Project Title: E-Learning Platform Development with Django and Docker

Student: Issayev Abzal

Date: 20.12.2024

Institution/Organization Name: KBTU

GitHub: https://github.com/Karisbala/WebDev2024/tree/main/FinalProject

Executive Summary

The goal of this project was to create a comprehensive e-learning platform that enables user enrollment, course browsing, lesson access, quiz attempts, and review submissions. By employing Django for backend logic and Docker for containerization, a maintainable and portable system was developed. The main achievements included successfully implementing essential models, such as Users, Courses, Enrollments, Lessons, Quizzes, and Reviews, and integrating these components with a frontend that communicates through a RESTful API. Notable outcomes involved enabling instructors to create and manage courses and quizzes, and allowing students to enroll in courses, complete lessons, and attempt quizzes. This led to a stable and functional platform that can be easily deployed and scaled. The recommendation is to consider continuous integration practices and further frontend enhancements in the future.

Table of Contents

Introduction

Docker’s containerization was utilized to isolate the e-learning application and its dependencies [1]. Django was employed to handle the backend logic and serve the content [2]. The project’s focus was on creating a dynamic e-learning environment with proper data management.
The goal was to develop a platform that allows users to register, enroll in courses, view lessons, attempt quizzes, and leave reviews. Instructors were given tools to create courses, add lessons, and manage quizzes. The solution was required to be containerized for smooth deployment.
The project covered backend and frontend integration, data modeling, RESTful API development [3], and Docker-based containerization. Limitations included the use of a mock dataset and a focus on essential features like user enrollment, course management, and quiz attempts.

System Architecture

The system architecture was designed to run multiple services as separate Docker containers. Each container performed a specific function. The Django application container served the backend logic. The PostgreSQL container stored the database [4]. The frontend application container interacted with the Django backend by sending HTTP requests. Docker Compose was used to define and start all containers together. Each container communicated through a Docker network. The application stack was organized so that the frontend could send requests to the Django API and the Django API could query the PostgreSQL database. This approach simplified scaling and maintenance. The containers could be replaced or updated without changing the overall system structure. The use of separate containers also improved isolation and security.

Code Snippet of docker compose configuration that illustrates service definitions for web and db:

```yaml
services:
  web:
    build: ./backend
    container_name: elearning_web
    command: gunicorn backend.wsgi:application --bind 0.0.0.0:8000
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DEBUG=1
      - DB_NAME=elearning_db
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - DB_HOST=db
      - DB_PORT=5432

  db:
    image: postgres:14
    container_name: elearning_db
    environment:
      - POSTGRES_DB=elearning_db
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

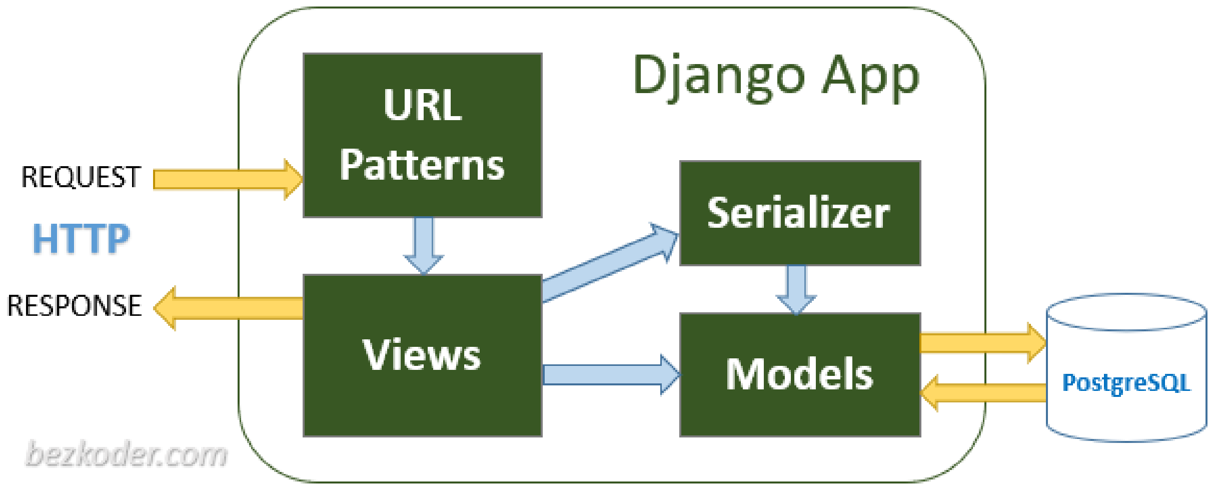


Figure 1. Diagram displaying app’s basic architecture.

Table Descriptions

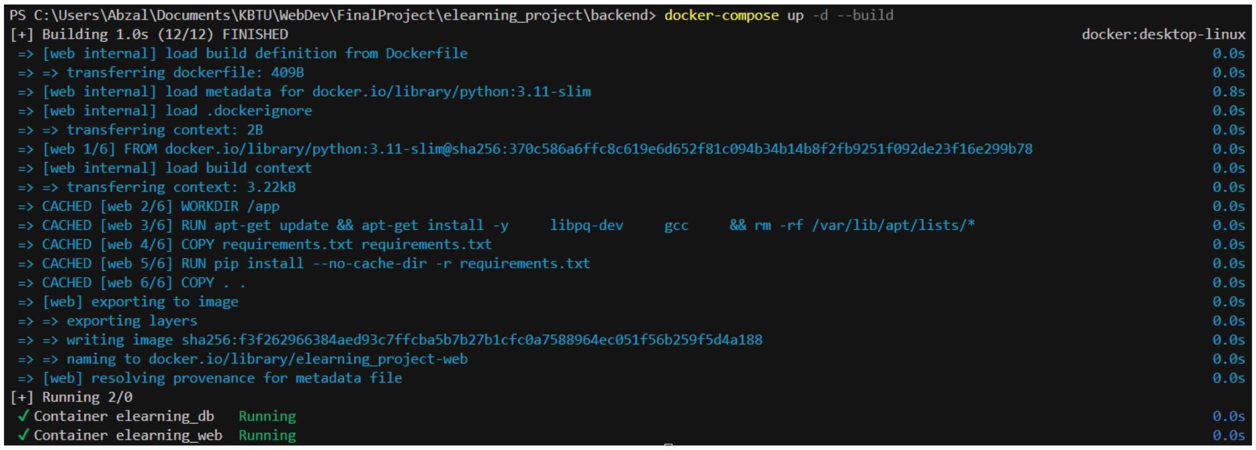
The Users table stored user authentication details and roles. The Courses table stored course metadata such as title description price and instructor reference. The Enrollments table tracked which users were enrolled in which courses. The Lessons table stored lesson content and video links. The Reviews table recorded user feedback on courses. The Categories table organized courses into groups. The Payments table stored payment transactions made by users. The Quizzes table represented course related quizzes and the QuizQuestions table contained questions and correct answers. The UserProgress table stored user progress data including completed lessons and quiz scores. Each table related to others through foreign keys ensuring data integrity.

Code Snippet of a Django model definition for the Course model with foreign keys to instructor and category:

```
class Course(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL,
null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    instructor = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='created_courses'
    )

    def __str__(self):
        return self.title
```

Intro to Containerization: Docker

Containerization was introduced to ensure that the application and its dependencies worked uniformly across all environments. Docker images were built to package the code and libraries together. Docker used lightweight virtualization which allowed faster startup times than virtual machines. The Docker engine ran containers that shared the host OS kernel. This reduced overhead. Docker provided a way to run the same image on any host that had Docker installed. This approach improved portability and reproducibility. By using Docker developers did not need to install all dependencies on their host machines. Instead they started the Docker container and everything ran inside it. This approach simplified setup and teardown steps.



```
PS C:\Users\Abzal\Documents\KBTU\WebDev\FinalProject\elearning_project\backend> docker-compose up -d --build
[+] Building 1.0s (12/12) FINISHED                                                                                                 docker:desktop-linux
 => [web internal] load build definition from Dockerfile                                                                                           0.0s
 => => transferring dockerfile: 409B                                                                                                               0.0s
 => [web internal] load metadata for docker.io/library/python:3.11-slim                                                                            0.8s
 => [web internal] load .dockerignore                                                                                                              0.0s
 => => transferring context: 2B                                                                                                                    0.0s
 => [web 1/6] FROM docker.io/library/python:3.11-slim@sha256:370c586a6ffc8c619e6d652f81c094b34b14b8f2fb9251f092de23f16e299b78                      0.0s
 => [web internal] load build context                                                                                                              0.0s
 => => transferring context: 3.22kB                                                                                                                0.0s
 => CACHED [web 2/6] WORKDIR /app                                                                                                                  0.0s
 => CACHED [web 3/6] RUN apt-get update && apt-get install -y     libpq-dev     gcc     && rm -rf /var/lib/apt/lists/*                             0.0s
 => CACHED [web 4/6] COPY requirements.txt requirements.txt                                                                                        0.0s
 => CACHED [web 5/6] RUN pip install --no-cache-dir -r requirements.txt                                                                            0.0s
 => CACHED [web 6/6] COPY . .                                                                                                                      0.0s
 => [web] exporting to image                                                                                                                       0.0s
 => => exporting layers                                                                                                                            0.0s
 => => writing image sha256:f3f262966384aed93c7ffcba5b7b27b1cfc0a7588964ec051f56b259f5d4a188                                                       0.0s
 => => naming to docker.io/library/elearning_project-web                                                                                           0.0s
 => [web] resolving provenance for metadata file                                                                                                   0.0s
[+] Running 2/0
 ✓ Container elearning_db   Running                                                                                                                0.0s
 ✓ Container elearning_web  Running                                                                                                                0.0s
```

Figure 2. Screenshot of the terminal showcasing the result of build command.

Dockerfile

A Dockerfile defined how the image for the Django application was built. It started from a base image such as python or python slim. Then it installed system packages and Python dependencies like Django and DRF. The Dockerfile set the working directory and copied the project files into the image. Then it set environment variables and exposed the application port. Finally it specified the command to run the Django application possibly through a production server like gunicorn. This ensured that anyone building the image got the same environment and dependencies. The Dockerfile acted like a recipe for creating the application image.

Code Snippet of the Dockerfile with FROM RUN and CMD instructions

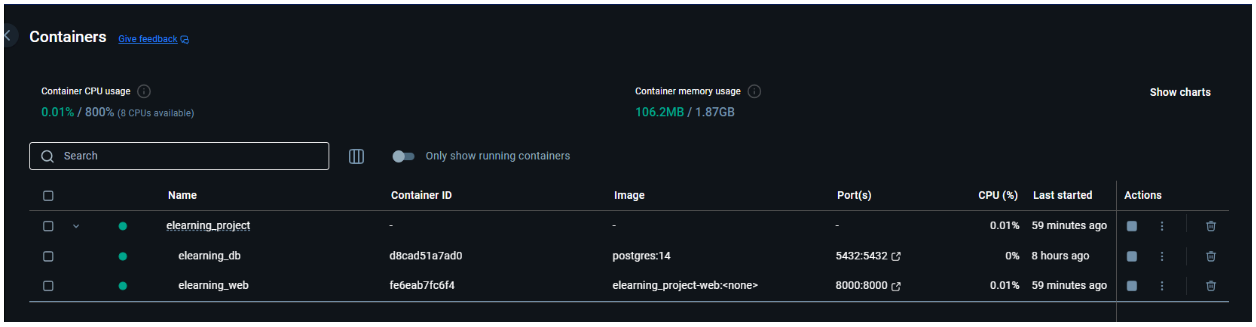
```dockerfile
FROM python:3.11-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
    libpq-dev \
    gcc \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

