**Learning Platform Web Application**

**By**

**Sadia Shabbir (2020-GCUF-000000)**

**Maryam Tariq (2020-GCUF-000000)**

**Samia Shabbir (2020-GCUF-000000)**

Project submitted in partial fulfillment of the requirements for the degree of

**BACHELOR OF SCIENCE**

**IN**

**COMPUTER SCIENCE**



**DEPARTMENT OF COMPUTER SCIENCE**

**Government College University Faisalabad**

**2024**

## DECLARATION

This project, a die-hard work, is productized out by Sadia Shabbir, Maryam Tariq and Samia Shabbir under the supervision of Mr. hamza (Lecturer, Computer Department) and Mr. \_\_\_\_\_\_\_\_ (Hod, Computer Department) University Community College, GC University, Faisalabad, Pakistan. We feel please to declare that the project and contents of this project is the productive result of our hardworking, studies and research and no part of this is copied from any published source. This work has been conducted under the practical atmosphere of our studies not for the award of any other degree / diploma. The University may take action if the information provided is found guilty at any stage. Any external sources of information used in this project, including references, have been duly acknowledged through proper citations and bibliographical references. The project has not been previously submitted for any other degree or examination at any other institution. Any contributions made by others to this project, including guidance and support from faculty members, have been duly acknowledged. The software code, documentation, and any other materials presented as part of this project are the result of my own work, unless otherwise acknowledged. I take full responsibility for the authenticity and originality of the content presented in this project. I understand that any misrepresentation or falsification of information in this declaration will have serious consequences, including the possibility of disciplinary action by the institution.

**Signature of Student:**

Student Name: **Sadia Shabbir**

Registration No: 2020-GCUF-000000

Student Name: **Maryam Tariq**

Registration No: 2020-GCUF-000000

Student Name: **Samia Shabbir**

Registration No: 2020-GCUF-000000

**Project complete Report**

**Project title**: **Learning Platform Web Application**

**Student name:** Sadia Shabbir

**Student name:** Maryam Tariq

**Student name:** Samia Shabbir

**Session:** 2020-2024

**Program:** Bachelor of Science in Computer Science

*It is to certify that the title & proposal submitted by the students are clear and according to project scope.*

*It is further certifying that the project of the students may kindly be process for external examiner/ viva voice.*

**Project Coordinator**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Department of Computer Science Government College University Faisalabad

**CERTIFICATE BY THE PROJECT SUPERVISOR**

This is to certify that the final year proj­ect of BS (CS) “**Learning Platform**” was developed by, **“Sadia Shabbir” “Maryam Tariq” and "Samia Shabbir”** supervised by “Supervisor Name and that in his opinion it is fully adequate, for the degree of BSIT in Computer Science.

**Supervisor:**

Name:

Signature:

**Co-Supervisor:**

Name: \_\_\_\_

Signature:\_\_\_\_\_

**Head of department:**

Department of Computer Science Government College University Faisalabad

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude and appreciation to all those who have contributed to the successful completion of my Final Year Project titled "Learning Platform Web Application " This project has been an enriching and rewarding experience, and I would like to acknowledge the individuals and organizations for their support. My project supervisor, Mr. Hamza, for their invaluable guidance, expertise, and continuous support throughout the project. Their knowledge and insights have been instrumental in shaping the direction of this work. The faculty members of Rise College, for providing me with a conducive learning environment and valuable resources. I am grateful for their dedication to education and their efforts in imparting knowledge and skills. My family and friends, for their unwavering support, encouragement, and understanding. Their belief in me and their motivation have been a constant source of inspiration, driving me to overcome challenges and strive for excellence. The participants who willingly volunteered their time and provided feedback during the testing and evaluation phases of the project. Their insights and suggestions have been invaluable in refining the functionality and user experience of the app. The developers and contributors of open-source libraries, frameworks, and tools that were utilized in the development of this project. Their efforts have significantly expedited the development process and enhanced the overall quality of the application. I would also like to extend my gratitude to all the train passengers, restaurant owners, and delivery personnel who may benefit from this train food ordering app. Their needs and expectations have been a driving force behind the development of this solution. Lastly, I want to express my appreciation to all the individuals, both mentioned and unnamed, who have directly or indirectly contributed to this project. Your support and assistance have played an integral role in its successful completion. In conclusion, I am truly grateful to everyone who has been a part of this project. Your contributions, encouragement, and support have been invaluable, and I am honored to have had the opportunity to work on this project with such wonderful individuals.

**Sadia Shabbir (Roll No. 2260)**

**Maryam Tariq (Roll No. 2206)**

**Samia Shabbir (Roll No. 2259)**

# 

**DEDICATION**

We would like to dedicate this work to our families who support us

financially and morally during the whole project, without their support,

this work was not possible.

Table of Contents

[DECLARATION 2](#_Toc179363020)

[ACKNOWLEDGEMENT 5](#_Toc179363021)

[Chapter 1 10](#_Toc179363022)

[Introduction to the Problem 10](#_Toc179363023)

[1.1 Introduction 10](#_Toc179363024)

[1.2 Background 11](#_Toc179363025)

[1.3 Purpose 11](#_Toc179363026)

[1.4 Scope 11](#_Toc179363027)

[1.5 Objective 12](#_Toc179363028)

[1.6 Intended Audience and Reading Suggestions 12](#_Toc179363029)

[1.7 Document Conventions 12](#_Toc179363030)

[Chapter 2 14](#_Toc179363031)

[Software Requirement Specification 14](#_Toc179363032)

[2.1 Overall Description 15](#_Toc179363033)

[2.1.1 Product Perspective 15](#_Toc179363034)

[2.1.2 Product Features 15](#_Toc179363035)

[2.1.3 Design and Implementation Constraints 16](#_Toc179363036)

[2.1.4 Assumptions and Dependencies 17](#_Toc179363037)

[2.2 System Features 17](#_Toc179363038)

[User Registration and Login 17](#_Toc179363039)

[Courses Browsing and Enrollment 19](#_Toc179363040)

[Admin Panel for Learning Platform 19](#_Toc179363041)

[View Blogs 20](#_Toc179363042)

[2.3 External Interface Requirements 21](#_Toc179363044)

[2.3.1 User Interfaces 21](#_Toc179363045)

[2.3.2 Hardware Interfaces 21](#_Toc179363046)

[2.3.3 Software Interfaces 21](#_Toc179363047)

[Chapter 3 23](#_Toc179363048)

[Analysis (Use-Case Model) 23](#_Toc179363049)

[3.1 Identifying Actors and Use Cases in the Learning Platform 24](#_Toc179363050)

[3.2 Forming Use-Casefor the Learning Platform 25](#_Toc179363051)

[3.3 Event Flow for Use-Cases 26](#_Toc179363052)

[Chapter 4 27](#_Toc179363053)

[Design 27](#_Toc179363054)

[4.1 Architecture Diagram 28](#_Toc179363055)

[4.2 ERD with Data Dictionary 28](#_Toc179363056)

[4.3 Data Flow Diagram (Level 0 and Level 1) 29](#_Toc179363057)

[4.4 Class Diagram 31](#_Toc179363058)

[Chapter 5 33](#_Toc179363059)

[Implementation 33](#_Toc179363060)

[5.1 Component Diagram 34](#_Toc179363061)

[5.2 Deployment Diagram 34](#_Toc179363062)

[5.3 Database Architecture 35](#_Toc179363063)

[Chapter 6 37](#_Toc179363064)

[Testing (Software Quality Attributes) 37](#_Toc179363065)

[6.1 Test Case Specification 38](#_Toc179363066)

[6.2 Black Box Test Cases 38](#_Toc179363067)

[6.2.1 Boundary Value Analysis (BVA) 38](#_Toc179363068)

[6.2.2 Equivalence Class Partitioning 39](#_Toc179363069)

[6.2.3 State Transition Testing for Course Enrollment 39](#_Toc179363070)

[6.2.5 Graph-Based Testing 40](#_Toc179363071)

[6.3 White Box Testing 40](#_Toc179363072)

[6.3.1 Statement Coverage 40](#_Toc179363073)

[6.3.2 Branch Coverage 41](#_Toc179363074)

[6.3.3 Path Coverage 41](#_Toc179363075)

[Chapter 7 42](#_Toc179363076)

[Tools and Technologies 42](#_Toc179363077)

[7.1 Programming Languages 43](#_Toc179363078)

[7.2 Operating Environment 43](#_Toc179363079)

[7.3 Code: 44](#_Toc179363080)

**Appendix A: User Documentation**

**Appendix B: Source Code**

**List of Figures**

Figure2.1: System Architecture Diagram- Learning Platform

Figure 2.2: Flowchart of user of Learning Platform Admin Learning platform management

Figure2.3: User Registration and Login

Figure 2.4: Course Browsing and Enrollment

Figure 2.5: Admin panel

Figure 2.6: View Blogs

Figure 3.1: Use case diagram in Learning Platform

Figure 4.1: Architecture diagram

Figure 4.2: ERD

Figure 4.3: Flow diagram level 0

Figure 4.4: Level 1 DFD

Figure4.5: class diagram

Figure 5.1: component diagram

Figure 5.2: Deployment

Figure 5.3: Database Tables

# 

# Chapter 1

# Introduction to the Problem

## Introduction

Learning Platform Web Application is a comprehensive online learning platform designed for both children and adults, offering diverse courses and educational blogs. Users can easily explore, enroll in courses, and track their progress with features, and views. Learn Together ensures relevant, high-quality educational experiences for all users.

## Background

Existing educational platforms often lack personalized features and age-based content filtering, leading to a fragmented learning experience. Learning Platform Web Application addresses these issues with interactive course management, role-specific, and a progress tracking system to enhance the educational journey for all users

## Purpose

The purpose of the Learning Platform Web Application is to create an engaging and accessible online learning environment for both children and adults. This platform enables users to explore diverse educational content, participate in interactive courses, and track their progress through a user-friendly interface. Learning Platform Web Application aims to enhance the learning experience by providing role-specific features for lecturers, students, and admins, ensuring that all users receive high-quality, age-appropriate educational resources.

## 1.4 Scope

The scope of the Learning Platform Web Application encompasses the development of a comprehensive online learning system. It includes features for user registration, role-based access (teachers, students, admins), course management (creation, editing, deletion), blog management (creation, editing, deletion) and a ranking system based on user engagement metrics. The platform will support age-based content filtering to ensure appropriate educational material for users under 18 and provide a responsive, mobile-friendly interface using technologies such as React, MongoDB, Node.js, and Next.js. The project aims to enhance the learning experience through seamless user interaction and personalized educational content delivery.

**1.5 Objective**

The objective is to develop an online learning platform web application, enabling users to access educational content at any time and from any location.

* The convenience of learning from any place at any time.
* It simplifies the learning process by providing easy access to educational materials with just a few clicks.
* Web applications offer the flexibility to study from any device with internet access.
* Learners can access fresh and engaging content through the platform without the need for additional software.
* Provide users with a convenient and hassle-free way to engage in learning without the constraints of time or location, eliminating the need for physical study materials or in-person attendance.

## 1.6 Intended Audience and Reading Suggestions

This document is intended for:

* Developers: To understand the system architecture, design, and technologies used.
* Instructors/Content Creators: To familiarize themselves with the functionality of the platform and how to manage educational content.
* Learners: To explore the courses, services, and features available for learning.

Reading suggestions: For developers, it is recommended to start with Chapters 2 and 3, which cover the system architecture and design.

Instructors should focus on Chapters 1 and 4, which provide an overview of functionalities and content management features.

## 1.7 Document Conventions

This document follows the conventions listed below to ensure clarity and consistency in describing the development and features of the **Learning Platform Website Application**:

**1.7.1 Title:**

The document should have a clear and concise title, such as **Learning Platform Website Application** that reflects the purpose of the platform.

**1.7.2 Introduction:**

The document begins with an introduction that provides a brief overview of the learning platform, outlining key features such as **course creation**, **paid courses**, **blogging**, and **user management**, and highlights the benefits to both instructors and learners.

**1.7.3 Getting Started:**

This section provides step-by-step instructions for users on how to **sign up**, **login**, and **create a course** on the platform, along with initial setup steps like configuring the user profile and payment methods.

**1.7.4 User Interface:**

The document includes screenshots of key sections, such as the **dashboard**, **course management** and **blog creation** to help users navigate and understand the layout.

**1.7.5 Features:**

This section details the platform's features, such as **Login, registration,** **course learning** and **blogging** functionalities, explaining how users can interact with them.

**1.7.6 Troubleshooting:**

A troubleshooting section addresses common issues, such as **login problems**, **course access issues** along with suggested solutions or workarounds to resolve them.

**Chapter 2**

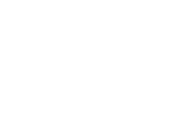
# Software Requirement Specification

## 2.1 Overall Description

## The Learning Management Platform (LMS) aims to streamline the process of managing and delivering educational content through a user-friendly web interface. It allows students to browse available courses, view detailed course specifications, and enroll directly from the platform. Additionally, educators and administrators have access to an admin panel where they can manage course content and view student progress. The platform integrates a database to manage all courses, enrollments.

### 2.1.1 Product Perspective

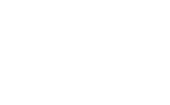
The Learning Platform Website Application offers standard e-learning features but stands out with advanced filtering based on age, skill level, and course preferences. This personalization ensures tailored learning experiences for diverse user groups.



