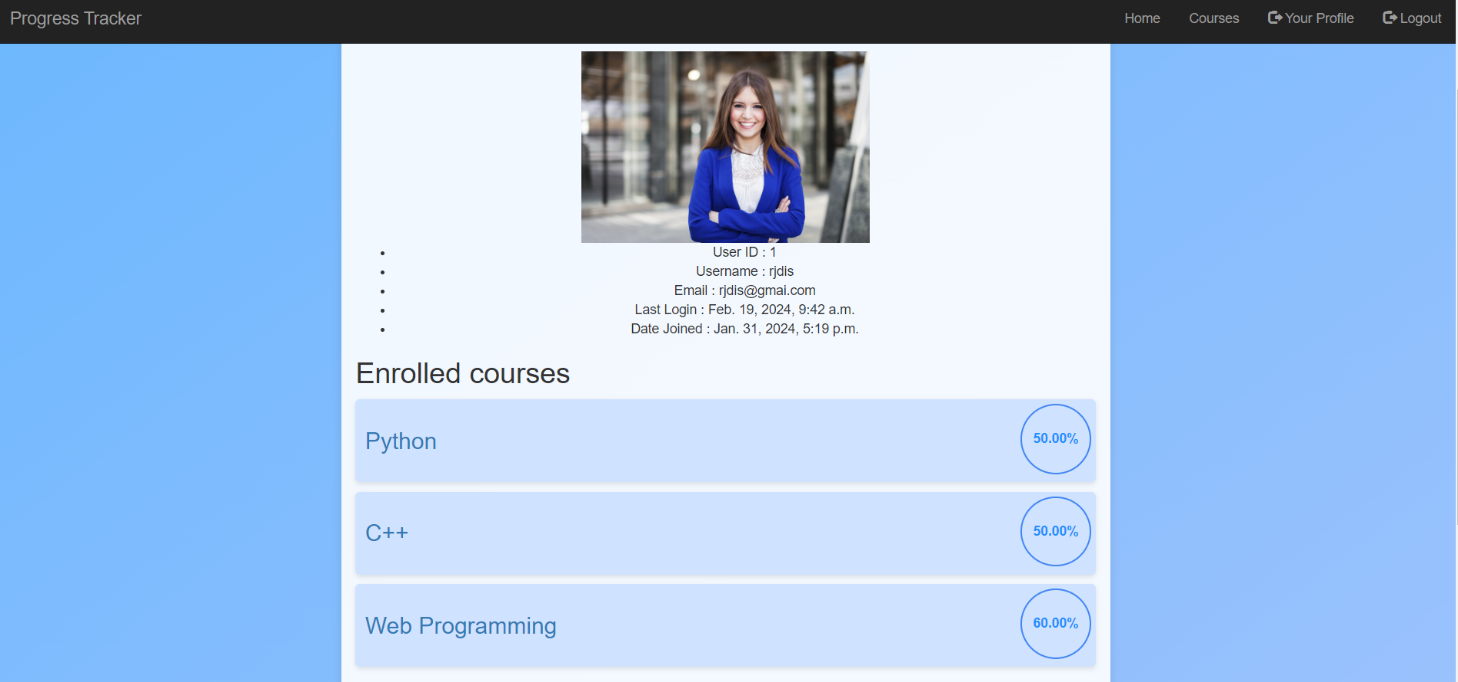


Fig 8.3 Profile Page

* Login and Registration Page:

This page facilitates user registration and login. Users can register with the 'UserRegisterForm,' and existing users can log in using the 'AuthenticationForm.' After registration, a welcome email is sent. The 'register.html' and 'login.html' templates handle the rendering of these forms.