ENV PYTHONUNBUFFERED=1

EXPOSE 8000

CMD ["gunicorn", "backend.wsgi:application", "--bind", "0.0.0.0:8000"]
```

Docker-Compose

Docker Compose was used to manage multiple containers together [5]. A single docker compose file was created to define services for the web application container and the database container. Docker Compose allowed these containers to be started stopped and rebuilt with a single command. The configuration included service definitions environment variables and volume mounts. By using Docker Compose it became easier to maintain consistent development and production environments. Each service was described in YAML format making the setup clear and understandable. No manual linking of containers was required because Docker Compose handled networking automatically. Scaling services could be done by specifying the number of container instances. This approach simplified the orchestration of all parts of the application stack.

Code Snippet of docker compose configuration that illustrates service definitions for web and db:

```
services:
  web:
    build: ./backend
    container_name: elearning_web
    command: gunicorn backend.wsgi:application --bind 0.0.0.0:8000
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DEBUG=1
      - DB_NAME=elearning_db
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - DB_HOST=db
      - DB_PORT=5432

  db:
    image: postgres:14
    container_name: elearning_db
    environment:
      - POSTGRES_DB=elearning_db
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```



Containers Give feedback

Container CPU usage
0.01% / 800% (8 CPUs available)

Container memory usage
106.2MB / 1.87GB

Show charts

Search

Only show running containers

Name	Container ID	Image	Port(s)	CPU (%)	Last started
elearning_project	-	-	-	0.01%	59 minutes ago
elearning_db	d8cad51a7ad0	postgres:14	5432:5432	0%	8 hours ago
elearning_web	fe6eab7fc6f4	elearning_project-web:<none>	8000:8000	0.01%	59 minutes ago

Figure 3. Screenshot from Docker Desktop showcasing created containers.

Docker Networking and Volumes

Docker networking features provided a virtual network that connected all containers [6]. The containers communicated by using service names instead of IP addresses. This eliminated the need to hardcode IPs. Docker volumes were used for data persistence. For example the PostgreSQL container used a volume to store database files outside the container filesystem. This ensured that data remained safe even if the container was removed. Volumes were declared in docker compose so they were created automatically. No complex manual steps were needed. The combination of Docker networking and volumes enabled a stable environment where the database retained its state and all containers worked together smoothly.

Code Snippet of docker compose configuration that illustrates declaration of networking and volumes:

```
services:
  web:
    build: ./backend
    container_name: elearning_web
    command: gunicorn backend.wsgi:application --bind 0.0.0.0:8000
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DEBUG=1
      - DB_NAME=elearning_db
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - DB_HOST=db
      - DB_PORT=5432

  db:
    image: postgres:14
    container_name: elearning_db
    environment:
      - POSTGRES_DB=elearning_db
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Django

Django was the main backend framework for the e learning platform. Django offered an ORM for database queries built in authentication and a simple structure that separated code into models views and templates. A new Django project was created and connected to the PostgreSQL database defined in docker compose. The settings file was updated to use environment variables so the database credentials and other configurations could be changed easily. Django migrations were used to create tables and apply schema changes. The admin interface was enabled to allow quick database inspection and content management. This framework made it easier to implement required features like handling user enrollments and displaying course information.
Code Snippets that show a snippet of settings.py where DATABASES dictionary references environment variables:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'elearning_db'),
        'USER': os.environ.get('DB_USER', 'postgres'),
        'PASSWORD': os.environ.get('DB_PASSWORD', 'postgres'),
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
}
```

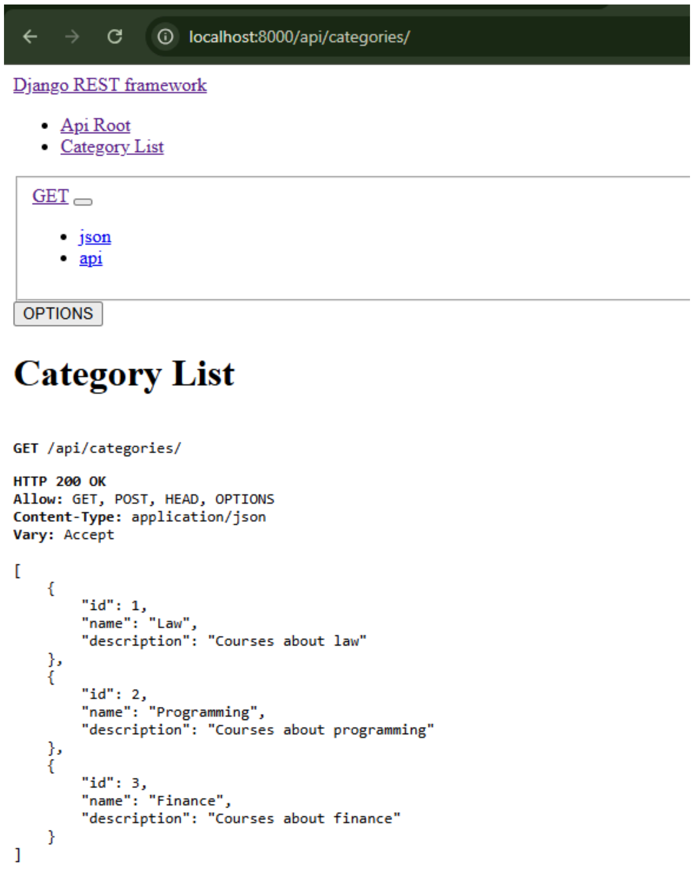
```
DEBUG=1
DB_NAME=elearning_db
DB_USER=postgres
DB_PASSWORD=postgres
DB_HOST=db
DB_PORT=5432
SECRET_KEY='django-insecure--ropgi8a%04nnr3!d5c064@v&sbv822dsv+0#oo-4jpm-!
^k&l'
```

Models

Django models represented the data structures of the platform [7]. Each model matched a table defined in the schema. The User model stored user credentials and roles. The Course model stored basic course data and a foreign key to the instructor. The Enrollment model tracked user course relationships and their statuses. The Lesson model stored lesson details including title content and optional video link. The Review model recorded user feedback with ratings and comments. The Category model classified courses. The Payment model recorded financial transactions. The Quiz model and QuizQuestion model represented quizzes and their questions. The UserProgress model stored user progress such as completed lessons and quiz scores. All foreign keys and many to many fields were defined so that relationships were clear. Migrations were run to create these tables in the database.

Code Snippet that shows a Django model class for the Category model with fields like name and description:

```
class Category(models.Model):
    name = models.CharField(max_length=100, unique=True)
    description = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.name
```



localhost:8000/api/categories/

Django REST framework

- Api Root
- Category List

GET

- json
- api

OPTIONS

Category List

```
GET /api/categories/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "name": "Law",
        "description": "Courses about law"
    },
    {
        "id": 2,
        "name": "Programming",
        "description": "Courses about programming"
    },
    {
        "id": 3,
        "name": "Finance",
        "description": "Courses about finance"
    }
]
```

Figure 4. Screenshot of the api page of categories.

Views

Views in Django handled the logic for each request [8]. Different types of views were implemented to return the needed data. Some views returned HTML templates and others returned JSON responses. Class based views were used for complex operations because they allowed code reuse and clear structure. The Django Rest Framework introduced ViewSets which made it easy to handle standard CRUD operations for models like Courses Lessons and Quizzes. The frontend sent requests to these views to get courses enroll a user or attempt a quiz. The views extracted data from the database through the models and then serialized it before sending the response. This approach ensured that the logic remained simple and maintainable.

Code Snippet that shows a snippet of a DRF ViewSet handling Course model CRUD operations:

```
class CourseViewSet(viewsets.ModelViewSet):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def get_queryset(self):
        queryset = super().get_queryset()
        category_id = self.request.query_params.get('category')
        if category_id:
            queryset = queryset.filter(category_id=category_id)
        return queryset

    def perform_create(self, serializer):
        user = self.request.user
        if not user.is_authenticated or not user.is_instructor:
            return Response({'detail': 'Only instructors can create 
courses.'}, status=status.HTTP_403_FORBIDDEN)
        serializer.save(instructor=user)
```

Templates

Templates were used for rendering HTML pages when a server side rendered output was required [9]. Although much of the data was served via API endpoints some pages were still provided as templates. Django template language allowed including dynamic content like a list of courses or user specific details. The templates were stored in a templates directory and named according to the view that rendered them. Basic styling was applied through simple CSS files. The combination of views and templates allowed the platform to present courses and lessons to users in a user friendly manner. Even though much logic occurred in the API the templates provided a fallback option for pages where server side rendering was beneficial. However it was replaced completely with ReactJS.