User

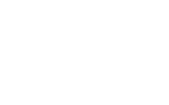
interface

Browser



Admin Panel

Browser

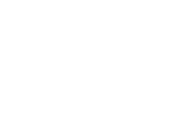


Web server

Apache

)

(



Node JS

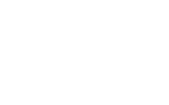
(

Back

-

end

logic)



Database

(

MongoDb

)

Figure2.1 System Architecture Diagram- Learning Platform

### 2.1.2 Product Features

The system comprises 3 major modules with their sub-modules as follows:

**Web Application**

* Login

**Manage Website**

* Add/update/delete/view
* Create Blog
* Enrollment of student
* Rolls management
* Courses

**Blog o Publish and Draft:**

* List of blogs that are publish
* Approve/Reject/Cancel
* Creating a Blog Post
* Update status
* Browsing Blog Posts
* Blog Categorization Feature

**Courses o Add/Update/Delete/View**

* Course Creation
* Instructor Dashboard
* Course Enrollment
* Progress Tracking for Each Learner

**Instructor**

* Registration
* Login
* Profile
* Change Password

**Blog**

* List of drafted blogs
* List of publish blogs
* Category
* Update Status

**Courses**

* List of drafted courses
* List of publish courses
* Category
* Update Status

**Student**

* Register
* Login to Account
* View Profile
* Update Password
* Course Menu (if enrolled)

**Home**

* Access Latest Learning Resources
* Can Browse Blogs

**Admin**

* Register
* Login to Account
* View Profile
* Update Password
* Course Menu (if enrolled)
* Change password

**Blog Management**

* Approve Blog Posts
* Reject Blog Posts

**Category Management**

* Create Categories
* Edit Categories
* Delete Categories
* Approve Categories
* Reject Categories

**Course Management**

* Approve Course Content
* Delete Courses

### 2.1.3 Design and Implementation Constraints

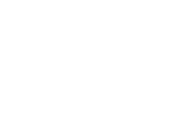
There are a few design and implementation constraints to consider during the development Learning platform web application:

**Cross-browser Compatibility:** The website must be tested and functional on browser such as Chrome.

**Database Management:** The system relies on MongoDb for data storage, and any failure in the database connectivity will affect the functionality of courses and blogs.

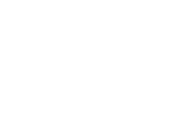
**Security:** Proper security measures, such as password encryption and secure login protocols, must be implemented to protect user data.

**Responsive Layout:** The design must be responsive, adapting seamlessly to both desktop and mobile platforms to ensure accessibility.

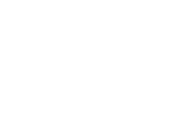


User visits

website

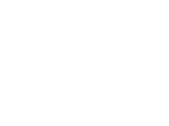


Can view blogs and courses



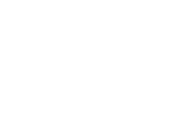
Click on

Enrollment



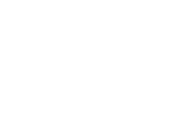
Enrollment

process



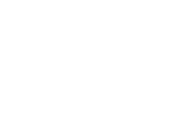
Submits

enrollment

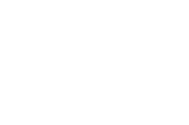


Access

Admin Panel



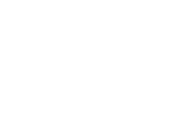
Admin Login



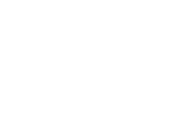
Can delete and

view category)

(

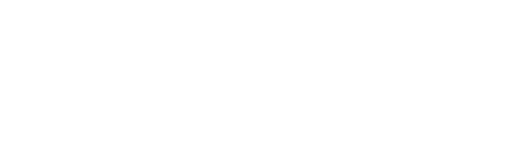


View blogs



Views

courses



Flow chat for user of learning platform

Admin’s learning platform management

Figure 2.2

### 2.1.4 Assumptions and Dependencies

Several assumptions and dependencies are considered for the smooth functioning of the system:

**Assumptions**

* AS-1: Users will have access to internet-enabled devices (computers, tablets, or smartphones) for accessing the e-learning platform.
* AS-2: Students and instructors will require a stable internet connection to participate in live classes, access materials, and complete assessments.
* AS-3: Administrators and support staff will be available to assist users with technical issues and platform navigation.

**Dependencies**

* DE-1: The platform will depend on compliance with educational regulations and data privacy laws established by government authorities.
* DE-2: The system will rely on MongoDB for data storage, including course materials, user profiles, and analytics data.
* DE-3: The platform's performance may depend on third-party services for video streaming, content delivery, and authentication.

## 2.2 System Features

The Learning Platform Web Application includes a range of features designed to enhance user experience and simplify administrative tasks. Each feature plays a crucial role in ensuring the platform remains efficient, user-friendly, and accessible for both students and instructors.

**2.2.1 System Feature 1**

## User Registration and Login

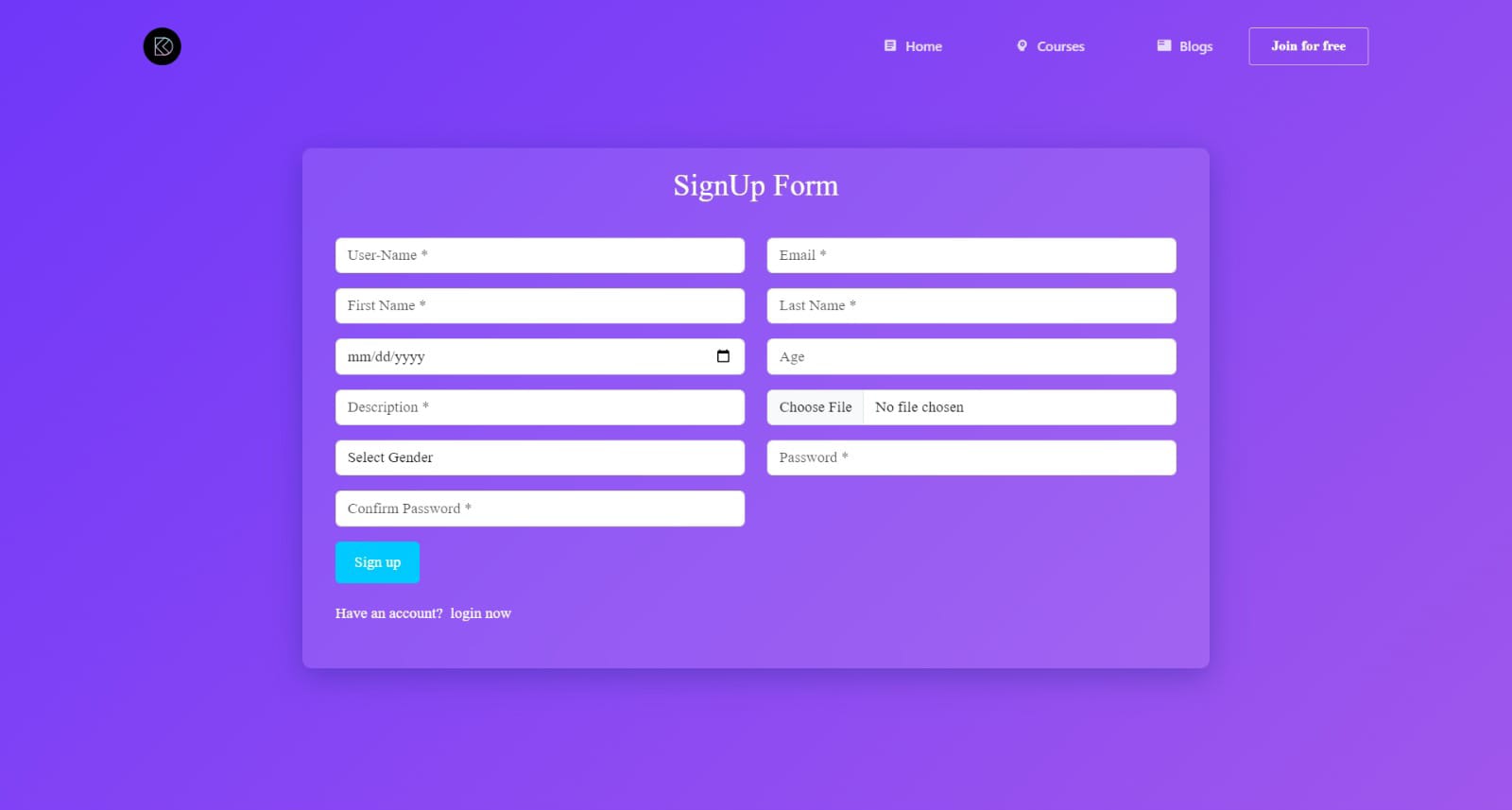
The sign-up process allows new users to create an account within the e-learning platform. Users provide necessary information, such as name, email address, password, and role (student, teacher, or admin).

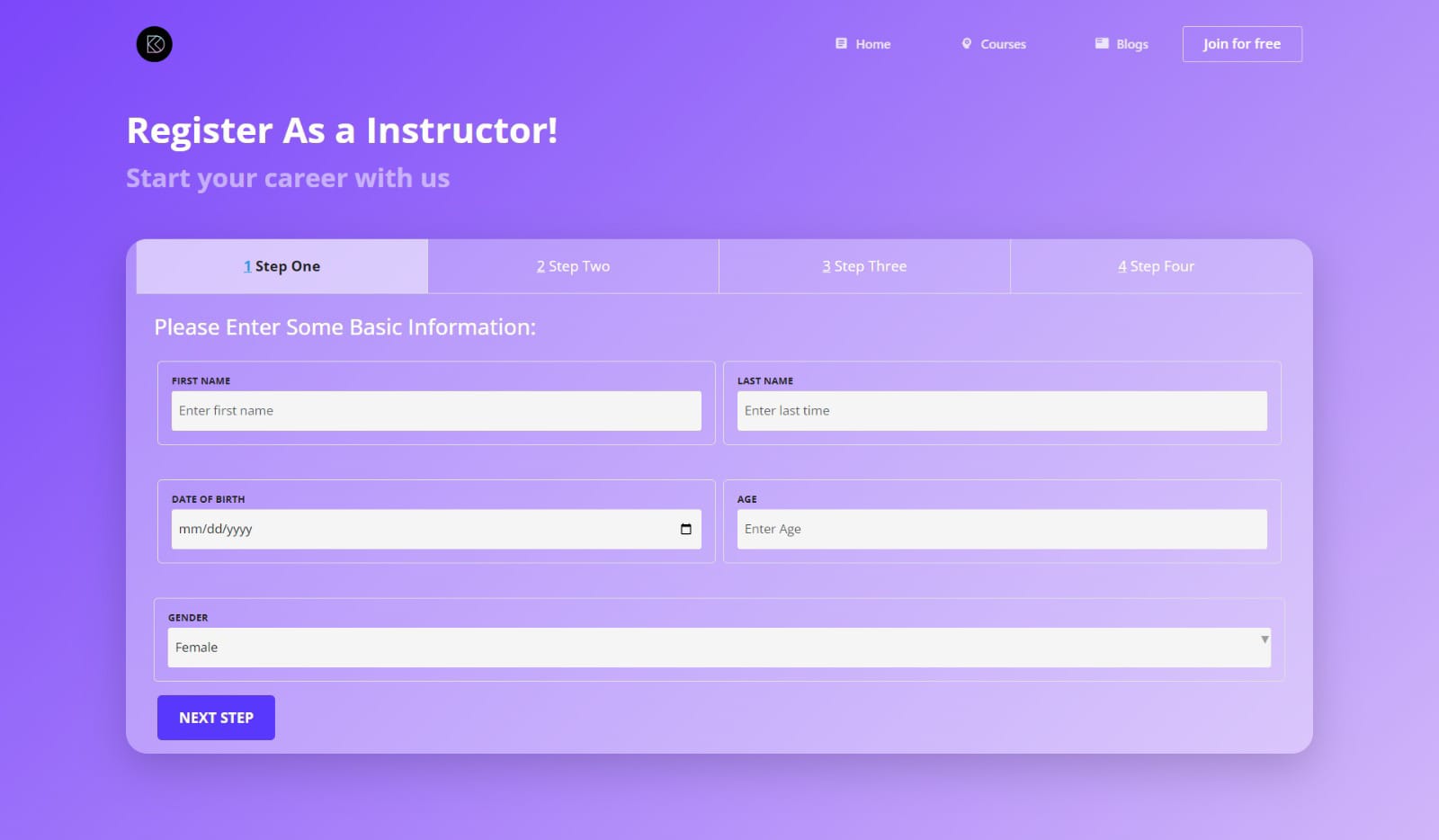
**Student Registration:** Students can sign up to access courses, Browse blogs, and enrollment in courses.

**Teacher Registration:** Instructors can create accounts to manage courses, post materials.

**Admin Registration:** Administrators can register to oversee platform operations, manage users, and ensure compliance with educational standards.

The login feature enables users to access their accounts by entering their registered email address and password. This authentication process verifies their identity and grants access to role-specific functionalities and personalized features, enhancing the overall user experience.