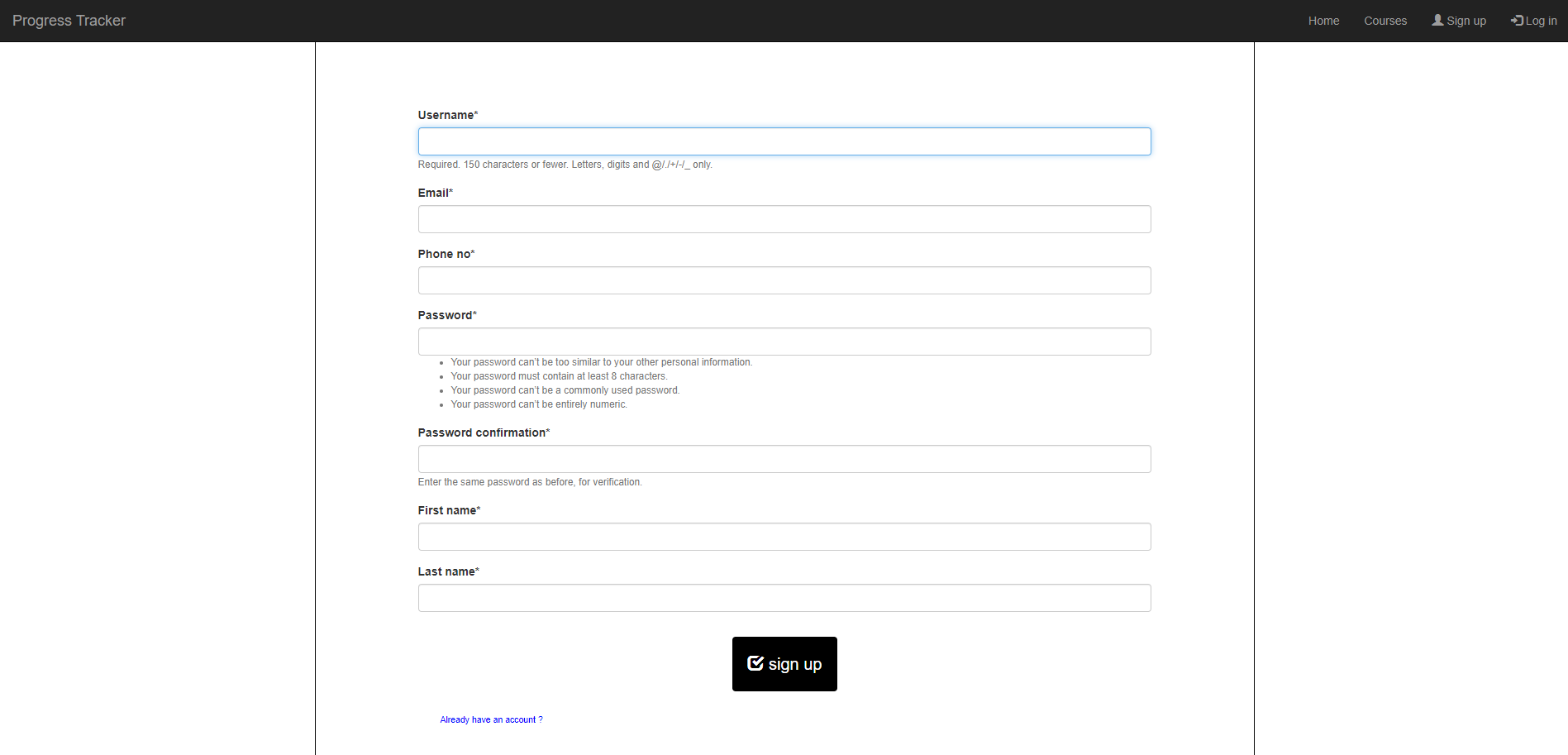
****

Fig 8.4 Login and Registration Page

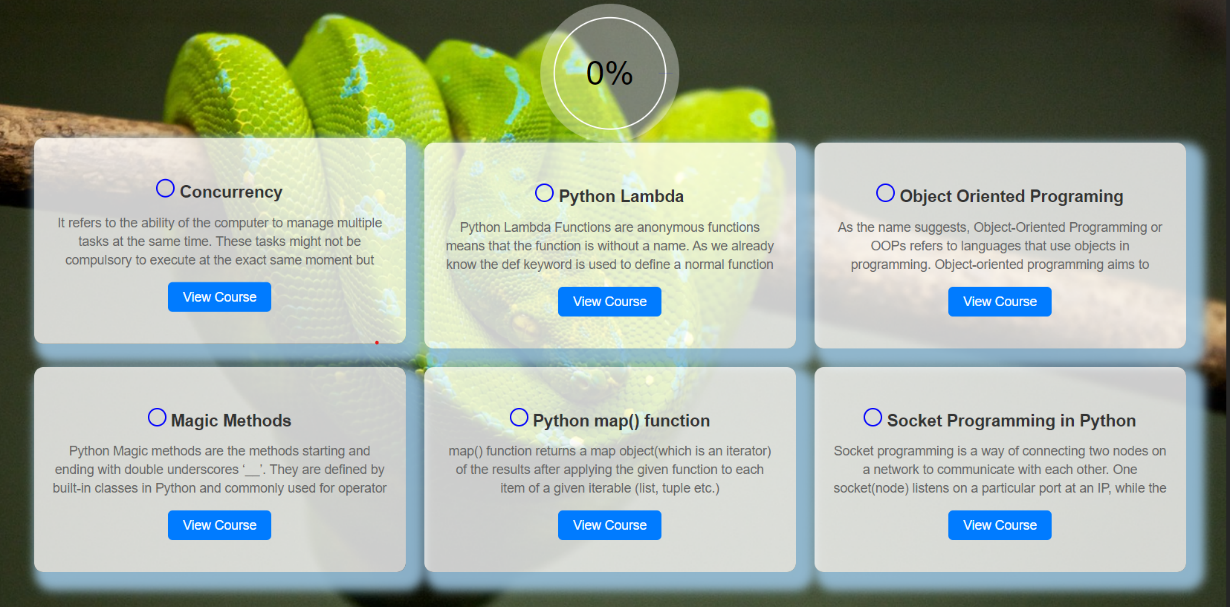


Fig 8.5 Before Trigger

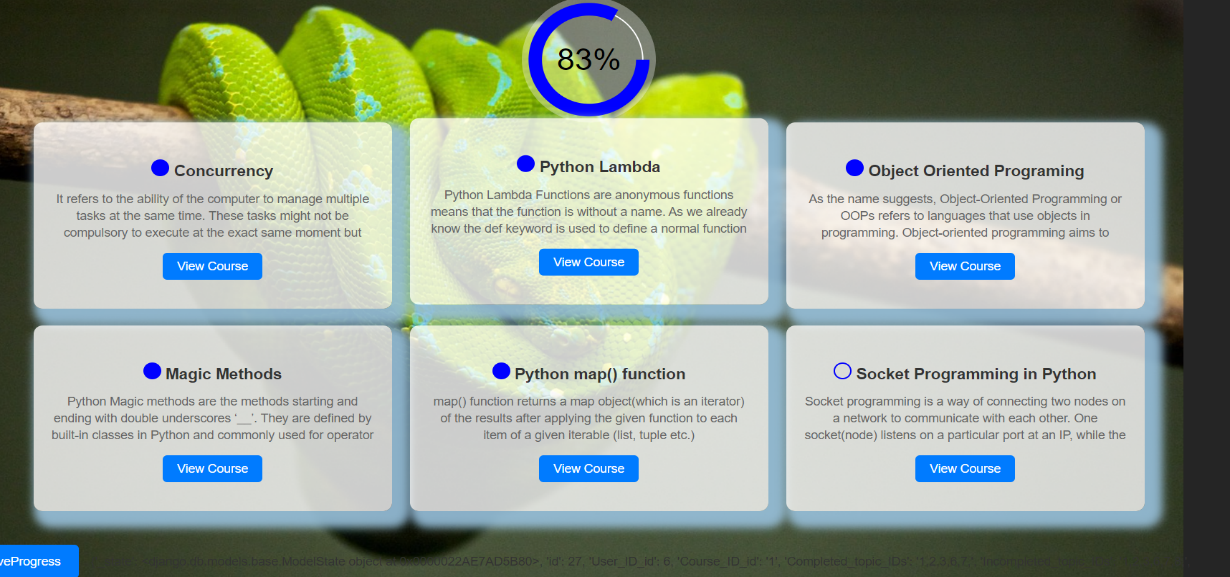


Fig 8.6 After Trigger

* User progress summary report:

The printed page gives an easy-to-understand report showing the percentage of completion for specific courses. It breaks down exactly how much progress users have made in each enrolled course, giving a clear picture of their achievements.

****

Fig 8.7 Print progress summary report

**CHAPTER – 9**

**CONCLUSION AND SCOPE**

The Learning Progress Tracker (LPT) project aims to revolutionize the educational experience by providing learners with a user-friendly and personalized tool to track and enhance their progress across various courses and topics. By focusing on personalized learning paths, efficient progress tracking, user-friendly interface design, and adaptability, the LPT offers a comprehensive solution to the evolving needs of modern learners.

**Scope of the Project:**

The scope of the Learning Progress Tracker project encompasses the development of a robust platform that enables learners to:

* Create personalized learning paths tailored to their individual preferences, strengths, and weaknesses.