Code Snippet shows a simple Django template file displaying a list of courses in an unordered list:

```
<!DOCTYPE html>
<html>
<head>
    <title>Available Courses</title>
</head>
<body>
    <h1>Available Courses</h1>
    <ul>
        {% for course in courses %}
        <li>
            <h2>{{ course.title }}</h2>
            <p>{{ course.description }}</p>
            <p><strong>Price:</strong> {{ course.price }}</p>
        </li>
        {% empty %}
        <li>No courses available.</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Django Rest Framework (DRF)

Django Rest Framework introduced a set of tools to build a robust API [10]. Serializers were used to convert Django model instances into JSON data. ViewSets handled repeated tasks like listing creating updating and deleting records. Permissions and authentication were supported out of the box. Using DRF ensured that the platform could easily serve data to the frontend or any other client. The code stayed clean and maintainable because DRF took care of many common tasks. The API endpoints produced JSON responses suitable for the frontend to parse and display. This arrangement simplified the communication between the frontend and backend.

Code Snippet that shows a snippet of a serializer class such as CourseSerializer referencing model fields:

```
class CourseSerializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = ['id', 'title', 'description', 'price', 'category',
'created_at', 'instructor']
        read_only_fields = ['created_at', 'instructor']
```

Code Snippet that shows a snippet of a url file of courses app:

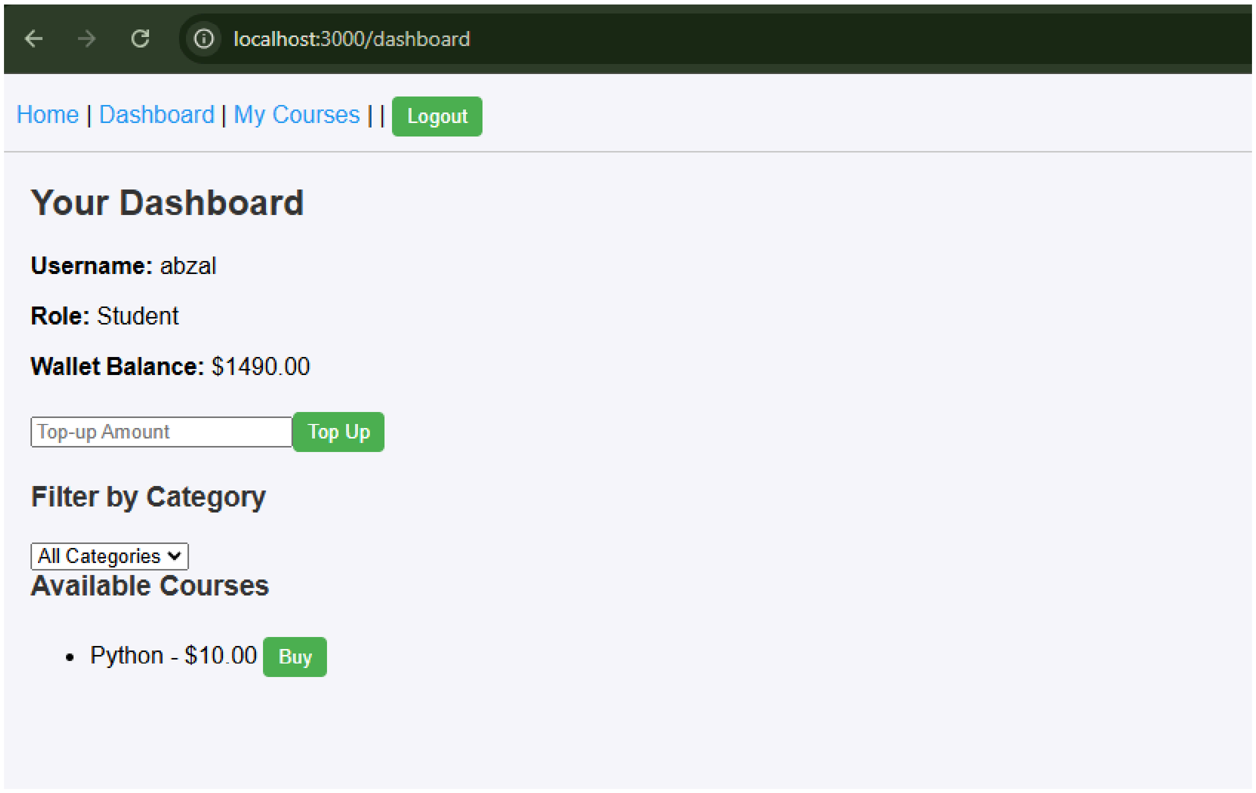
```
from rest_framework.routers import DefaultRouter
from .views import CourseViewSet, LessonViewSet

router = DefaultRouter()
router.register(r'courses', CourseViewSet, basename='course')
router.register(r'lessons', LessonViewSet, basename='lesson')

urlpatterns = router.urls
```

Frontend Integration

A frontend framework communicated with the Django REST API endpoints. The frontend sent GET requests to fetch course lists or lesson details. It sent POST requests to enroll users in courses or to submit quiz attempts. Each response was parsed and displayed in a user friendly format. The frontend used library React to handle state and update the screen dynamically [11]. When a user clicked a button to purchase a course or mark a lesson as completed the frontend called the appropriate API endpoint and updated the UI based on the response. This integration allowed a seamless experience for both students and instructors.

Code Snippet that shows a snippet of a fetch call in the frontend to retrieve courses data from the API:

```
const fetchCourses = async (categoryId = '') => {
    try {
      let url = 'courses/';
      if (categoryId) {
        url += `?category=${categoryId}`;
      }
      const response = await API.get(url);
      setCourses(response.data);
    } catch (err) {
      console.error('Failed to fetch courses', err);
    }
  };
```


localhost:3000/dashboard

Home | Dashboard | My Courses | | Logout

Your Dashboard

Username: abzal

Role: Student

Wallet Balance: $1490.00

Top-up Amount Top Up

Filter by Category

All Categories

Available Courses

- Python - $10.00 Buy

Figure 5. Screenshot of a Dashboard with a list of courses (only 1 course “Python”).

Challenges and Solutions

Several challenges were encountered during development. One key challenge involved making the quiz attempt endpoint work correctly. Initially a 405 error was returned when trying to submit quiz answers. This issue was solved by adjusting the URL configuration and ensuring that the endpoint path matched exactly what the frontend called. Another challenge was enabling instructors to create multiple lessons easily. This was solved by adding a form to the manage course page where instructors could enter lesson details and submit them. Each submission created a new lesson. Another challenge involved ensuring that all containers started and communicated properly. Running docker compose commands and adjusting environment variables solved this. The final code and configuration allowed smooth interactions between the frontend and the backend.

Code Snippet that shows the quiz attempt URL configuration in quizzes urls.py:

```
from django.urls import path
from rest_framework.routers import DefaultRouter
from .views import QuizViewSet, QuizQuestionViewSet, QuizAttemptView

router = DefaultRouter()
router.register(r'quizzes', QuizViewSet, basename='quiz')
router.register(r'quiz-questions', QuizQuestionViewSet,
basename='quiz-question')