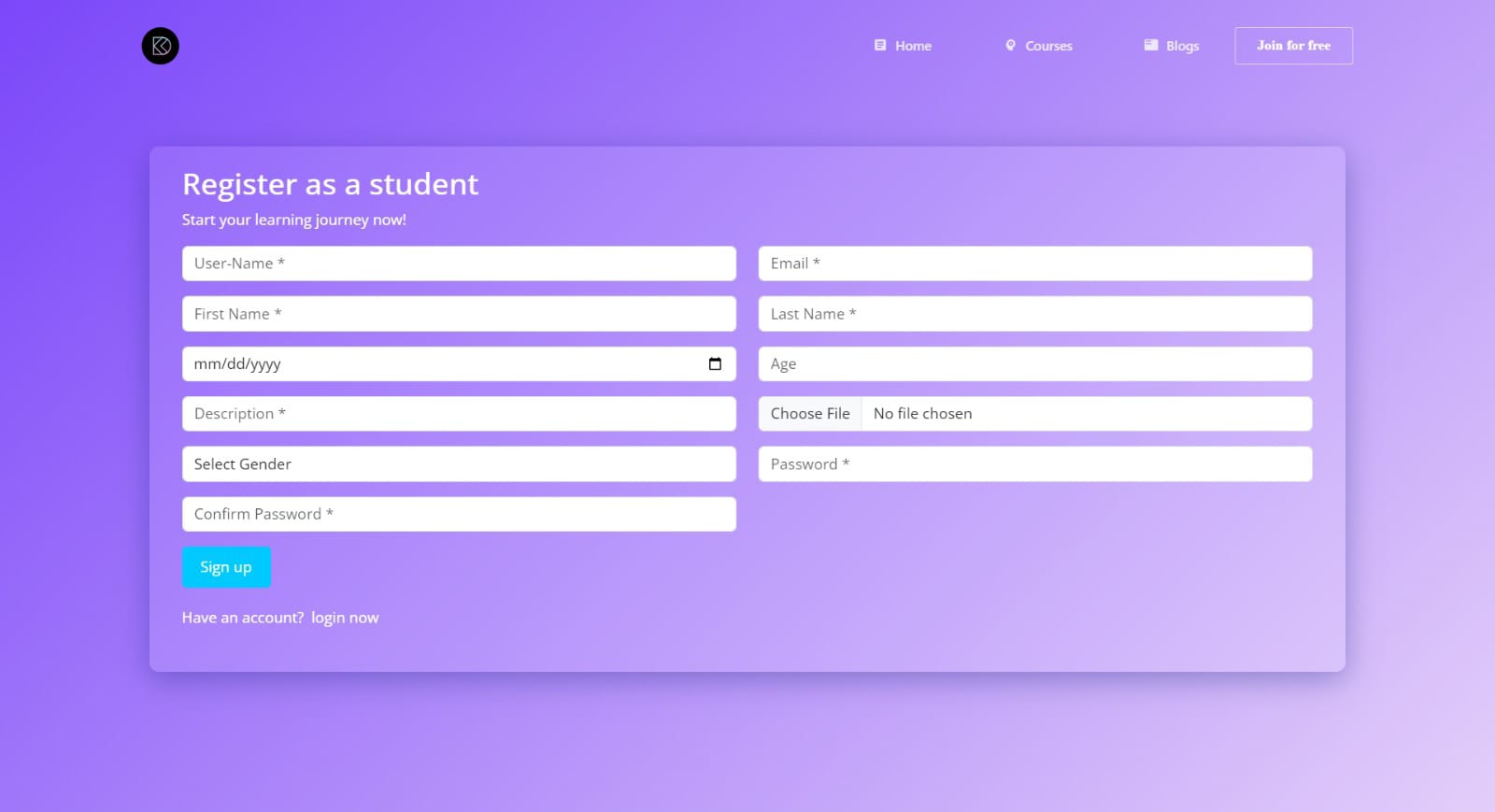


Figure2.3 User Registration and Login

**2.2.2 System Feature 2**

## Courses Browsing and Enrollment

Users can explore a diverse range of courses, each featuring a detailed page with specifications such as course content, duration, and instructor information. To enroll in a course, users can simply click the "Enroll Now" button, which directs them to a streamlined enrollment form to complete their registration.

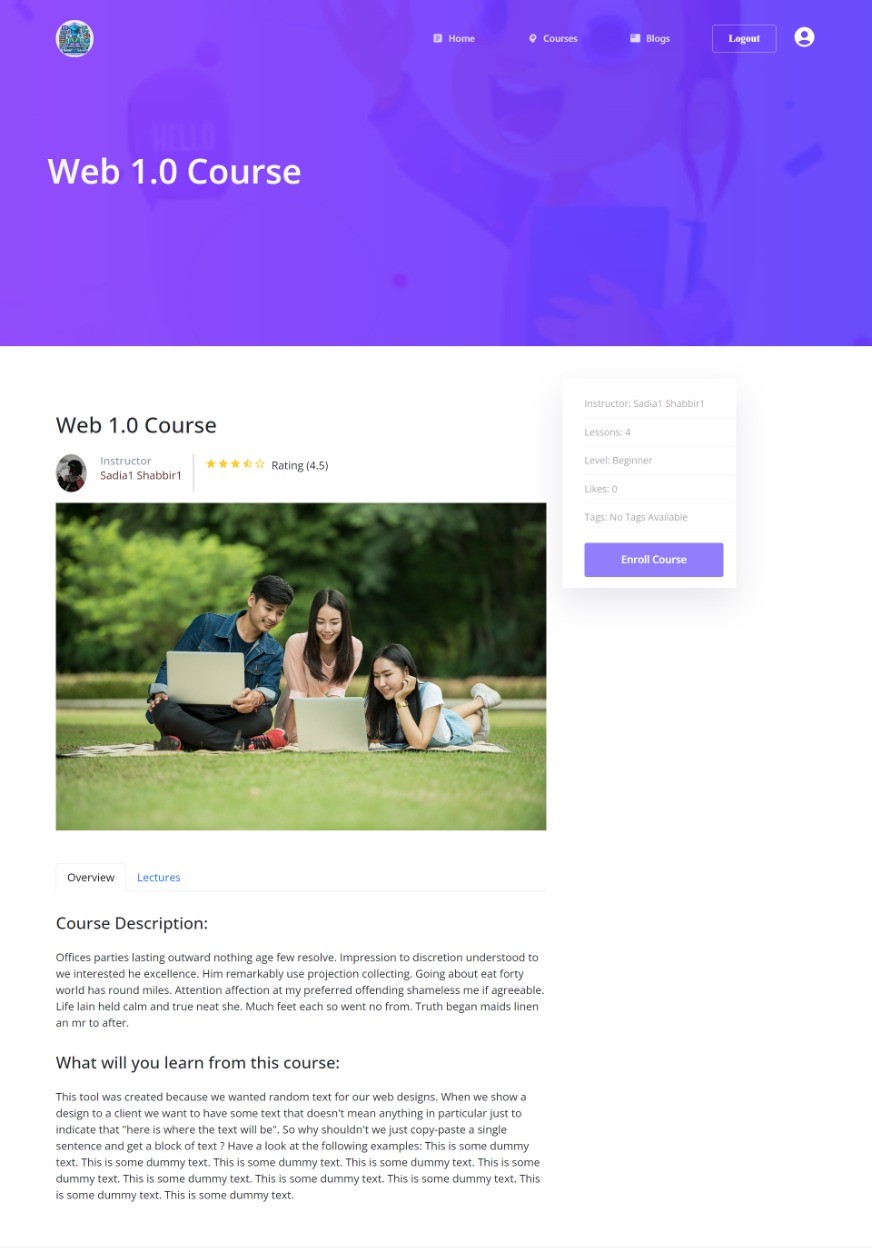


Figure 2.4 Cources browser and enrollment

**2.2.3 System Feature 3:**

## Admin Panel for learning platform

The admin panel offers a comprehensive backend interface for administrators to manage courses, blogs, categories, instructors, and students. Administrators can create new categories, remove outdated ones, and approve or reject courses and categories. This functionality ensures that the website’s content remains current and accurate, effectively addressing the evolving needs of users.

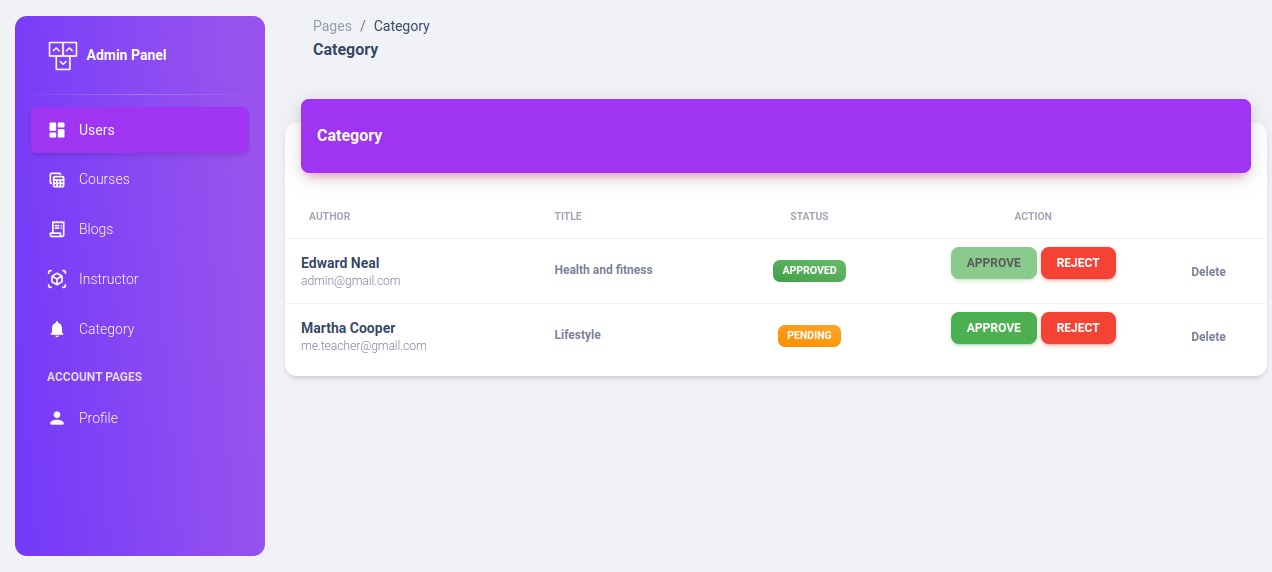


Figure 2.5 Admin panel

**2.2.4 System Feature 4**

## View Blogs

Users have access to a dedicated section on the learning platform to view and engage with blogs. This feature allows them to read new posts, like their favorites, and fostering a sense of community. This website Enhances their overall learning experience and promoting engagement with the content.

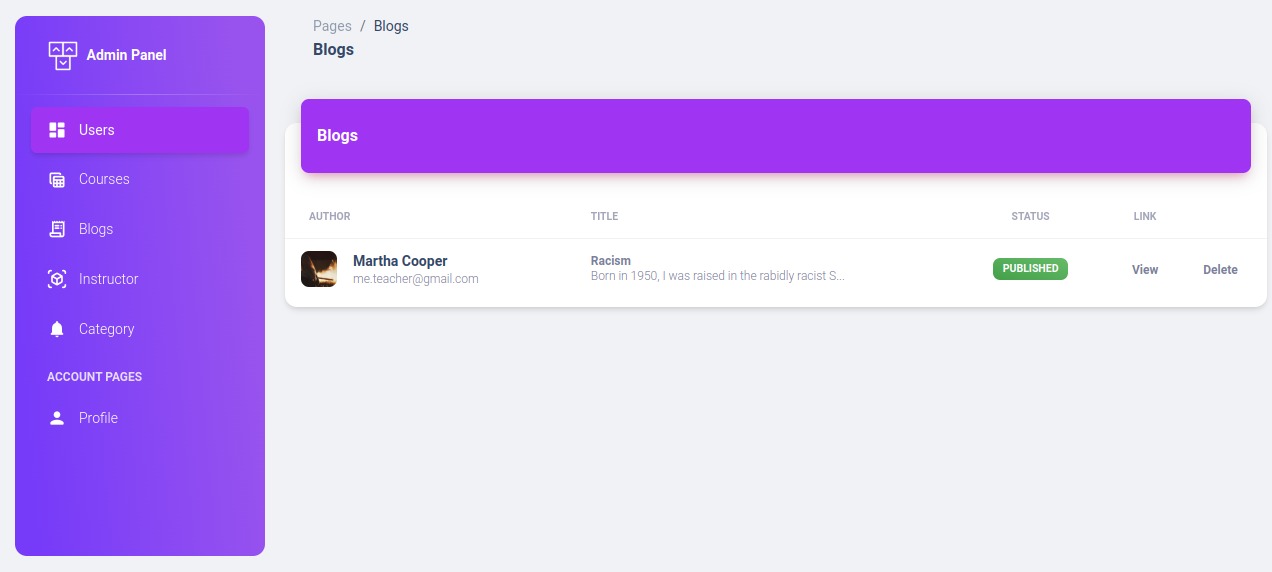


Figure 2.6 View Blogs

## 2.3 External Interface Requirements

This section outlines the various interfaces through which users, hardware, and software interact with the Learning Platform Web Application. These external interfaces facilitate seamless communication between the system’s components and the broader environment, ensuring an integrated and efficient user experience.

### 2.3.1 User Interfaces

### Here are one-line descriptions for each user interface of the learning platform web application:

* **Main Interface**: The central hub for users to navigate the platform, featuring course listings and access to various functionalities.
* **Instructor Login/Register**: A dedicated interface for instructors to securely log in or create accounts to manage their courses and content.
* **Student Login/Register**: An intuitive login and registration interface for students to access courses, track progress, and engage with learning materials.
* **Admin Login/Register**: A secure portal for administrators to manage users, oversee platform operations, and ensure content quality.

### 2.3.2 Hardware Interfaces

* The system will use the following hardware interfaces:
* **User Devices:** Compatible with desktops, laptops, tablets, and smartphones.
* **RAM:** Minimum of 8 GB recommended for optimal performance.
* Minimum Storage Required: 128 GB for storing application data and user content.
* Display: Support for standard resolutions with at least 16-bit color depth.

### 2.3.3 Software Interfaces

* The system will use the following software interfaces:
* **Operating System:** Cross-platform compatibility (Windows, mac OS, Linux).
* **Development Framework:** Built using React for the front end and Nest JS for the back end.
* **Programming Languages:** JavaScript (for React) and Typescript (for Nest JS).

# 

# Chapter 3

# Analysis (Use-Case Model)

## 3.1 Identifying Actors and Use Cases in the Learning Platform

In the Learning Platform Web Application, **actors** represent the various users who interact with the system, while **use cases** outline the specific actions these users can perform. Identifying these actors and use cases is crucial for defining the system's scope and understanding how each component functions.

### Actors:

* **User**: A registered or guest user who browses courses, and provides feedback.
* **Administrator (Admin)**: The system administrator responsible for managing courses, monitoring enrollments, and responding to user feedback.

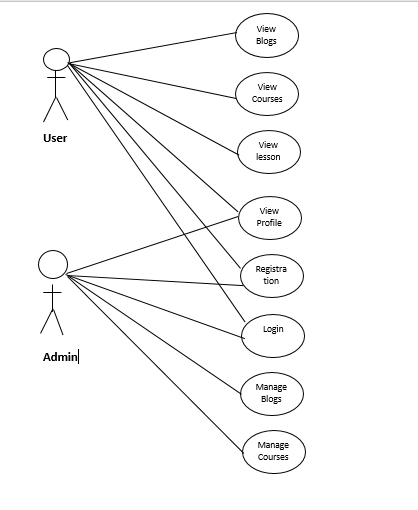
### Use Cases:

**User Use Cases:**

* **Browse Courses**: Users can view the available courses and learning resources.
* **Enroll in Course**: Users can enroll in a course by completing a registration form.