* Track their progress across different courses and topics in real-time, allowing for easy monitoring of achievements and identification of areas for improvement.
* Access a user-friendly interface designed for easy navigation, ensuring accessibility for users with varying levels of technological proficiency.
* Ensure adaptability and scalability to accommodate different educational settings and future enhancements or expansions.

**Potential Future Improvements :**

* Advanced Analytics: Implement deeper insights and trend analysis for learning patterns.
* Integration with LMS: Seamlessly integrate with existing Learning Management Systems.
* Mobile App Development: Create a mobile app for on-the-go progress tracking.
* Gamification: Add badges, rewards, and challenges to enhance engagement.
* Social Learning: Enable forums and collaboration tools for peer interaction.
* External Resource Integration: Incorporate access to online courses and tutorials.
* Personalized Recommendations: Offer tailored course suggestions based on user data.
* Accessibility: Improve features for users with disabilities.
* Feedback Mechanisms: Gather user input to guide platform enhancements.
* Cross-Platform Compatibility: Ensure a consistent experience across devices.

Overall, the Learning Progress Tracker project endeavors to empower learners in their educational journey, providing them with the tools and resources they need to succeed in an ever-changing learning landscape.

**References and Weblinks**

* Official Django Documentation (https://docs.djangoproject.com/)
* Django for Beginners (https://djangoforbeginners.com/)
* Official Python Documentation (https://docs.python.org/3/)
* GeeksforGeeks - Python Programming Language (https://www.geeksforgeeks.org/python-programming-language/)
* MDN Web Docs (https://developer.mozilla.org/en-US/docs/Web)
* W3Schools (https://www.w3schools.com/)
* W3Schools - SQL Tutorial (https://www.w3schools.com/sql/)
* Tutorialspoint - Database Management System (https://www.tutorialspoint.com/dbms/index.htm)
* GeeksforGeeks - Database Management System (DBMS) (https://www.geeksforgeeks.org/dbms-database-management-systems/)
* Mode Analytics - SQL Tutorial (https://mode.com/sql-tutorial/)
* SQLZoo (https://sqlzoo.net/)
* Official PostgreSQL Documentation (https://www.postgresql.org/docs/)
* PostgreSQL Tutorial (https://www.postgresqltutorial.com/)
* DigitalOcean - How To Use PostgreSQL with Django (https://www.digitalocean.com/community/tutorial\_series/how-to-use-postgresql-with-django)