urlpatterns = [
    path('quizzes/attempt/', QuizAttemptView.as_view(),
name='quiz_attempt'),
] + router.urls
```

Conclusion

In conclusion the e learning platform was built using Django for backend logic and Docker for containerization. The project goals were met. Users could register browse courses enroll in them view lessons attempt quizzes and leave reviews. Instructors could create and manage courses add lessons and organize quizzes. Docker helped ensure that the application remained portable and consistent across different environments. Django Rest Framework provided a clean way to expose and consume APIs. Although the UI remained basic it still offered core functionality. This foundation could be extended later with more advanced features and better styling. The completed platform demonstrated the usefulness of these tools and technologies in building a complex web application efficiently.

References

1. Docker Inc. (n.d.). *What is a container?* Retrieved from https://www.docker.com/resources/what-container/
2. Django Software Foundation. (n.d.). *Django documentation (version 5.1).* Retrieved from https://docs.djangoproject.com/en/5.1/
3. GeeksforGeeks. (n.d.). *REST API introduction.* Retrieved from https://www.geeksforgeeks.org/rest-api-introduction/
4. PostgreSQL Global Development Group. (n.d.). *PostgreSQL.* Retrieved from https://www.postgresql.org/
5. Docker Inc. (n.d.). *Docker Compose.* Retrieved from https://docs.docker.com/compose/
6. Docker Inc. (n.d.). *Docker Engine network.* Retrieved from https://docs.docker.com/engine/network/
7. Django Software Foundation. (n.d.). *Django documentation (version 5.1): Database models.* Retrieved from https://docs.djangoproject.com/en/5.1/topics/db/models/
8. Django Software Foundation. (n.d.). *Django documentation (version 5.1): HTTP views.* Retrieved from https://docs.djangoproject.com/en/5.1/topics/http/views/
9. Django Software Foundation. (n.d.). *Django documentation (version 5.1): Templates.* Retrieved from https://docs.djangoproject.com/en/5.1/topics/templates/
10. Encode OSS Ltd. (n.d.). *Django REST framework.* Retrieved from https://www.django-rest-framework.org/
11. Meta Platforms, Inc. (n.d.). *React documentation.* Retrieved from https://react.dev/

Appendices

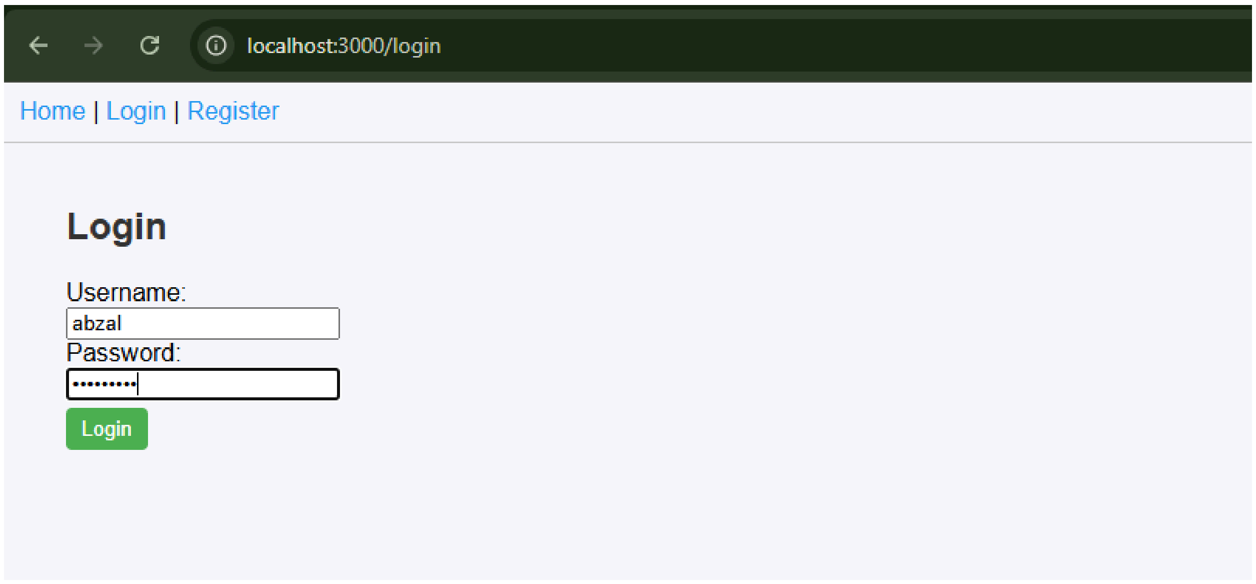


Figure 6. Login page.

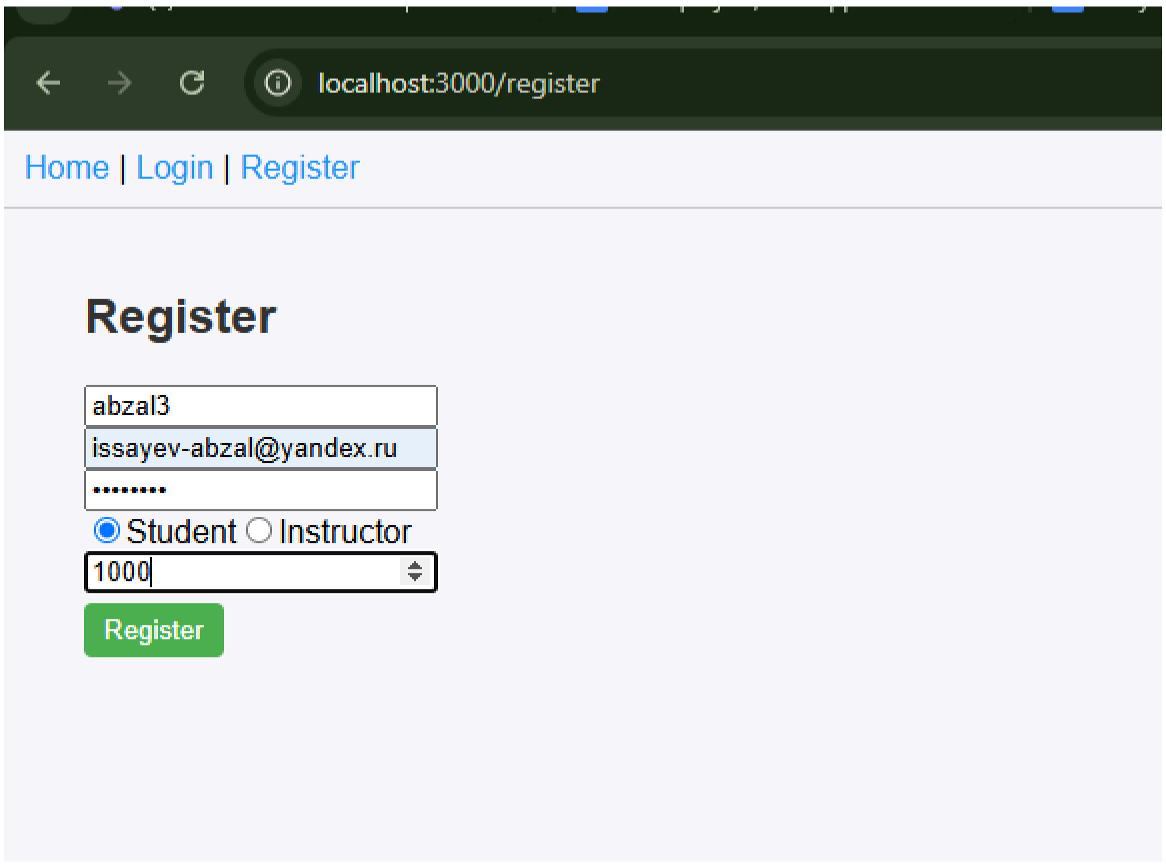
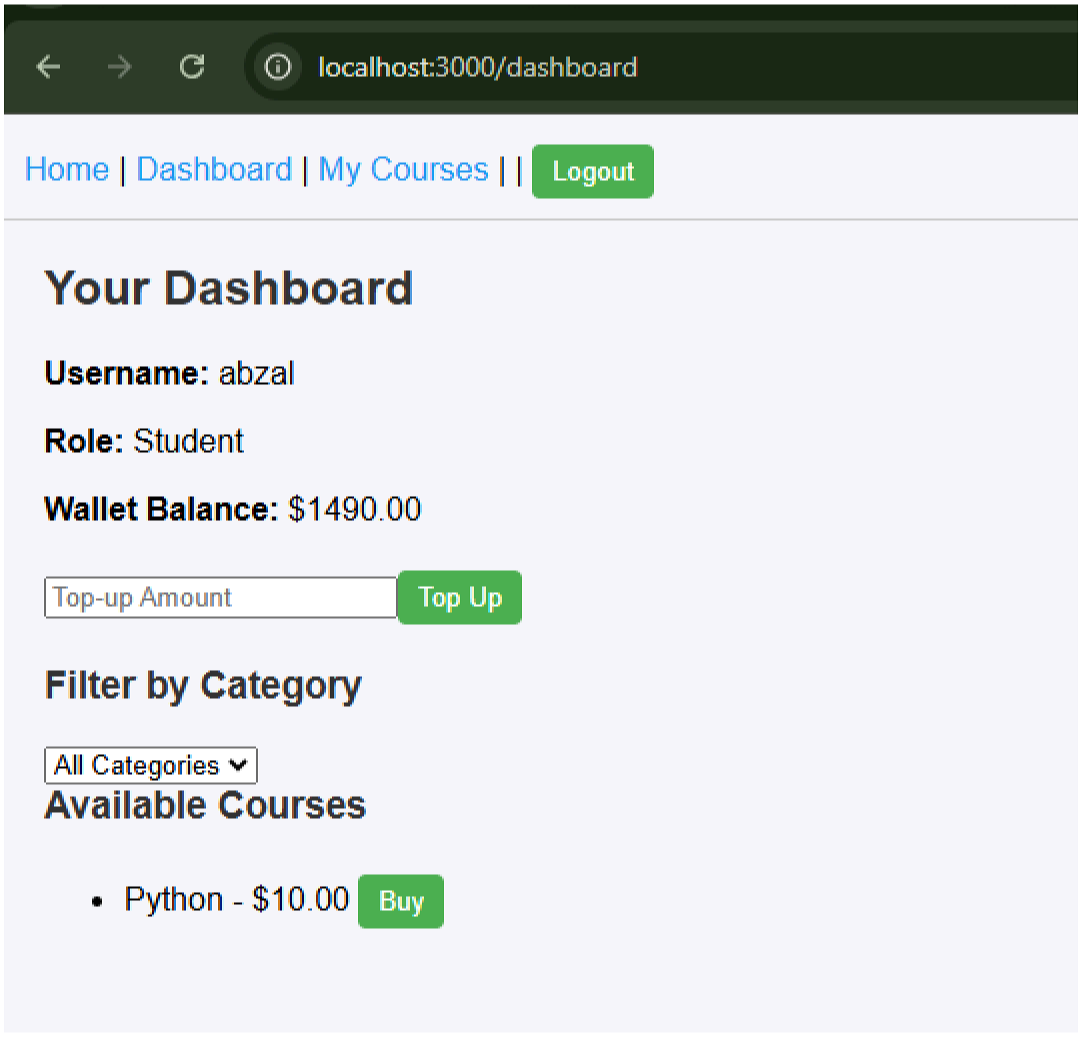


Figure 7. Register page.


localhost:3000/dashboard

Home | Dashboard | My Courses | | Logout

Your Dashboard

Username: abzal

Role: Student

Wallet Balance: $1490.00

Top-up Amount Top Up

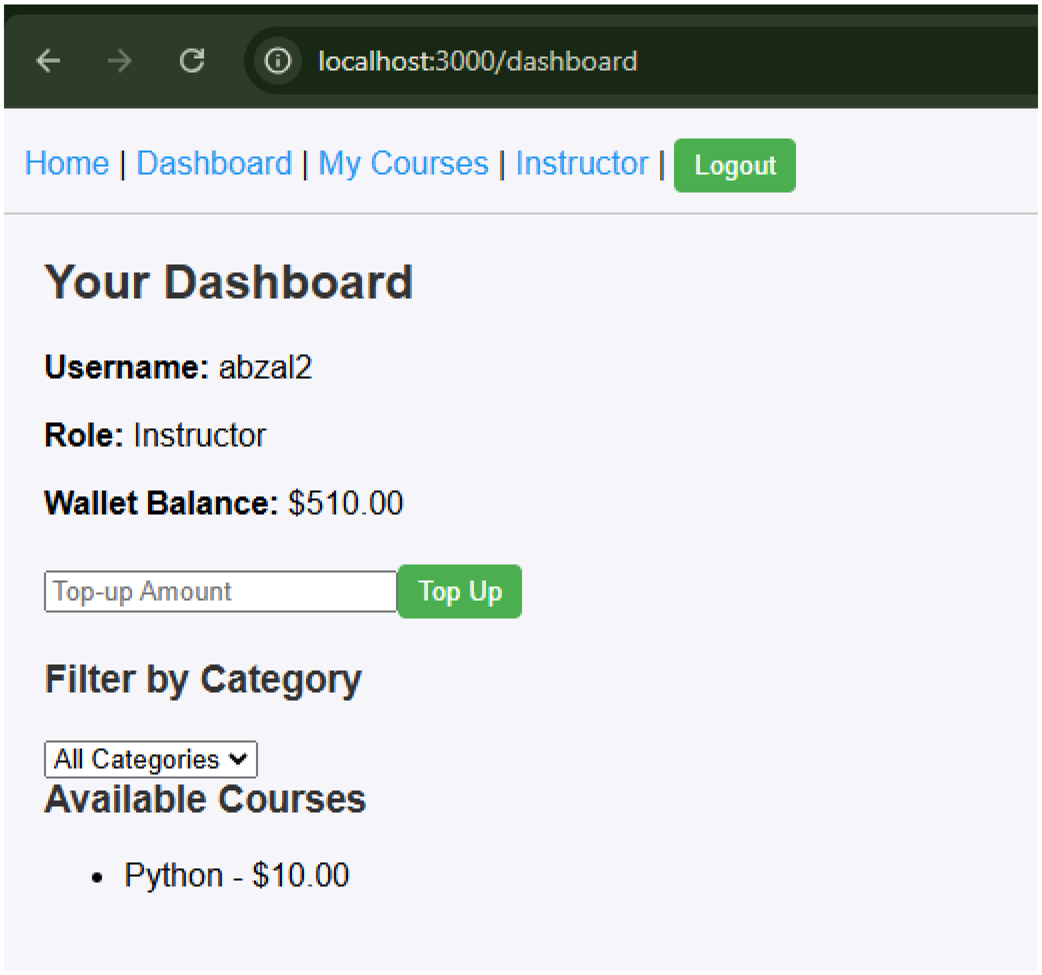
Filter by Category

All Categories

Available Courses

- Python - $10.00 Buy