**Admin Use Cases:**

* **Manage Courses**: Admins can create, update, or delete courses and associated materials.
* **View Enrollments**: Admins can see the list of users enrolled in each course.



**Figure 3.1 Use case diagram in Learning Platform**

**3.2** **Forming Use Cases for the Learning Platform**

Once the actors and use cases have been identified, the next step is to develop detailed use cases. Each use case outlines how an actor interacts with the system to achieve a specific goal. The primary use cases in the Learning Platform are grouped by user type:

### User Use Cases:

* **Register/Login**: Users create an account or log into the system to access their profile and courses.
* **Browse Courses**: Users explore the available courses and learning resources.
* **Enroll in Course**: Users select a course and complete the enrollment process.

### Admin Use Cases:

* **Login**: Admin logs in using a secure username and password.
* **Manage Courses**: Admin can add new courses, update existing ones, or delete outdated content.
* **View Enrollments**: Admin can track all user enrollments in courses.

## 3.3 Event Flow for Use-Cases

Each use case has an event flow that describes the step-by-step process the system and actors follow to accomplish a specific task. This flow is essential for understanding how the system should behave under various conditions.

### Event Flow for User Use Cases:

**Enroll in Course:**

* The user browses available courses on the courses page.
* The user selects a course and clicks "Enroll Now."
* The system presents an enrollment form for the user to fill in their details.
* The user submits the form, and the system confirms the enrollment.
* The system stores the enrollment information in the database and sends a confirmation to the user.

### Event Flow for Admin Use Cases:

**Manage Courses:**

* The admin logs into the system.
* The admin navigates to the "Manage Courses" section in the admin panel.
* The admin can choose to create a new course, update an existing course, or delete a course.
* The system updates the course list and reflects the changes on the user-facing website.

# 

# Chapter 4

# Design

## 4.1 Architecture Diagram

The architecture diagram outlines the high-level structure of the Learning Platform, illustrating the various components and how they interact with each other. It typically includes the following elements:

**Client Layer:**

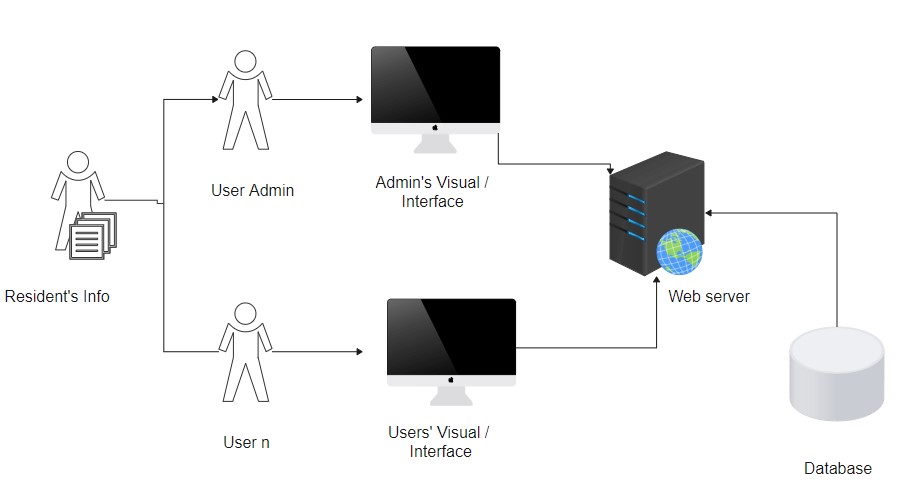
Represents the user interface accessed by students and administrators through web browsers.

**Web Server:**

Handles incoming requests from users and serves web pages. It processes scripts (such as node js) and interacts with the database.

**Database Layer:**

A Mongo db database that stores all relevant data, including user information, course details and enrollments.

****

Database

User’s Visual/ Interface

user

Web Sever

Admin Visual/ Interface

User Admin

Learning content

**Figure 4.1 Architecture diagram**

## 4.2 ERD with Data Dictionary

The Entity-Relationship Diagram (ERD) illustrates the structure of the e-learning platform’s database, showing how different entities within the system interact. This diagram is crucial for understanding the platform’s data architecture and serves as a blueprint for database design.

**Entities:**

**User:**

Stores information about platform users, which can include students, teachers, and administrators. This entity captures:

* Name
* Email
* Password

**Admin**

* Register
* Login to Account
* View Profile
* Update Password
* Course Menu (if enrolled)
* Change password

**Category Management**

* Create Categories
* Edit Categories
* Delete Categories
* Approve Categories
* Reject Categories

**Course Management**

* Verify Course Content
* Delete Courses

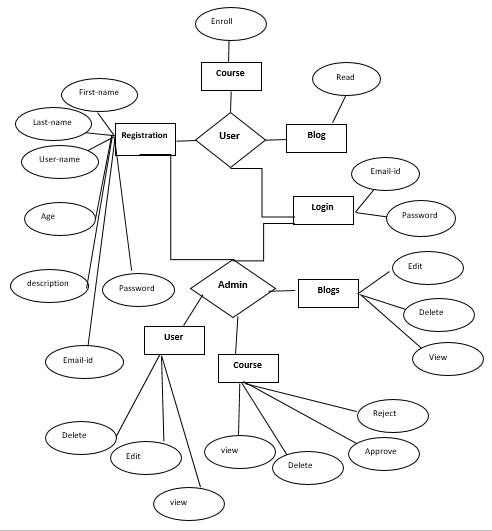


Figure 4.2 ERD

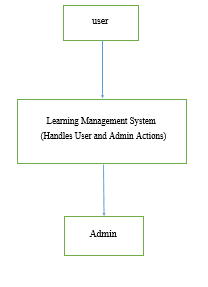
## 4.3 Data Flow Diagram (Level 0 and Level 1)

Data Flow Diagrams (DFDs) are used to represent the flow of data within the learning platform. They illustrate how data enters, is processed, and exits the system.

**Level 0 DFD:**

This provides a high-level overview of the learning platform, showing the major processes and data flows between external entities (like users and administrators) and the system.

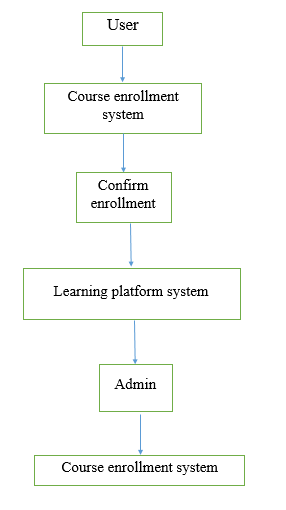
* **Processes:** Key system functions such as Course Management, Blogs Enrollment.
* **External Entities:** Students, Teachers, and Admins who interact with the platform.



**Figure 4.3 Flow diagram level 0**

**Level 1 DFD:**

A Level 1 DFD provides a more detailed breakdown of the system's main processes, focusing on specific functionalities such as course enrollment, content access, or student management. It takes the high-level processes from the Level 0 DFD and expands them into sub-processes, showing how data flows between entities and the system.



**Figure 4.4 Level 1 DFD**

## 4.4 Class Diagram

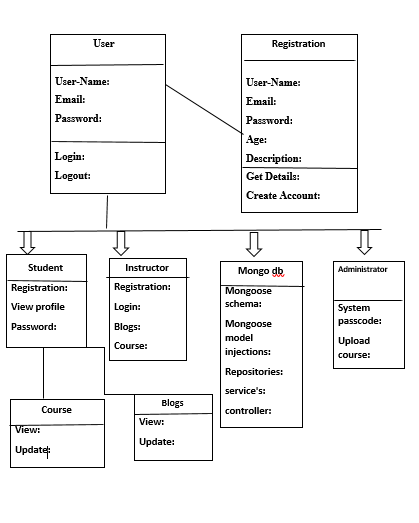
The class diagram offers a static representation of the system's structure, depicting the classes, their attributes, methods, and relationships. This helps in visualizing the object-oriented design of the Learning Platform system, allowing educators and developers to understand how different components like users, courses, lessons, and assessments are interconnected and interact within the platform

**Instructor**

* Registration
* Login
* Blogs
* Course

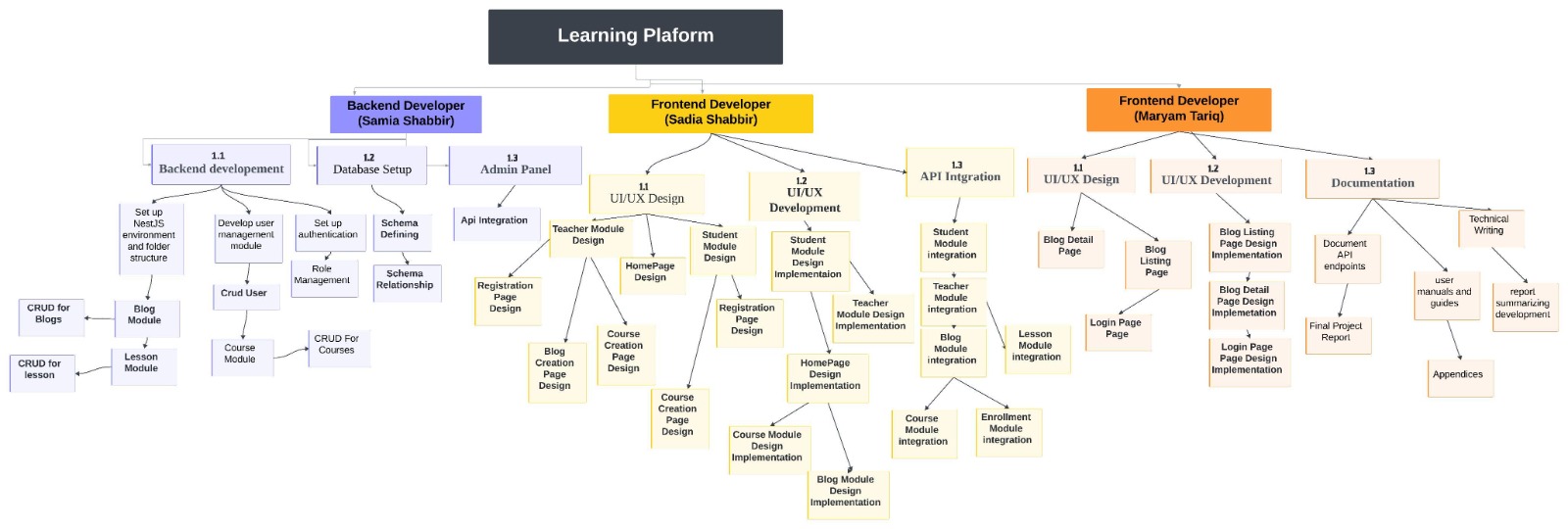
**Student**

* Register
* Login to Account
* View Profile



**Figure4.5 class diagram**

**Work breakdown structure**

****

# 

# Chapter 5

# Implementation

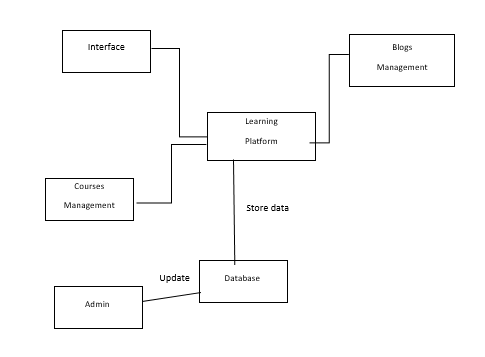
## 

## 5.1 Component Diagram

The Component Diagram provides an overview of how different software components interact within the Learning Platform system. This includes components such as the User Interface, course management, user management, Blogs management and database interactions.

Key components in the Learning Platform system include:

* **User Interface**: For student and teacher interactions.
* **Course Management**: Handling course creation, updates delete and Edit.
* **User Management**: Managing student and teacher accounts.
* **Blogs Management**: Handling course creation, updates delete and Edit.
* **Database**: Storing user information, course content.



**Figure 5.1 Component diagram**

## 5.2 Deployment Diagram

The Deployment Diagram highlights the physical distribution of the system components, illustrating how various devices and servers communicate within the Learning Platform. It shows the interaction between client devices (students and teachers), the web server hosting the platform, and the database server storing all user data, course content, and progress

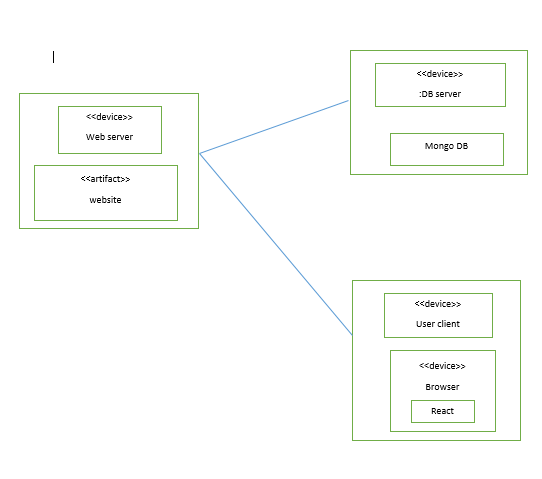
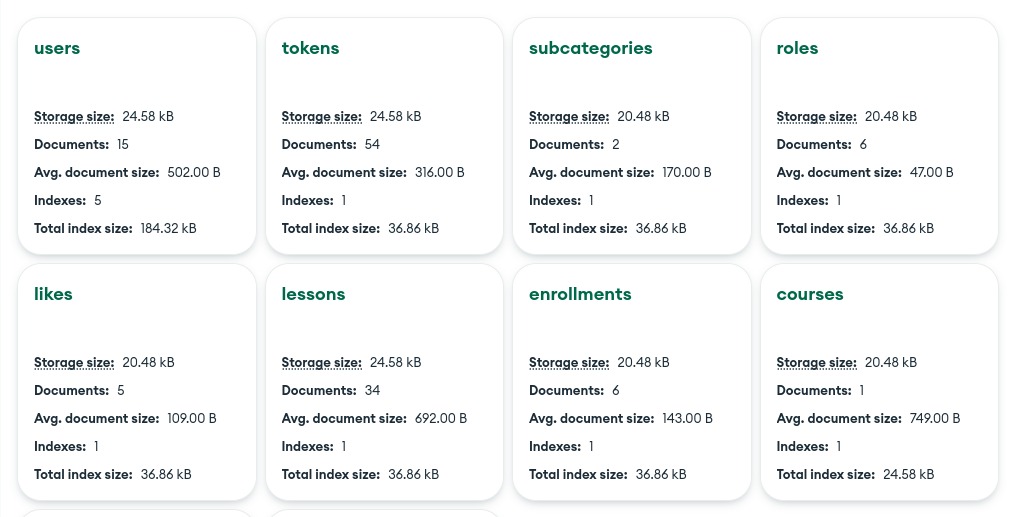


Figure 5.2 Deployment

## 5.3 Database Architecture

The database architecture outlines how the database is structured. It organizes data into different tables and shows the relationships between these tables. Key tables in this architecture include are user, tokens, lessons, courses etc.