Figure 8. Dashboard page of student user.


localhost:3000/dashboard

Home | Dashboard | My Courses | Instructor | Logout

Your Dashboard

Username: abzal2

Role: Instructor

Wallet Balance: $510.00

Top-up Amount | Top Up

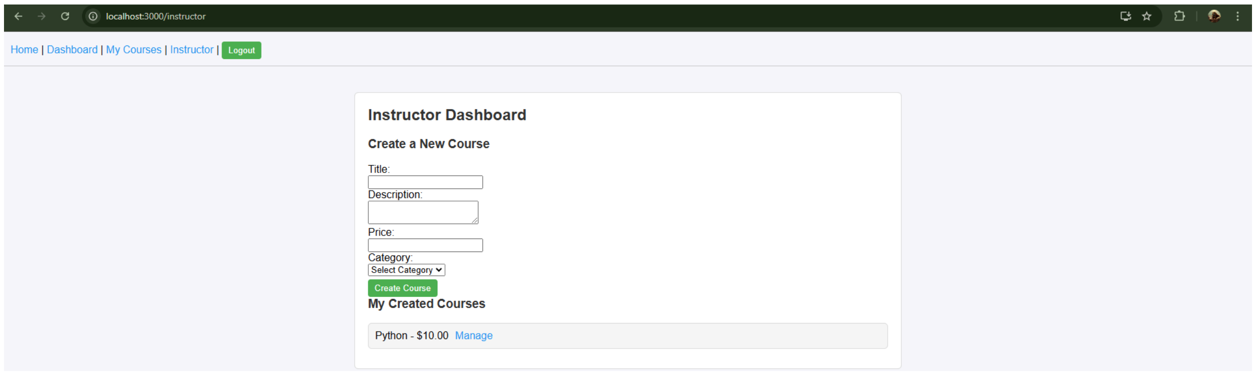
Filter by Category

All Categories

Available Courses

- Python - $10.00

Figure 9. Dashboard page of instructor user.


localhost:3000/instructor

Home | Dashboard | My Courses | Instructor | Logout

Instructor Dashboard

Create a New Course

Title:

Description:

Price:

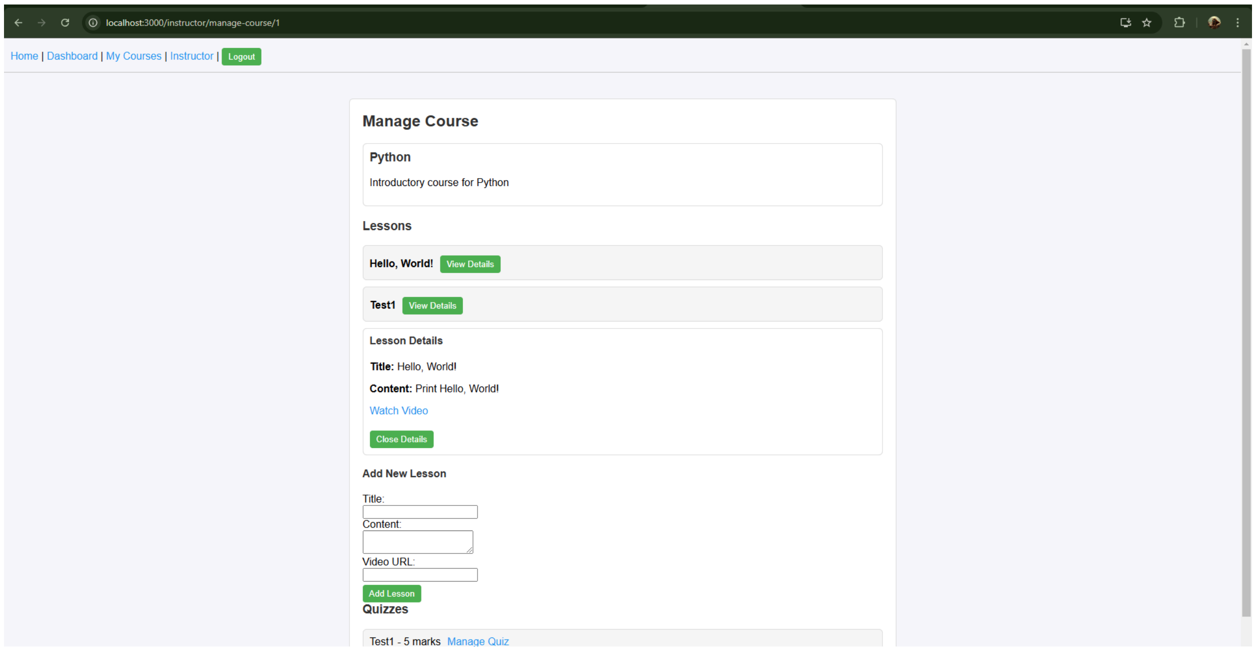
Category:
Select Category

Create Course

My Created Courses

Python - $10.00 Manage

Figure 10. Instructor page.


localhost:3000/instructor/manage-course/1
Home | Dashboard | My Courses | Instructor | Logout

Manage Course

Python

Introductory course for Python

Lessons

Hello, World! View Details

Test1 View Details

Lesson Details

Title: Hello, World!

Content: Print Hello, World!

Watch Video

Close Details

Add New Lesson

Title:

Content:

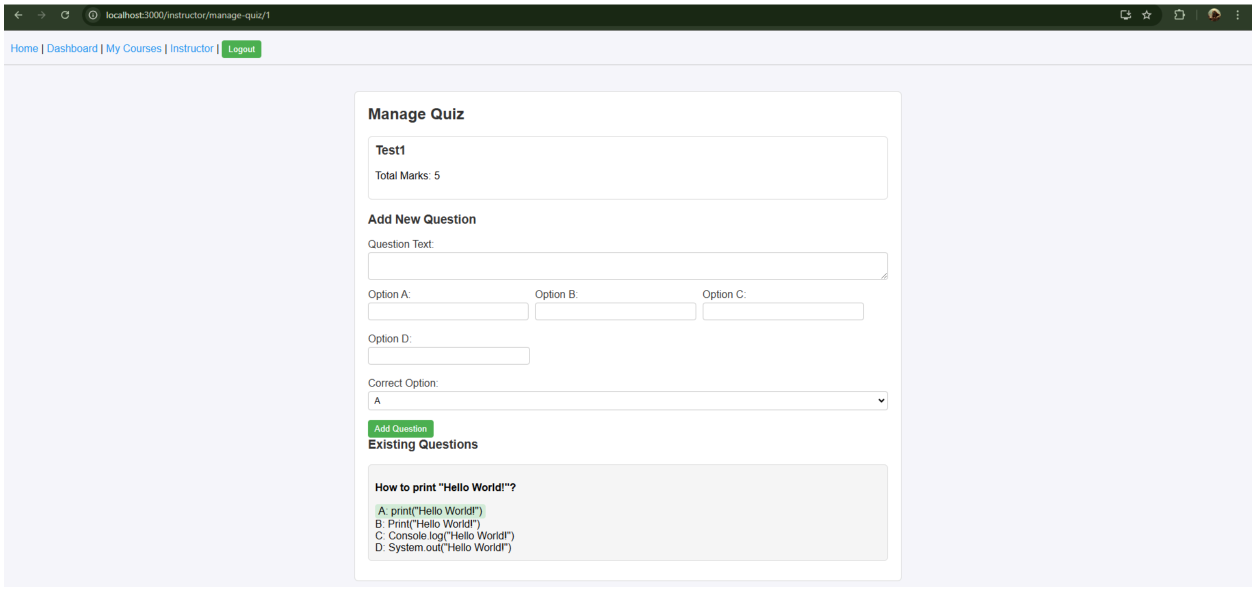
Video URL:

Add Lesson

Quizzes

Test1 - 5 marks Manage Quiz

Figure 11. Manage course page.


localhost:3000/instructor/manage-quiz/1
Home | Dashboard | My Courses | Instructor | Logout

Manage Quiz

Test1

Total Marks: 5

Add New Question

Question Text:

Option A: Option B: Option C:

Option D:

Correct Option:

A

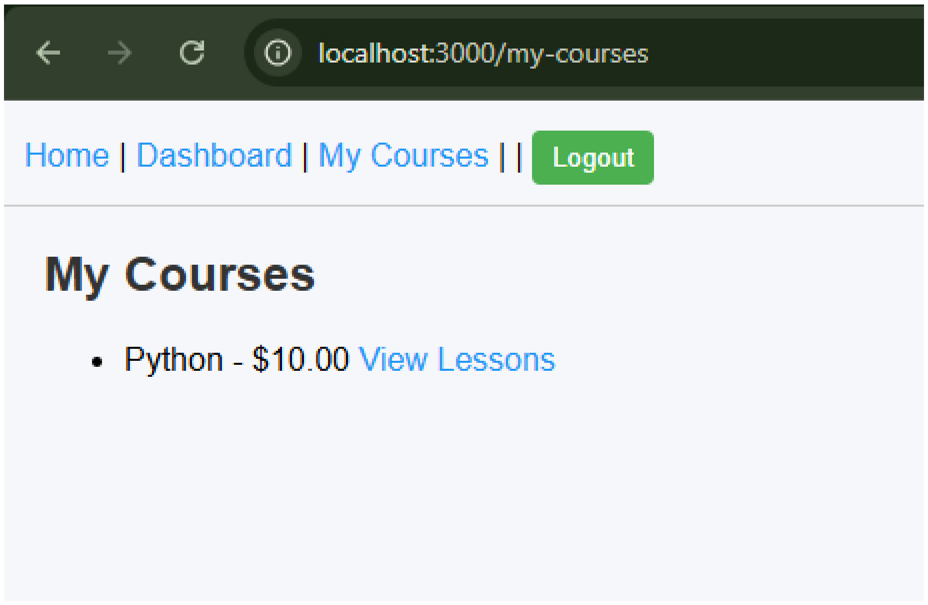
Add Question

Existing Questions

How to print "Hello World!"?

A: print("Hello World!")
B: Print("Hello World!")
C: Console.log("Hello World!")
D: System.out("Hello World!")

Figure 12. Manage quizz page.



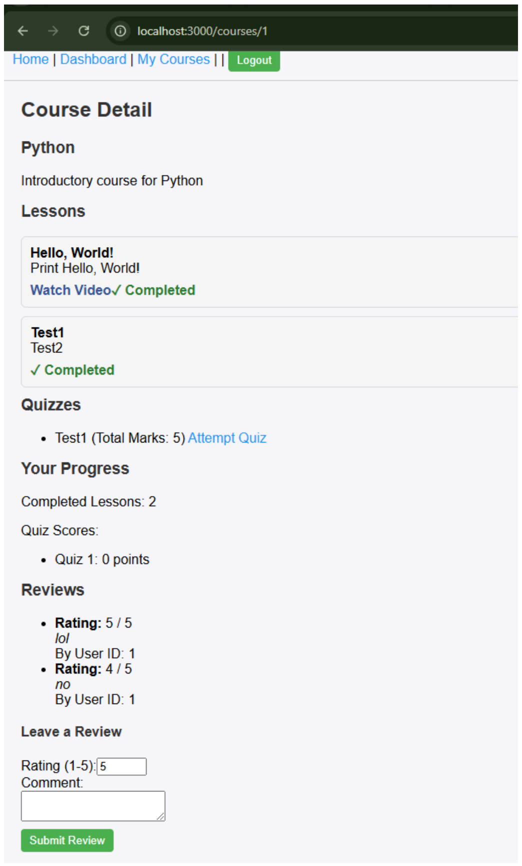
localhost:3000/my-courses

Home | Dashboard | My Courses | | Logout

My Courses

- Python - $10.00 View Lessons

Figure 13. My courses page.



localhost:3000/courses/1

Home | Dashboard | My Courses | | Logout

Course Detail

Python

Introductory course for Python

Lessons

Hello, World!
Print Hello, World!
Watch Video✓ Completed

Test1
Test2
✓ Completed

Quizzes

- Test1 (Total Marks: 5) Attempt Quiz

Your Progress

Completed Lessons: 2

Quiz Scores:

- Quiz 1: 0 points

Reviews

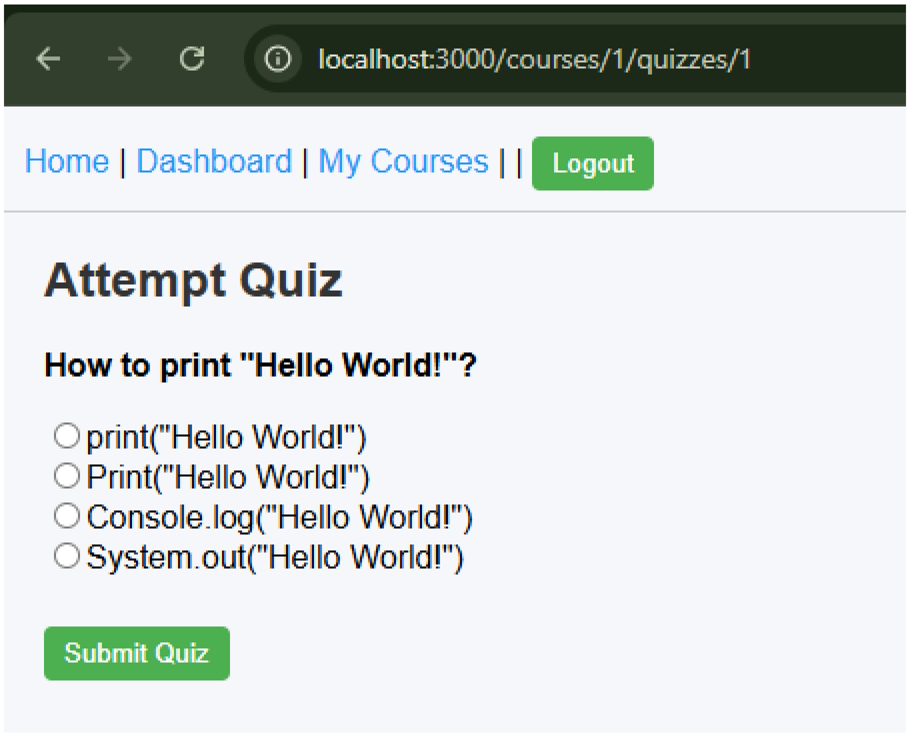
- Rating: 5 / 5
  lol
  By User ID: 1
- Rating: 4 / 5
  no
  By User ID: 1

Leave a Review

Rating (1-5): 5
Comment:

Submit Review

Figure 14. Course Detail page.

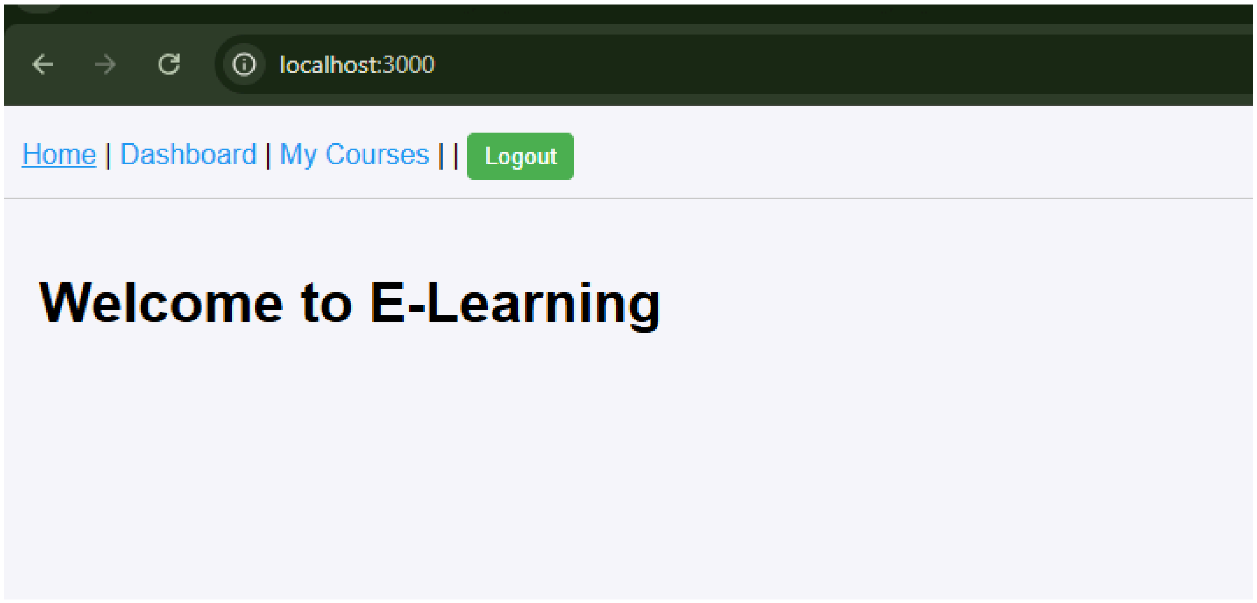

localhost:3000/courses/1/quizzes/1
Home | Dashboard | My Courses | |
Logout

Attempt Quiz

How to print "Hello World!"?

- ◯ print("Hello World!")
- ◯ Print("Hello World!")
- ◯ Console.log("Hello World!")
- ◯ System.out("Hello World!")

Figure 15. Attempt Quiz page.



Welcome to E-Learning

Figure 16. Home page.

Code Snippets from account app:

```python
from django.contrib.auth.models import AbstractUser
from django.db import models

class User(AbstractUser):
    # AbstractUser already has: username, email, password, first_name,
last_name
    is_student = models.BooleanField(default=False)
    is_instructor = models.BooleanField(default=False)
    wallet = models.DecimalField(max_digits=10, decimal_places=2,
default=0.00)

    def __str__(self):
        return self.username
```

```
from rest_framework import serializers
from .models import User

class UserRegistrationSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True)
    is_student = serializers.BooleanField(required=True)
    is_instructor = serializers.BooleanField(required=True)

    class Meta:
        model = User
        fields = ['username', 'email', 'password', 'is_student',
'is_instructor', 'wallet']

    def validate(self, attrs):
        if attrs.get('is_student') and attrs.get('is_instructor'):
            raise serializers.ValidationError("User cannot be both student
and instructor.")
        return attrs

    def create(self, validated_data):
        password = validated_data.pop('password')
        user = User(**validated_data)
        user.set_password(password)
        user.save()
        return user

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'email', 'is_student',
'is_instructor', 'wallet']
```

```
from django.urls import path
from .views import RegisterView, CurrentUserView
from rest_framework_simplejwt.views import TokenObtainPairView,
TokenRefreshView

urlpatterns = [
    path('register/', RegisterView.as_view(), name='register'),
    path('token/', TokenObtainPairView.as_view(),
name='token_obtain_pair'),
    path('token/refresh/', TokenRefreshView.as_view(),
name='token_refresh'),
    path('user/', CurrentUserView.as_view(), name='current_user'),
]
```

```
from rest_framework import generics, permissions
from rest_framework.response import Response
from rest_framework.status import HTTP_201_CREATED, HTTP_400_BAD_REQUEST
from .serializers import UserRegistrationSerializer, UserSerializer
from rest_framework.views import APIView
from rest_framework.permissions import IsAuthenticated

class RegisterView(generics.GenericAPIView):
    serializer_class = UserRegistrationSerializer
    permission_classes = [permissions.AllowAny]

    def post(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        if serializer.is_valid():
            user = serializer.save()
            return Response(UserSerializer(user).data,
status=HTTP_201_CREATED)
        return Response(serializer.errors, status=HTTP_400_BAD_REQUEST)

class CurrentUserView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        serializer = UserSerializer(request.user)
        return Response(serializer.data)
```

Code Snippets from backend app:

```
"""
Django settings for backend project.