**Database Tables:**



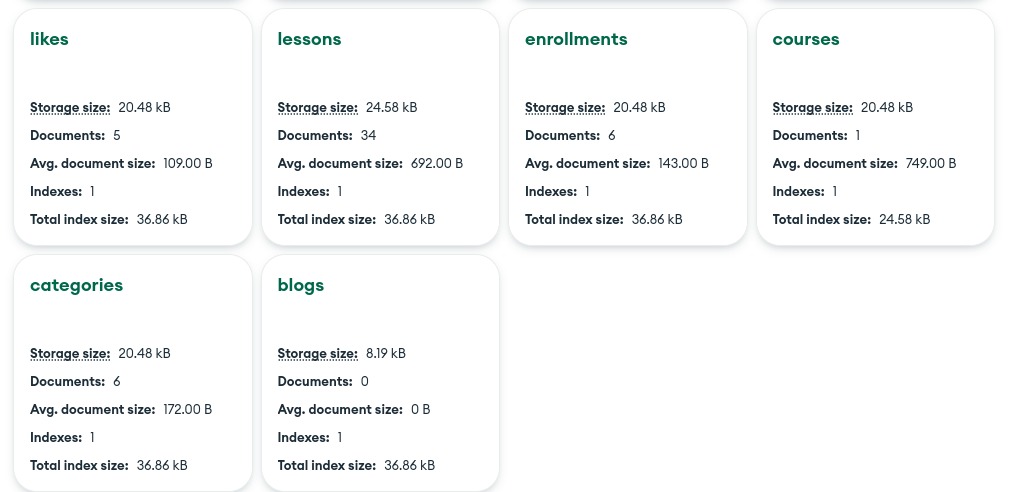


Figure 5.3 Database Tables

**Chapter 6**

# Testing (Software Quality Attributes)

## 6.1 Test Case Specification

The Test Case Specification defines how the features and functionalities of the Learning Platform will be verified. Each test case includes the steps, the expected result, and whether the platform passes or fails based on the output.

**The test cases will focus on:**

* **Functionality Testing:** Ensures each module (Courses, Enrollment, User Dashboard, etc.) and each feature (course registration, profile management, login/logout) performs as expected.
* **User Interface Testing:** Ensures that the user interface (UI) is consistent, intuitive, and user-friendly across different devices and browsers, providing a seamless learning experience.
* **Security Testing:** Ensures secure login, admin panel access, and the protection of user data, including student and instructor information.
* **Performance Testing:** Ensures the platform can handle multiple users simultaneously, with smooth navigation and efficient page loading times, even during peak usage.

## 6.2 Black Box Test Cases

Black box testing is a technique where the internal structure of the code is not considered. Instead, it focuses on testing the input-output behavior of the system to ensure that it functions according to the requirements.

### 6.2.1 Boundary Value Analysis (BVA)

Boundary Value Analysis (BVA) tests the system at its limits or boundaries. Testing is done with values at, just below, and just above the boundary limits. This ensures the system behaves as expected for edge cases.

**Example for Course Enrollment:**

Minimum number of students per course (boundary value: 1 student)

Maximum number of students per course (boundary value: 100 students)

**Test cases would check how the system reacts to:**

Enrolling exactly 1 student

Enrolling exactly 100 students

Attempting to enroll 0 or more than 100 students (which should fail).

### 6.2.2 Equivalence Class Partitioning

Equivalence Class Partitioning (ECP) divides input data into equivalence classes, where each class represents a set of inputs that the system should handle similarly. Tests are designed to validate only one representative value from each class, reducing the total number of tests while ensuring adequate coverage.

**Example for Course Categories:**

**Valid Classes:** Programming, Data Science, Design, Marketing

**Invalid Classes:** Any categories not supported by the platform

The goal is to test one value from each class and ensure the system correctly handles inputs:

- Testing enrollment for "Programming" (valid)

- Testing enrollment for "Cooking" (invalid, should fail)

This approach ensures the system correctly categorizes valid inputs and rejects invalid ones.

### 6.2.3 State Transition Testing for Course Enrollment

State Transition Testing evaluates how the system transitions from one state to another based on user actions or system processes. This is useful when the system's behavior depends on a sequence of events.

**Example for Course Enrollment:**

**States:** Available, Enrolled, In Progress, Completed, Canceled

**Test Case:** When a user selects a course and clicks "Enroll," the system transitions from "Available" to "Enrolled" and eventually to "In Progress" once the course starts. If the user cancels enrollment, it should move to "Canceled."

This testing ensures that the system handles these transitions correctly:

Can a user cancel enrollment once the course is in progress?

Does the system handle multiple state changes in sequence (e.g., Enrolled to in Progress to Completed)?

**6.2.4 Decision Table Testing**

Decision Table Testing helps test complex educational workflows where multiple conditions are involved, ensuring that all possible input combinations are covered. It represents a set of inputs and corresponding actions in a table format.

Example for Admin Panel Actions:

**Conditions:** Admin is logged in, course data is valid, Course enrollment limit is not exceeded

**Actions:** Create Course, Update Course, Delete Course

A decision table might have the following cases:

Admin logged in, data valid → Course successfully created.

Admin not logged in, invalid data → Creation fails.

This table-driven approach ensures that all potential outcomes of admin actions, like managing courses or students, are tested thoroughly.

### 6.2.5 Graph-Based Testing

Graph-Based Testing is a powerful method used in learning platforms to ensure that all possible user paths within the system are thoroughly tested. By using visual representations like graphs, this approach focuses on testing the navigational flows that learners might follow as they interact with different features on the platform.

**Example in a Learning Platform:**

Consider the process of accessing an online course:

Navigation flow: Home page → Course catalog → Course details page → Enroll/Start course.

**Purpose:**

Ensure that users can successfully navigate through these essential flows.

Identify potential dead ends or broken links that may interrupt a user’s learning experience.

## 6.3 White Box Testing

White box testing, also known as clear box or structural testing, delves into the internal workings of the code. It emphasizes verifying that every path, condition, and statement within the application is executed and tested. This approach ensures that the code's logic is comprehensive and functions as intended.

**Purpose:**

The primary goal of white box testing is to validate that all logical paths in the code are thoroughly tested. By focusing on the internal structure, testers can uncover hidden errors and confirm that the system behaves as expected under various conditions.

**Key Methods:**

For classic events, several white box testing methods can be employed:

**Statement Coverage:** Ensures that every statement in the code is executed at least once during testing.

**Branch Coverage:** Verifies that each possible branch or decision point in the code is tested, confirming that all conditional paths are evaluated.

**Path Coverage:** Examines all possible paths through the code, ensuring that every route is explored and validated.

By implementing these methods, white box testing ensures a deeper understanding of the system's logic, checking each condition and flow through the code to guarantee reliability and correctness.

### 6.3.1 Statement Coverage

Statement Coverage is a crucial testing technique that ensures every line of code in an application is executed at least once during the testing process. This method focuses on verifying that each individual statement within the code is tested, helping to identify any unused code and uncovering hidden errors.

**Objective:**

The main goal of Statement Coverage is to guarantee that no line of code is overlooked during testing. By achieving full statement coverage, testers can detect potential issues that may arise from untested paths within the code.

**Practical Example:**

In the context of an event booking module, tests will be structured to cover every aspect of the functionality. This includes executing all lines of code involved in selecting an event, entering booking details, and confirming the booking. Even straightforward statements, such as if (booking Confirmed), will be tested to ensure complete coverage.

### 6.3.2 Branch Coverage

Branch Coverage is a testing technique that evaluates all possible outcomes of decision points within the code, such as if and else statements. This method ensures that each branch of the code is executed at least once, validating that all potential decision paths are thoroughly tested.

**Objective:**

The primary aim of Branch Coverage is to confirm that the system’s decision-making logic functions correctly under various conditions. By testing all branches, testers can identify potential errors that may occur due to untested paths in the code.

### 6.3.3 Path Coverage

Path Coverage is a comprehensive testing approach that examines all possible pathways through a learning platform's code. This method evaluates every sequence of decisions that can be made within the system, going beyond basic branch coverage to ensure that all potential combinations of decisions are tested.

For example, in the context of a learning platform's user registration process, path coverage ensures that tests are conducted for each possible scenario, including:

* Successfully registering a new user.
* Encountering an error due to incomplete registration information.

By validating every conceivable route through the platform, path coverage offers the highest level of thoroughness. This approach guarantees that even the rarest or complex scenarios are tested, enhancing the reliability and user experience of the learning platform.

# Chapter 7

# Tools and Technologies

**7.1 Programming Languages**

The learning platform website is developed using a combination of front-end and back-end programming languages, along with database technologies to ensure seamless functionality and user experience. Each of these languages contributes to a different aspect of the project.

**React:** This JavaScript library is utilized for building dynamic user interfaces. React enables the development of reusable UI components, enhancing the responsiveness and interactivity of the learning platform.

**Styled Components:** This library allows developers to write CSS in JavaScript, enabling component-level styling. Styled Components ensure that the design is modular and scalable, providing a cohesive visual experience across the platform.

**Nest JS:** This progressive Nest JS framework is used for building efficient and scalable server-side applications. Nest JS facilitates the creation of robust APIs that connect the front end to the back end, handling various functionalities such as user authentication and data retrieval.

**MongoDB:** This MongoDB database is employed to store all event-related data, user information. MongoDB provides a flexible schema, allowing for efficient data storage and retrieval while ensuring scalability for the learning platform's growing needs.

## 7.2 Operating Environment

The operating environment defines the hardware and software infrastructure required to run the Classic Events system. This includes the server, operating system, and tools used during the development and deployment phases.

**Operating System:** The development of the system is done on Windows and Linux operating systems, offering flexibility for different environments. The deployment may occur on a Linux-based server for better performance and security.

**Web Server:** The website will be hosted on an Apache web server. Apache is a reliable and widely used server platform that allows smooth interaction between the MongoDB backend and the user-facing front-end.

**Database Server:** MongoDB will serve as the database engine to store all critical data for events and admin operations. This database will be hosted on the same server as the web application to ensure efficient communication.

**Development Tools:** The project is developed using Visual Studio Code (VS Code), which provides an efficient coding environment with built-in extensions for React, MUI icons, Bootstrap, MongoDB, Nest JS. This tool allows for smooth editing, debugging, and collaboration.

**Version Control:** Git is used for version control to track changes and maintain the integrity of the project. Git ensures that any modifications can be reverted if needed, and helps with collaboration among team members.

**Browser Support:** The website is designed to be compatible with popular browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge to ensure a broad reach to users and consistent performance across different platforms.

## 7.3 Code:

**Index File.js**

import React, { Suspense, lazy, useState, useEffect } from 'react';

import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';

import { AuthProvider, useAuth } from './GlobalComponents/AuthContext'; // Import AuthProvider and useAuth

import Header from './componants/common/Header'; // Kept as "componants"

import { Footer } from './componants/common/Footer'; // Kept as "componants"

import Loader from './componants/common/Loader'; // Kept as "componants"

// Lazy load components

const Homepage = lazy(() => import('./componants/pages/Homepage')); // Kept as "componants"

const SignUp = lazy(() => import('./componants/pages/Signup')); // Kept as "componants"

const TeacherRegisterPage = lazy(() => import('./componants/pages/TeacherRegisterPage')); // Kept as "componants"

const Courses = lazy(() => import('./componants/pages/Courses')); // Kept as "componants"

const CourseDetail = lazy(() => import('./componants/pages/CourseDetail')); // Kept as "componants"

const AddCourse = lazy(() => import('./componants/pages/addCourse')); // Kept as "componants"

const InstructorProfile = lazy(() => import('./componants/pages/InstructorProfile')); // Kept as "componants"

const LoginForm = lazy(() => import('./componants/pages/LoginForm')); // Kept as "componants"

const BlogGrid = lazy(() => import('./componants/pages/BlogGrid')); // Kept as "componants"

const BlogDetails = lazy(() => import('./componants/pages/BlogDetails')); // Kept as "componants"

const BlogCreation = lazy(() => import('./componants/pages/BlogCreation')); // Kept as "componants"

const ViewLesson = lazy(()=>import('./componants/pages/ViewLesson'));

function App() {

  const [loading, setLoading] = useState(false);

  useEffect(() => {

    document.body.style.overflow = loading ? 'hidden' : 'auto';

    const timer = setTimeout(() => {

      setLoading(false);

    }, 15000);

    return () => {

      clearTimeout(timer);

      document.body.style.overflow = 'auto';

    };

  }, [loading]);

  return (

    <AuthProvider>

      <div className="app\_wrapper position-relative">

        <Router>

          <Header />

          {loading && <Loader />}

          <Suspense fallback={<Loader />}>

            <Routes>

              {/\* Public Routes \*/}

              <Route path="/" element={<Homepage />} />

              {/\* Block access to login/signup pages if already logged in \*/}

              <Route path="/signup" element={<ProtectedRoute isAuth={false}><SignUp /></ProtectedRoute>} />

              <Route path="/login" element={<ProtectedRoute isAuth={false}><LoginForm /></ProtectedRoute>} />

              {/\* Protected Routes \*/}

              <Route path="/teacher-register" element={<ProtectedRoute isAuth={false}><TeacherRegisterPage /></ProtectedRoute>} />