Generated by 'django-admin startproject' using Django 5.1.1.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

AUTH_USER_MODEL = 'accounts.User'

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get('SECRET_KEY', 'fallback_secret_key')
```

```python
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = os.environ.get('DEBUG', '0') == '1'

ALLOWED_HOSTS = ['*']


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework_simplejwt',
    'accounts',
    'courses',
    'categories',
    'enrollments',
    'payments',
    'reviews',
    'quizzes',
    'corsheaders',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

CORS_ALLOW_ALL_ORIGINS = True

CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",  # React frontend
]

CORS_ALLOW_CREDENTIALS = True

CORS_ALLOW_METHODS = [
    "GET",
    "POST",
    "PUT",
    "PATCH",
    "DELETE",
    "OPTIONS",
]

CORS_ALLOW_HEADERS = [
    "content-type",
    "authorization",
    "x-csrftoken",
]

ROOT_URLCONF = 'backend.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'backend.wsgi.application'

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticatedOrReadOnly',
    ]
}

# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'elearning_db'),
        'USER': os.environ.get('DB_USER', 'postgres'),
        'PASSWORD': os.environ.get('DB_PASSWORD', 'postgres'),
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
}


# Password validation
```

```
#
https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validato
rs

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator'
,
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC_URL = 'static/'
STATIC_ROOT = BASE_DIR / 'staticfiles'

# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/accounts/', include('accounts.urls')),
    path('api/', include('categories.urls')),
    path('api/', include('courses.urls')),
    path('api/payments/', include('payments.urls')),
    path('api/enrollments/', include('enrollments.urls')),
    path('api/', include('reviews.urls')),
    path('api/', include('quizzes.urls')),
]
```

Code Snippets from category app:

```
from rest_framework import serializers
from .models import Category

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = ['id', 'name', 'description']
```

```
from rest_framework.routers import DefaultRouter
from .views import CategoryViewSet

router = DefaultRouter()
router.register(r'categories', CategoryViewSet, basename='category')

urlpatterns = router.urls
```

```
from rest_framework import viewsets, permissions
from .models import Category
from .serializers import CategorySerializer

class CategoryViewSet(viewsets.ModelViewSet):
    queryset = Category.objects.all()
    serializer_class = CategorySerializer

    def get_permissions(self):
        if self.request.method in ['POST', 'PUT', 'PATCH', 'DELETE']:
            return [permissions.IsAuthenticated()]
        return [permissions.AllowAny()]
```

Code Snippets from courses app:

```
class Lesson(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='lessons')
    title = models.CharField(max_length=255)
    content = models.TextField()
    video_url = models.URLField(blank=True, null=True)

    def __str__(self):
        return f"{self.title} - {self.course.title}"
```

```
class LessonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Lesson
        fields = ['id', 'course', 'title', 'content', 'video_url']
        read_only_fields = ['course']
```

```

class LessonViewSet(viewsets.ModelViewSet):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def get_queryset(self):
        queryset = super().get_queryset()
        user = self.request.user

        if not user.is_authenticated:
            return queryset.none()

        instructor_courses = Course.objects.filter(instructor=user)

        enrolled_course_ids =
Enrollment.objects.filter(user=user).values_list('course_id', flat=True)

        allowed_courses = instructor_courses.values_list('id',
flat=True).union(enrolled_course_ids)

        return queryset.filter(course_id__in=allowed_courses)

    def perform_create(self, serializer):
        user = self.request.user
        course_id = self.request.data.get('course')

        if course_id is None:
            raise ValueError("Course ID is required to create a lesson.")

        try:
            course = Course.objects.get(id=course_id)
        except Course.DoesNotExist:
            return Response({"detail": "Course not found."},
status=status.HTTP_400_BAD_REQUEST)

        if course.instructor != user:
            return Response({"detail": "Only the course instructor can add
lessons."}, status=status.HTTP_403_FORBIDDEN)

        serializer.save(course=course)

    def update(self, request, *args, **kwargs):
        lesson = self.get_object()
        if lesson.course.instructor != request.user:
            return Response({"detail": "Not allowed."},
status=status.HTTP_403_FORBIDDEN)
        return super().update(request, *args, **kwargs)
    def destroy(self, request, *args, **kwargs):
        lesson = self.get_object()
        if lesson.course.instructor != request.user:
            return Response({"detail": "Not allowed."}, 
status=status.HTTP_403_FORBIDDEN)
        return super().destroy(request, *args, **kwargs)
```

Code Snippets from enrollments app:

```
from django.db import models
from django.conf import settings
from courses.models import Course
from django.contrib.postgres.fields import JSONField

class Enrollment(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, 
on_delete=models.CASCADE, related_name='enrollments')
    course = models.ForeignKey(Course, on_delete=models.CASCADE, 
related_name='enrollments')
    enrollment_date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=20, default='enrolled')

    def __str__(self):
        return f"{self.user.username} enrolled in {self.course.title}"

class UserProgress(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, 
on_delete=models.CASCADE, related_name='progress_records')
    course = models.ForeignKey(Course, on_delete=models.CASCADE, 
related_name='progress_records')
    completed_lessons = models.JSONField(default=list)
    quiz_scores = models.JSONField(default=dict)

    def __str__(self):
        return f"Progress of {self.user.username} in {self.course.title}"
```

```
from rest_framework import serializers

class CoursePurchaseSerializer(serializers.Serializer):
    course_id = serializers.IntegerField()

    def validate_course_id(self, value):
        if value <= 0:
            raise serializers.ValidationError("Invalid course ID.")
        return value
```

```
from django.urls import path
from .views import CoursePurchaseView, UserEnrollmentsView, 
MarkLessonCompleteView, UserProgressView

urlpatterns = [
    path('purchase/', CoursePurchaseView.as_view(), 
name='course_purchase'),
    path('user-enrollments/', UserEnrollmentsView.as_view(),
name='user_enrollments'),
    path('mark-lesson-complete/', MarkLessonCompleteView.as_view(),
name='mark_lesson_complete'),
    path('progress/', UserProgressView.as_view(), name='user_progress'),
]
```

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status, permissions
from django.shortcuts import get_object_or_404
from courses.models import Course, Lesson
from accounts.models import User
from .models import Enrollment, UserProgress
from .serializers import CoursePurchaseSerializer
from courses.serializers import CourseSerializer

class CoursePurchaseView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        serializer = CoursePurchaseSerializer(data=request.data)
        if serializer.is_valid():
            course_id = serializer.validated_data['course_id']
            user = request.user
            course = get_object_or_404(Course, id=course_id)

            if Enrollment.objects.filter(user=user,
course=course).exists():
                return Response({"detail": "Already enrolled in this
course."}, status=status.HTTP_400_BAD_REQUEST)

            if course.instructor == user:
                return Response({"detail": "Instructors cannot enroll in
their own courses."}, status=status.HTTP_400_BAD_REQUEST)

            if user.wallet < course.price:
                return Response({"detail": "Insufficient funds in
wallet."}, status=status.HTTP_400_BAD_REQUEST)

            user.wallet -= course.price
            user.save()

            instructor = course.instructor
            instructor.wallet += course.price
            instructor.save()

            enrollment = Enrollment.objects.create(user=user,
course=course, status='enrolled')

            return Response({
                "detail": "Enrolled successfully",
                "course_id": course.id,
                "user_id": user.id,
                "user_wallet_balance": str(user.wallet),
                "enrollment_id": enrollment.id
            }, status=status.HTTP_201_CREATED)

        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

class UserEnrollmentsView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request):
        enrollments = Enrollment.objects.filter(user=request.user,
status='enrolled').select_related('course')
        courses = [e.course for e in enrollments]
        serializer = CourseSerializer(courses, many=True)
        return Response(serializer.data)

class MarkLessonCompleteView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        course_id = request.data.get('course_id')
        lesson_id = request.data.get('lesson_id')
        if not course_id or not lesson_id:
            return Response({"detail": "course_id and lesson_id are
required."}, status=400)

        enrolled = Enrollment.objects.filter(user=request.user,
course_id=course_id, status='enrolled').exists()
        if not enrolled:
            return Response({"detail": "User not enrolled in this
course."}, status=403)

        try:
            lesson = Lesson.objects.get(id=lesson_id, course_id=course_id)
        except Lesson.DoesNotExist:
            return Response({"detail": "Lesson not found in this
course."}, status=404)

        # Update UserProgress
        progress, created =
UserProgress.objects.get_or_create(user=request.user, course_id=course_id)
        completed_lessons = progress.completed_lessons
        if lesson_id not in completed_lessons:
            completed_lessons.append(lesson_id)
        progress.completed_lessons = completed_lessons
        progress.save()

        return Response({"detail": "Lesson marked as completed."},
status=200)

class UserProgressView(APIView):
    permission_classes = [permissions.IsAuthenticated]
    def get(self, request):
        course_id = request.GET.get('course')
        if not course_id:
            return Response({"detail": "course parameter is required."},
status=400)

        enrolled = Enrollment.objects.filter(user=request.user,
course_id=course_id, status='enrolled').exists()
        if not enrolled:
            return Response({"detail": "User not enrolled in this
course."}, status=403)

        progress, created =
UserProgress.objects.get_or_create(user=request.user, course_id=course_id)
        return Response({
            "completed_lessons": progress.completed_lessons,
            "quiz_scores": progress.quiz_scores
        })
```

Code Snippets from payments app:

```
from django.db import models
from django.conf import settings

class Payment(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, related_name='payments')
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    payment_date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=20, default='completed')

    def __str__(self):
        return f"Payment of {self.amount} by {self.user.username} on
{self.payment_date}"
```

```
from rest_framework import serializers

class WalletTopUpSerializer(serializers.Serializer):
    amount = serializers.DecimalField(max_digits=10, decimal_places=2)

    def validate_amount(self, value):
        if value <= 0:
            raise serializers.ValidationError("Amount must be greater than
zero.")
        return value
```

```
from django.urls import path
from .views import WalletTopUpView

urlpatterns = [
    path('top-up/', WalletTopUpView.as_view(), name='wallet_top_up'),
]
```

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status, permissions
from .serializers import WalletTopUpSerializer
from .models import Payment

class WalletTopUpView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        serializer = WalletTopUpSerializer(data=request.data)
        if serializer.is_valid():
            amount = serializer.validated_data['amount']
            user = request.user
            user.wallet += amount
            user.save()
            Payment.objects.create(user=user, amount=amount,
status='completed')
            return Response({"wallet_balance": str(user.wallet)},
status=status.HTTP_200_OK)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