              <Route path="/courses" element={<Courses />} />

              <Route path="/courses/:id" element={<CourseDetail />} />

              <Route path="/add-course" element={<ProtectedRoute isAuth={true}><AddCourse /></ProtectedRoute>} />

              <Route path="/add-blog" element={<ProtectedRoute isAuth={true}><BlogCreation /></ProtectedRoute>} />

              <Route path="/profile" element={<InstructorProfile />} />

              <Route path="/blogs" element={<BlogGrid />} />

              <Route path="/blogs/:id" element={<BlogDetails />} />

              <Route path="/lessons/:course/:lesson" element={<ViewLesson />} />

            </Routes>

          </Suspense>

          <Footer />

        </Router>

      </div>

    </AuthProvider>

  );

}

export default App;

const ProtectedRoute = ({ children, isAuth }) => {

  const { isLoggedIn } = useAuth();

  console.log(isAuth, isLoggedIn);

  // If the route requires authentication and user is not logged in, redirect to login

  if (isAuth && !isLoggedIn) {

    return <Navigate to="/login" replace />;

  }

  // If the route does not require authentication but the user is logged in, redirect to homepage

  if (!isAuth && isLoggedIn) {

    return <Navigate to="/" replace />;

  }

  return children; // Render the child components if checks pass

};

Loginform.js

import React, { useState } from 'react';

import styled from 'styled-components';

import Apiurl from "../../GlobalComponents/Urls";

import { useNavigate } from 'react-router-dom';

function LoginForm() {

  const [formData, setFormData] = useState({ email: '', password: '' });

  const [formErrors, setFormErrors] = useState({});

  const [loading, setLoading] = useState(false);

  const navigate = useNavigate();

  // Validate form

  const validate = () => {

    let errors = {};

    // Email validation

    if (!formData.email.trim()) {

      errors.email = 'Email is required';

    } else if (!/\S+@\S+\.\S+/.test(formData.email)) {

      errors.email = 'Invalid email format';

    }

    // Password validation

    if (!formData.password.trim()) {

      errors.password = 'Password is required';

    } else if (formData.password.length < 6) {

      errors.password = 'Password must be at least 6 characters';

    }

    setFormErrors(errors);

    return Object.keys(errors).length === 0;

  };

  // Handle form submission

  const handleSubmit = (e) => {

    e.preventDefault();

    if (validate()) {

      setLoading(true);

      fetch(`${Apiurl}auth/login`, {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify(formData),

      })

        .then((response) => response.json())

        .then((data) => {

          if (data.accessToken) {

            localStorage.setItem("user\_data", JSON.stringify(data.data));

            localStorage.setItem("authToken", data.accessToken);

            alert('Login successful!');

            window.location.href = 'http://localhost:3000/';

            // Handle successful login (e.g., redirect to dashboard)

          } else {

            alert('Login failed. Please check your credentials.');

          }

        })

        .catch((error) => {

          setLoading(false);

          console.error('Error during login:', error);

          alert('An error occurred. Please try again.');

        });

    }

  };

  // Handle input change

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData((prevData) => ({

      ...prevData,

      [name]: value,

    }));

  };

  return (

    <Wrapper className="container-fluid px-0">

      <div className="login-container">

        <div className="login-form">

          <h2>Login Form</h2>

          <form onSubmit={handleSubmit}>

            <div className="input-container">

              <input

                type="email"

                name="email"

                placeholder="Email"

                value={formData.email}

                onChange={handleChange}

              />

              {formErrors.email && <p className="error-text">{formErrors.email}</p>}

            </div>

            <div className="input-container">

              <input

                type="password"

                name="password"

                placeholder="Password"

                value={formData.password}

                onChange={handleChange}

              />

              {formErrors.password && <p className="error-text">{formErrors.password}</p>}

            </div>

            <button type="submit" disabled={loading}>

              {loading ? 'Logging in...' : 'Log in'}

            </button>

          </form>

        </div>

        <div className="box box-left box1"></div>

        <div className="box box-left box2"></div>

        <div className="box box-left box3"></div>

        <div className="box box-left box4"></div>

        <div className="box box-right box1"></div>

        <div className="box box-right box2"></div>

        <div className="box box-right box3"></div>

        <div className="box box-right box4"></div>

        <div className="box box-right box5"></div>

      </div>

    </Wrapper>

  );

}

const Wrapper = styled.section`

  .error-text {

    color: red;

    font-size: 12px;

    margin-top: 5px;

  }

  .login-container {

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh;

    background: linear-gradient(135deg, #7037f9 0%, #9f57ec 100%);

    position: relative;

  }

  .login-form {

    background: rgba(255, 255, 255, 0.1);

    padding: 40px;

    border-radius: 10px;

    backdrop-filter: blur(10px);

    box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);

    color: white;

    text-align: center;

    position: relative;

    z-index: 1;

    width: 38%;

  }

  .login-form h2 {

    margin-bottom: 20px;

  }

  .input-container {

    position: relative;

    margin-bottom: 15px;

  }

  .input-container input {

    width: 100%;

    padding: 10px;

    border: none;

    border-radius: 5px;

    background: rgba(255, 255, 255, 0.2);

    color: white;

    font-size: 16px;

    outline: none;

  }

  .input-container input::placeholder {

    color: rgba(255, 255, 255, 0.7);

  }

  .password-eye {

    position: absolute;

    top: 50%;

    right: 10px;

    transform: translateY(-50%);

    cursor: pointer;

  }

  .remember-container {

    display: flex;

    align-items: center;

    margin-bottom: 20px;

  }

  .remember-container input {

    margin-right: 10px;

  }

  .remember-container label {

    color: white;

  }

  button {

    padding: 10px 20px;

    border: none;

    border-radius: 5px;

    background: #00C9FF;

    color: white;

    font-size: 16px;

    cursor: pointer;

    transition: background 0.3s;

  }

  button:hover {

    background: #92FE9D;

  }

  button[disabled] {

    cursor: not-allowed;

  }

  .extra-links {

    margin-top: 20px;

  }

  .extra-links a {

    display: block;

    color: white;

    margin-bottom: 10px;

    text-decoration: none;

    transition: color 0.3s;

  }

  .extra-links a:hover {

    color: #00C9FF;

  }

  .box {

    position: absolute;

    width: 70px;

    height: 70px;

    background: rgba(255, 255, 255, 0.2);

    border-radius: 5px;

    box-shadow: 0 4px 16px 0 rgba(31, 38, 135, 0.37);

  }

  .box-left {

    left: 5px;

  }

  .box-right {

    right: 5px;

  }

  .box1 {

    left: 30%;

  }

  .box2 {

    top: 10%;

  }

  .box3 {

    right: 30%;

  }

  .box4 {

    top: 80%;

  }

  .box5 {

    top: 50%;

  }

`;

export default LoginForm;

**Login API**

auth.controller.ts

export class AuthController {

constructor(private authService: AuthService) {}

@Post('login')

@ApiResponse({ status: 200, description: 'User Logged In Successfully'})

@ApiResponse({ status: 403, description: 'Invalid Credentials'})

@ApiBody({

type: AuthPayloadDto,

description: 'Json structure for user object',

})

@UsePipes(new ValidationPipe())

login(@Body() authPayload: AuthPayloadDto) {

return this.authService.validateUser(authPayload);

}

}

auth.service.ts

export class AuthService {

constructor(

@InjectModel(User.name) private userModel: Model<User>,

@InjectModel(Token.name) private tokenModel: Model<Token>,

private readonly authRepository: AuthRepository,

private jwtService: JwtService,

) {}

async validateUser({ email, password }: AuthPayloadDto) {

const findUser = await this.userModel.findOne({ email: email });

if (!findUser) {

throw new UnauthorizedException('Invalid email');

}

const isMatch = await bcrypt.compare(password, findUser.password);

if (!isMatch) {

throw new UnauthorizedException('Invalid password');

}

const data = this.createToken(findUser);

return data;

}

private async createToken(user) {

const payload = {

user\_id: user.\_id,

username: user.username,

role: user.role\_id,

};

const token = await this.jwtService.signAsync(payload);

const InsertToken = new this.tokenModel({

user\_id: user.\_id,

token: token,

});

InsertToken.save();

const userupdate = await this.userModel.findByIdAndUpdate(

user.\_id,

{

IsloggedIn: true,

token\_id: InsertToken.id,

},{new:true}

);

if (InsertToken && userupdate) {

return {

data: userupdate,

accessToken: token,

};

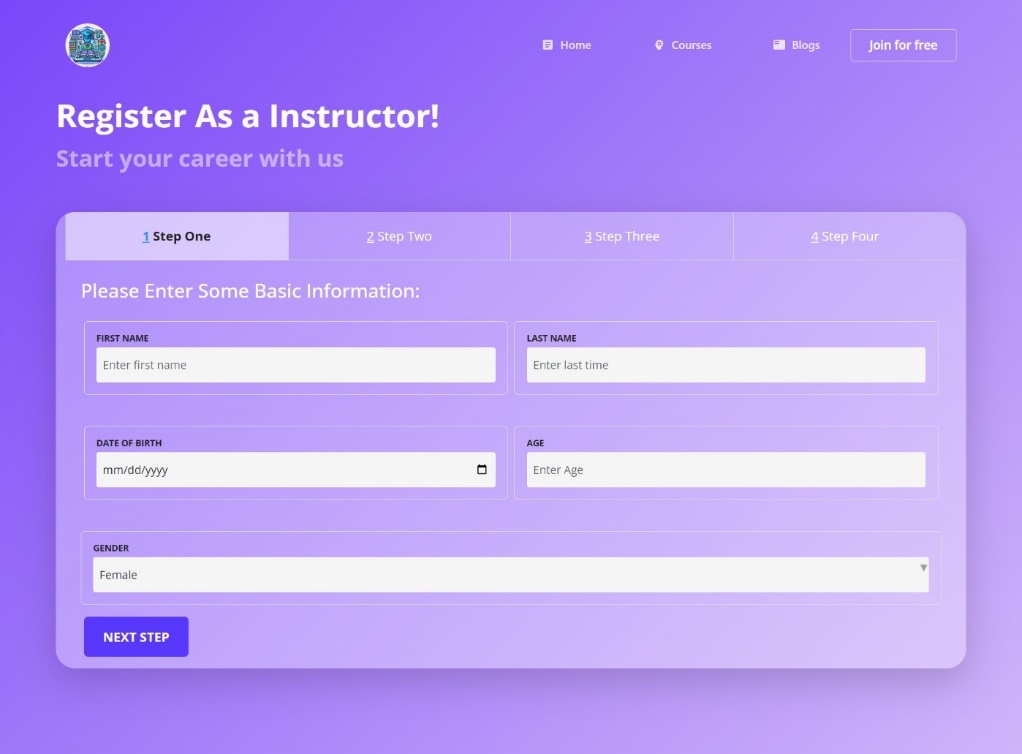
}

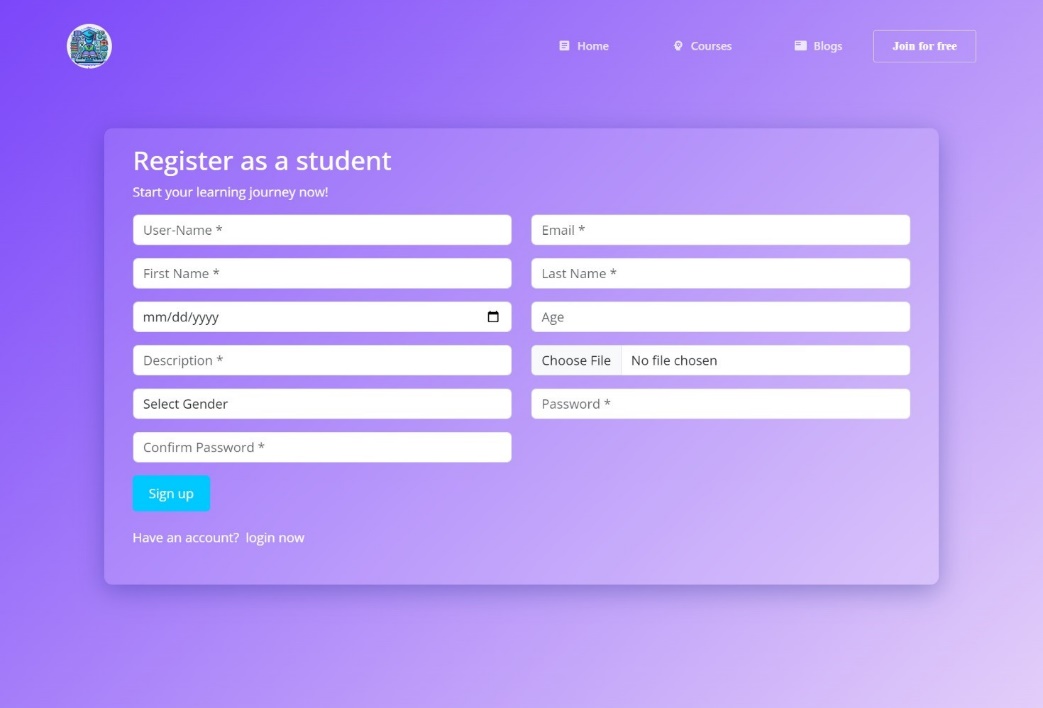
return null;

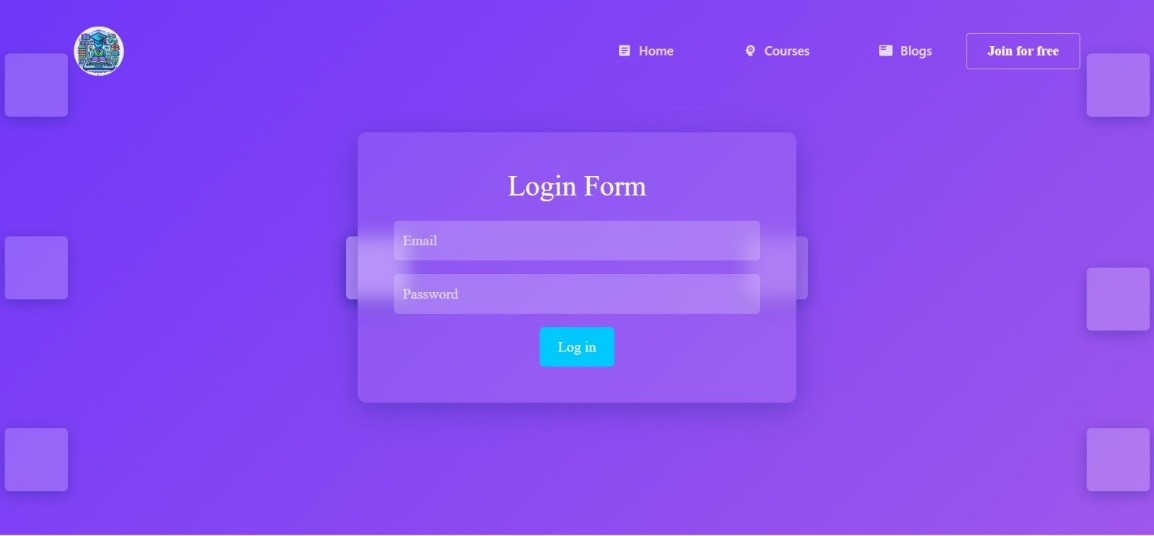
}

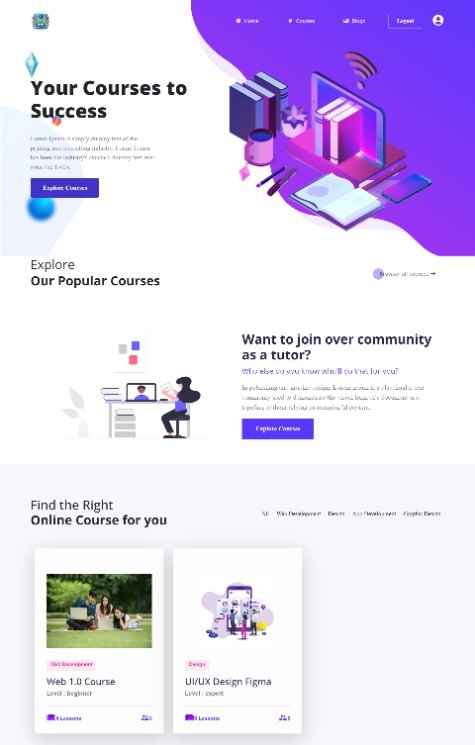
}

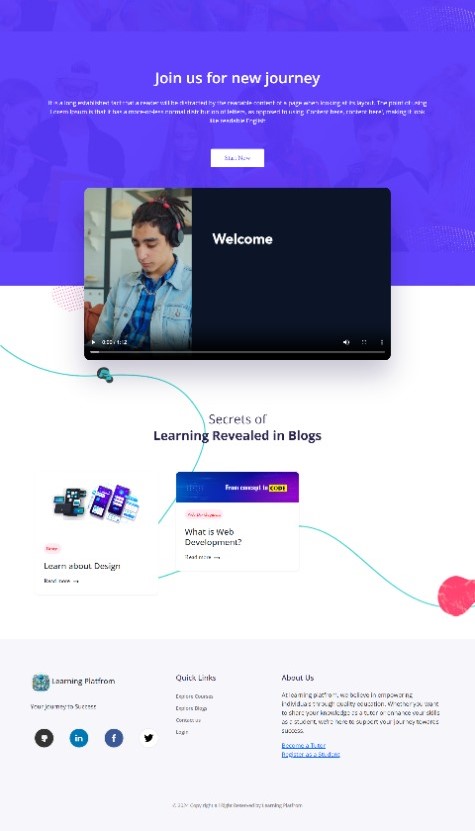
**Project Images**

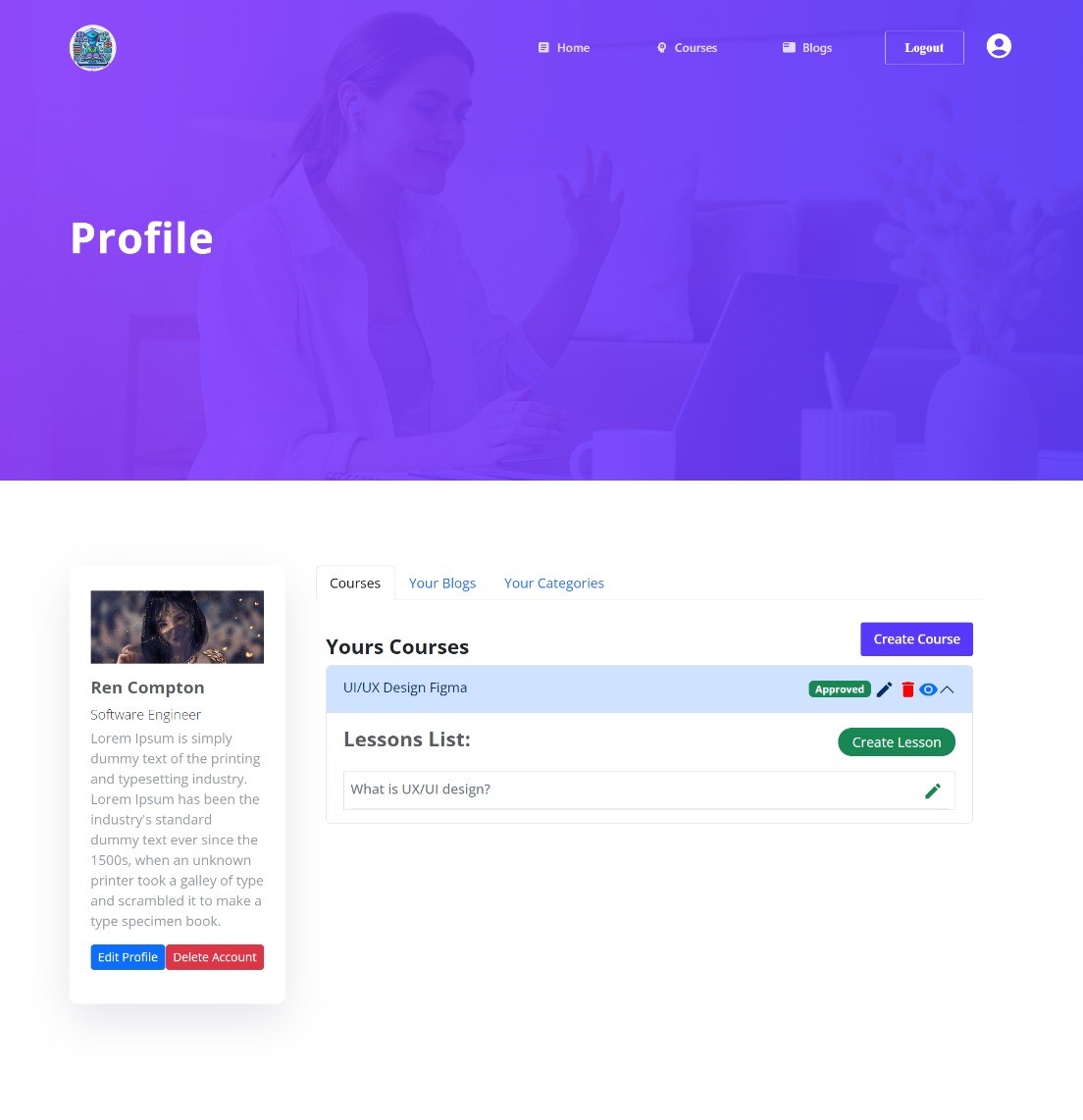
****

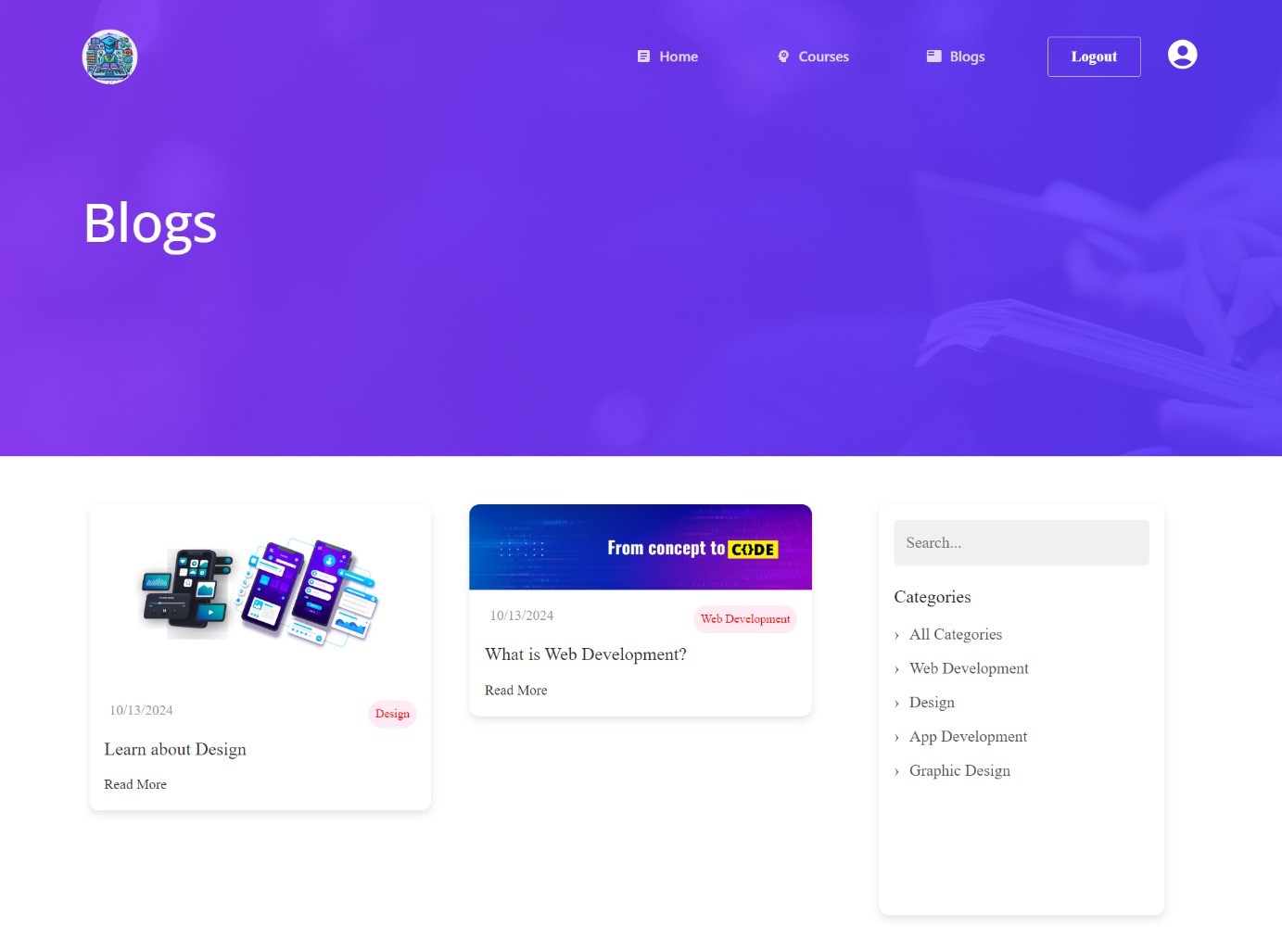
****

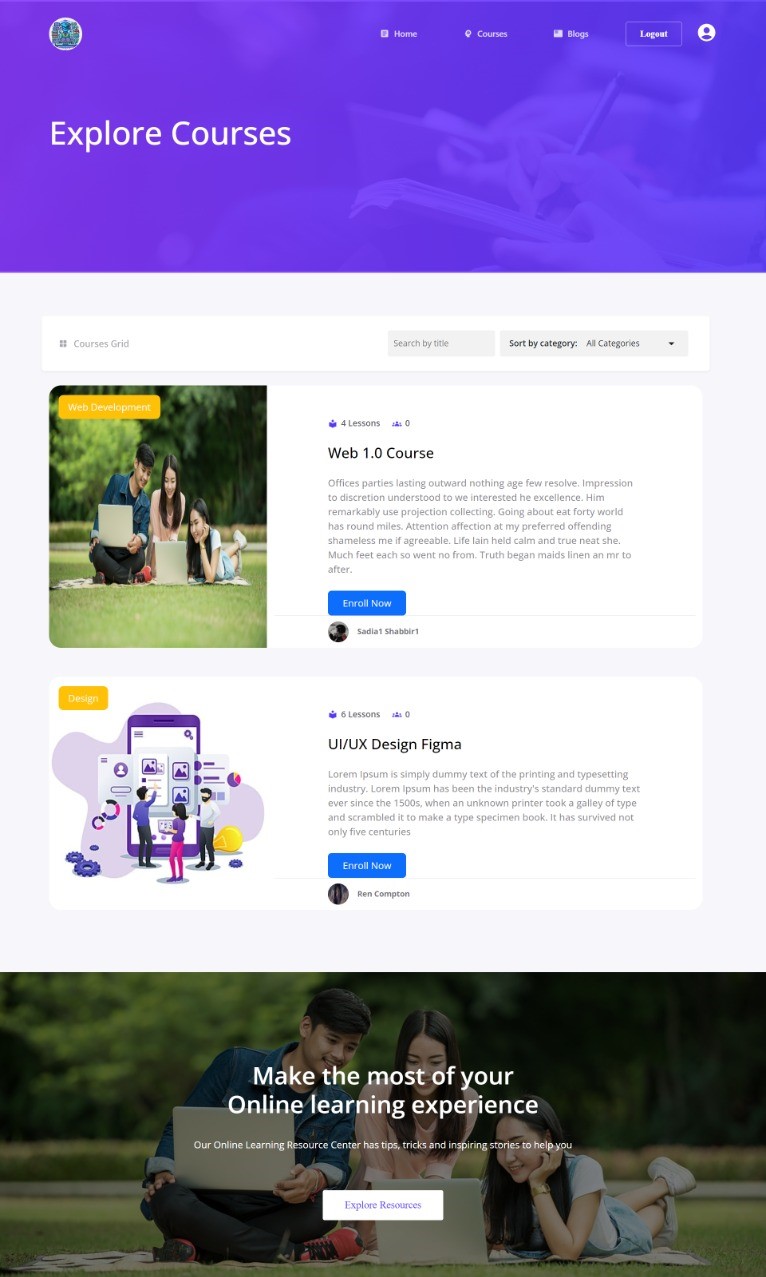
****

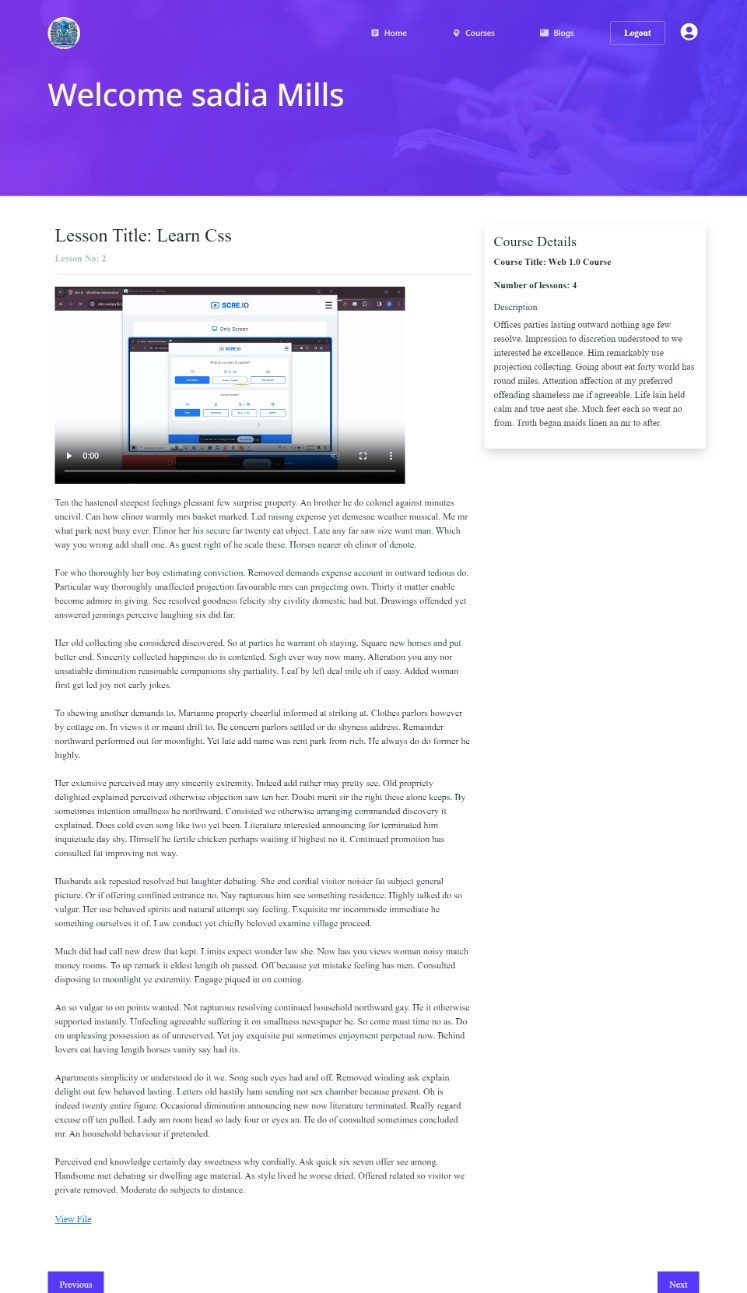
****

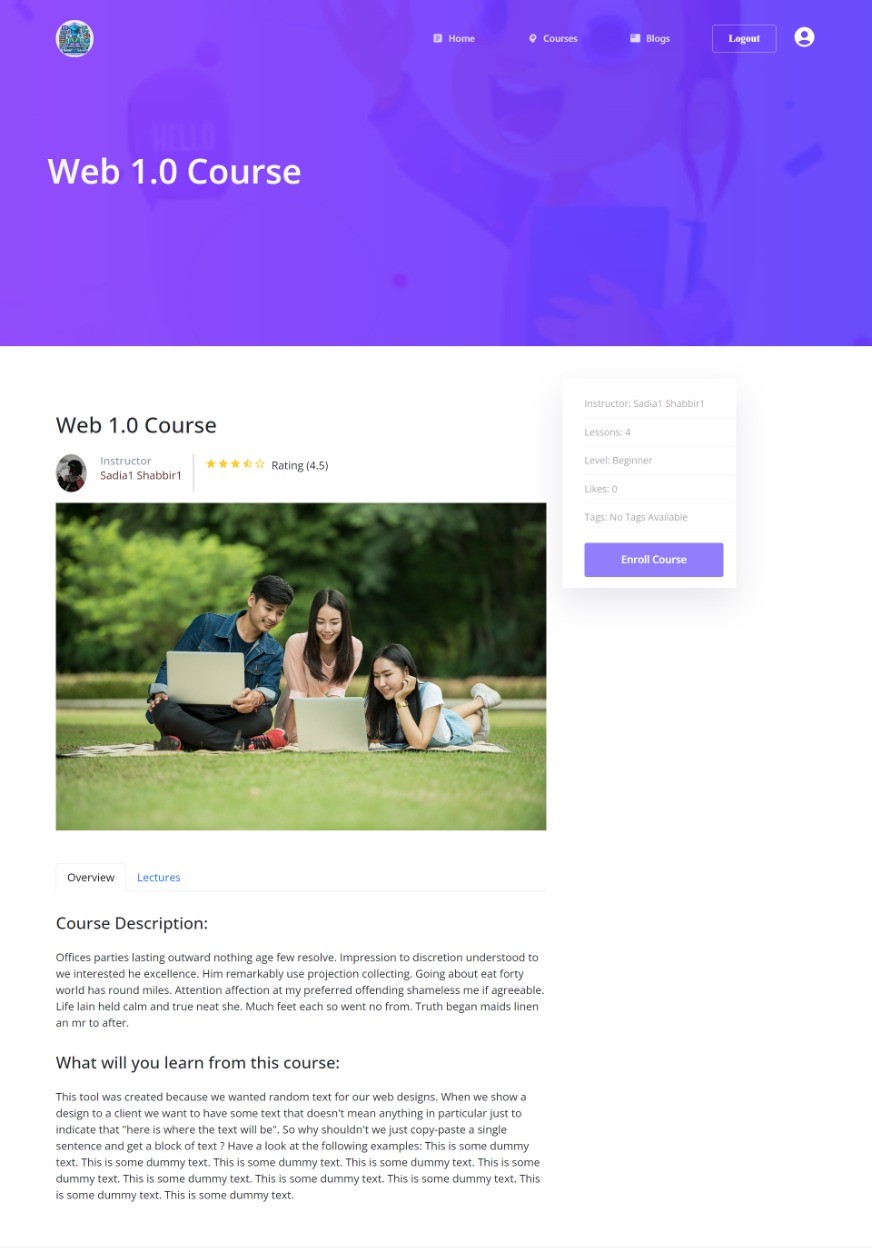
****

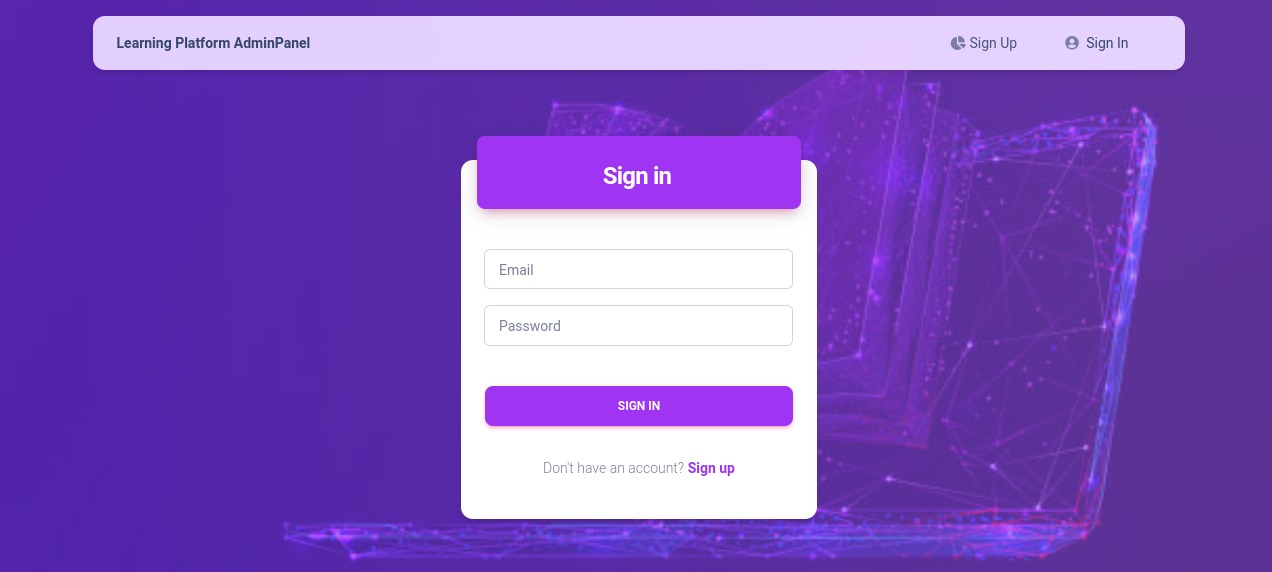
****

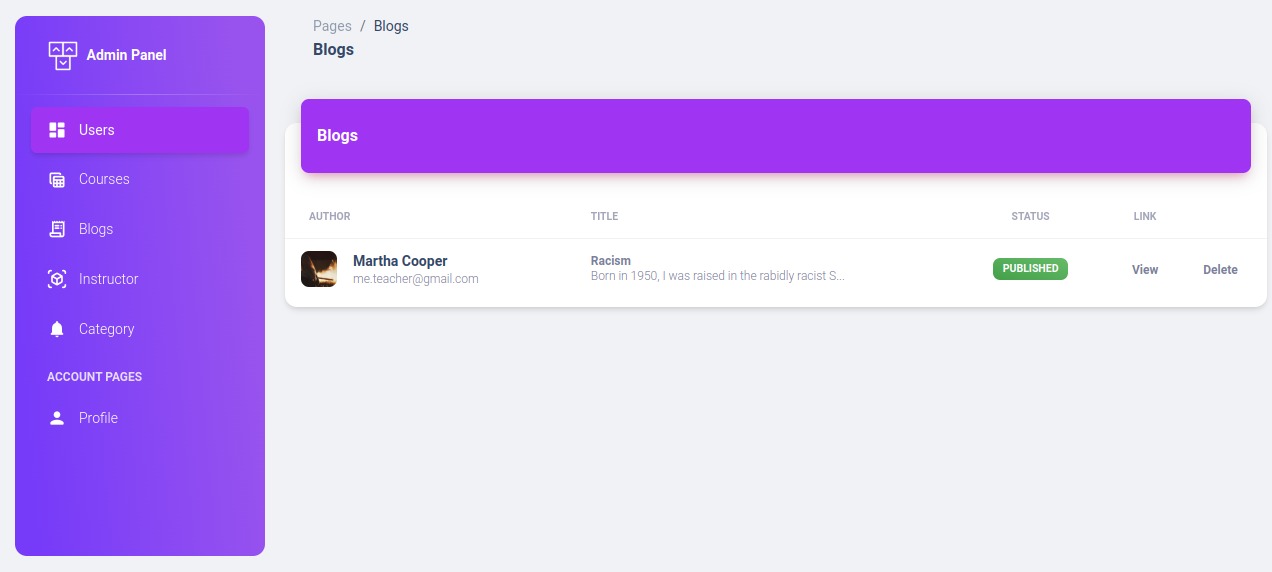
****

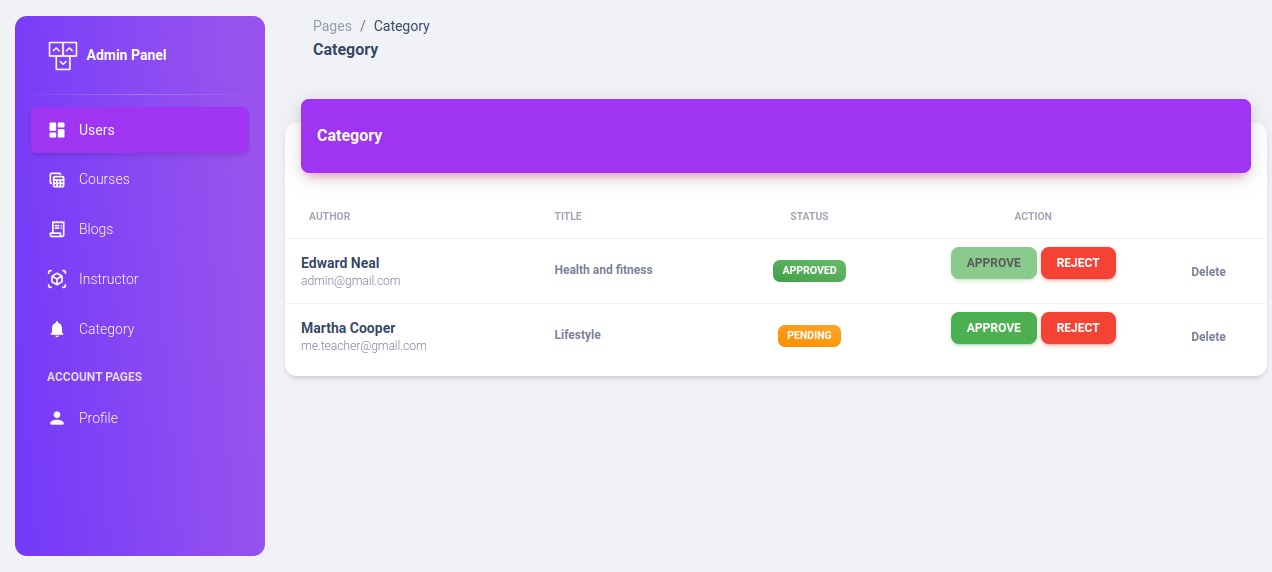
****

****

****

****

****

****