Code Snippets from quizzes app:

```
from django.db import models
from courses.models import Course

class Quiz(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='quizzes')
    title = models.CharField(max_length=200)
    total_marks = models.IntegerField(default=0)

    def __str__(self):
        return f"{self.title} - {self.course.title}"

class QuizQuestion(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE,
related_name='questions')
    question_text = models.TextField()
    option_a = models.CharField(max_length=200)
    option_b = models.CharField(max_length=200)
    option_c = models.CharField(max_length=200)
    option_d = models.CharField(max_length=200)
    correct_option = models.CharField(max_length=1)

    def __str__(self):
        return f"Question for {self.quiz.title}:
{self.question_text[:50]}"
```

```
from rest_framework import serializers
from .models import Quiz, QuizQuestion

class QuizQuestionSerializer(serializers.ModelSerializer):
    class Meta:
        model = QuizQuestion
        fields = ['id', 'quiz', 'question_text', 'option_a', 'option_b',
'option_c', 'option_d', 'correct_option']
        read_only_fields = ['quiz']

class QuizSerializer(serializers.ModelSerializer):
    questions = QuizQuestionSerializer(many=True, read_only=True)

    class Meta:
        model = Quiz
        fields = ['id', 'course', 'title', 'total_marks', 'questions']
        read_only_fields = ['course', 'questions']

class QuizAttemptSerializer(serializers.Serializer):
    quiz_id = serializers.IntegerField()
    answers = serializers.DictField(
        child=serializers.CharField()
    )

    def validate_quiz_id(self, value):
        if value <= 0:
            raise serializers.ValidationError("Invalid quiz_id.")
        return value
```

```
from django.urls import path
from rest_framework.routers import DefaultRouter
from .views import QuizViewSet, QuizQuestionViewSet, QuizAttemptView

router = DefaultRouter()
router.register(r'quizzes', QuizViewSet, basename='quiz')
router.register(r'quiz-questions', QuizQuestionViewSet,
basename='quiz-question')

urlpatterns = [
    path('quizzes/attempt/', QuizAttemptView.as_view(),
name='quiz_attempt'),
] + router.urls
```

```
from rest_framework import viewsets, permissions, status
from rest_framework.response import Response
from django.shortcuts import get_object_or_404
from .models import Quiz, QuizQuestion
from .serializers import QuizSerializer, QuizQuestionSerializer,
QuizAttemptSerializer
from courses.models import Course
from enrollments.models import Enrollment, UserProgress
from rest_framework.views import APIView

class QuizViewSet(viewsets.ModelViewSet):
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    def get_queryset(self):
        user = self.request.user
        queryset = super().get_queryset()
        if not user.is_authenticated:
            return queryset.none()

        instructor_courses =
Course.objects.filter(instructor=user).values_list('id', flat=True)
        enrolled_course_ids =
Enrollment.objects.filter(user=user).values_list('course_id', flat=True)
        allowed_courses =
set(instructor_courses).union(enrolled_course_ids)
        return queryset.filter(course_id__in=allowed_courses)

    def perform_create(self, serializer):
        user = self.request.user
        course_id = self.request.data.get('course')
        if course_id is None:
            return Response({"detail": "Course is required."},
status=status.HTTP_400_BAD_REQUEST)

        course = get_object_or_404(Course, id=course_id)
        if course.instructor != user:
            return Response({"detail": "Only the instructor can create
quizzes for this course."}, status=status.HTTP_403_FORBIDDEN)

        serializer.save(course=course)

class QuizQuestionViewSet(viewsets.ModelViewSet):
    queryset = QuizQuestion.objects.all()
    serializer_class = QuizQuestionSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def get_queryset(self):
        user = self.request.user
        queryset = super().get_queryset()
        if not user.is_authenticated:
            return queryset.none()

        instructor_courses =
Course.objects.filter(instructor=user).values_list('id', flat=True)
        enrolled_course_ids =
Enrollment.objects.filter(user=user).values_list('course_id', flat=True)
        allowed_courses =
set(instructor_courses).union(enrolled_course_ids)
        return queryset.filter(quiz__course_id__in=allowed_courses)

    def perform_create(self, serializer):
        user = self.request.user
        quiz_id = self.request.data.get('quiz')
        if quiz_id is None:
            return Response({"detail": "Quiz is required."},
status=status.HTTP_400_BAD_REQUEST)
```

```
        quiz = get_object_or_404(Quiz, id=quiz_id)
        if quiz.course.instructor != user:
            return Response({"detail": "Only the instructor can add
questions to this quiz."}, status=status.HTTP_403_FORBIDDEN)

        serializer.save(quiz=quiz)

class QuizAttemptView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        serializer = QuizAttemptSerializer(data=request.data)
        if serializer.is_valid():
            quiz_id = serializer.validated_data['quiz_id']
            answers = serializer.validated_data['answers']
            user = request.user

            quiz = get_object_or_404(Quiz, id=quiz_id)
            enrolled = Enrollment.objects.filter(user=user,
course=quiz.course).exists()
            if not enrolled:
                return Response({"detail": "You must be enrolled in this
course to attempt the quiz."}, status=status.HTTP_403_FORBIDDEN)

            questions = quiz.questions.all()
            correct_count = 0
            for q in questions:
                user_answer = answers.get(str(q.id))
                if user_answer and user_answer.upper() ==
q.correct_option:
                    correct_count += 1

            total_questions = questions.count()
            score = int((correct_count / total_questions) *
quiz.total_marks) if total_questions > 0 else 0

            progress, created =
UserProgress.objects.get_or_create(user=user, course=quiz.course)
            quiz_scores = progress.quiz_scores
            quiz_scores[str(quiz.id)] = score
            progress.quiz_scores = quiz_scores
            progress.save()

            return Response({
                "detail": "Quiz attempted.",
                "score": score,
                "correct_answers": correct_count,
                "total_questions": total_questions
            }, status=status.HTTP_200_OK)

        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

Code Snippets from reviews app:

```python
from django.db import models
from django.conf import settings
from courses.models import Course

class Review(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='reviews')
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, related_name='reviews')
    rating = models.IntegerField()
    comment = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Review by {self.user.username} on {self.course.title}"
```

```python
from rest_framework import serializers
from .models import Review

class ReviewSerializer(serializers.ModelSerializer):
    class Meta:
        model = Review
        fields = ['id', 'course', 'user', 'rating', 'comment',
'created_at']
        read_only_fields = ['user', 'created_at']

    def validate_rating(self, value):
        if value < 1 or value > 5:
            raise serializers.ValidationError("Rating must be between 1
and 5.")
        return value
```

```python
from rest_framework.routers import DefaultRouter
from .views import ReviewViewSet

router = DefaultRouter()
router.register(r'reviews', ReviewViewSet, basename='review')

urlpatterns = router.urls
```

```python
from rest_framework import viewsets, permissions, status
from rest_framework.response import Response
from django.shortcuts import get_object_or_404
from .models import Review
from .serializers import ReviewSerializer
from enrollments.models import Enrollment
from courses.models import Course

class ReviewViewSet(viewsets.ModelViewSet):
    queryset = Review.objects.all()
    serializer_class = ReviewSerializer

    def get_queryset(self):
        queryset = super().get_queryset()
        course_id = self.request.query_params.get('course')
        if course_id:
            queryset = queryset.filter(course_id=course_id)
        return queryset

    def get_permissions(self):
        if self.request.method in ['POST', 'PUT', 'PATCH', 'DELETE']:
            return [permissions.IsAuthenticated()]
        return [permissions.AllowAny()]

    def perform_create(self, serializer):
        user = self.request.user
        course_id = self.request.data.get('course')
        if course_id is None:
            return Response({"detail": "Course ID is required."},
status=status.HTTP_400_BAD_REQUEST)

        course = get_object_or_404(Course, id=course_id)

        if course.instructor == user:
            return Response({"detail": "Instructors cannot review their
own course."}, status=status.HTTP_403_FORBIDDEN)

        enrolled = Enrollment.objects.filter(user=user,
course=course).exists()
        if not enrolled:
            return Response({"detail": "User must be enrolled in the
course to leave a review."}, status=status.HTTP_403_FORBIDDEN)

        serializer.save(user=user, course=course)
```

Code Snippet of App.js file:

```
import React, { useEffect, useState } from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from
'react-router-dom';
import API from './services/api';
import Login from './pages/Login';
import Register from './pages/Register';
import Dashboard from './pages/Dashboard';
import MyCourses from './pages/MyCourses';
import CourseDetail from './pages/CourseDetail';
import QuizAttempt from './pages/QuizzAttempt';
import InstructorDashboard from './pages/InstructorDashboard';
import ManageCourse from './pages/ManageCourse';
import ManageQuiz from './pages/ManageQuiz';
import './styles.css';

const App = () => {
  const isAuthenticated = !!localStorage.getItem('access_token');
  const [userInfo, setUserInfo] = useState(null);

  const handleLogout = () => {
    localStorage.removeItem('access_token');
    localStorage.removeItem('refresh_token');
    window.location.href = '/login';
  };

  useEffect(() => {
    const fetchUserInfo = async () => {
      if (isAuthenticated) {
        try {
          const response = await API.get('accounts/user/');
          setUserInfo(response.data);
        } catch (err) {
          console.error('Failed to fetch user info', err);
        }
      }
    };
    fetchUserInfo();
  }, [isAuthenticated]);

  return (
    <Router>
      <nav style={{ padding: '10px', borderBottom: '1px solid #ccc' }}>
        <Link to="/">Home</Link> |{' '}
        {isAuthenticated ? (
          <>
            <Link to="/dashboard">Dashboard</Link> |{' '}
            <Link to="/my-courses">My Courses</Link> |{' '}
            {userInfo && userInfo.is_instructor && <Link
to="/instructor">Instructor</Link>} |{' '}
            <button onClick={handleLogout}>Logout</button>
          </>
        ) : (
          <>
            <Link to="/login">Login</Link> |{' '}
            <Link to="/register">Register</Link>
          </>
        )}
      </nav>

      <div style={{ padding: '20px' }}>
        <Routes>
          <Route path="/" element={<h1>Welcome to E-Learning</h1>} />
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />
          {isAuthenticated && <Route path="/dashboard" element={<Dashboard
/>} />}
          {isAuthenticated && <Route path="/my-courses"
element={<MyCourses />} />}
          {isAuthenticated && <Route path="/courses/:id"
element={<CourseDetail />} />}
          {isAuthenticated && <Route path="/courses/:id/quizzes/:quiz_id"
element={<QuizAttempt />} />}
          {isAuthenticated && userInfo && userInfo.is_instructor && (
            <Route path="/instructor" element={<InstructorDashboard
userInfo={userInfo} />} />
          )}
          {isAuthenticated && userInfo && userInfo.is_instructor && (
            <>
              <Route path="/instructor" element={<InstructorDashboard
userInfo={userInfo} />} />
              <Route path="/instructor/manage-course/:course_id"
element={<ManageCourse />} />
              <Route path="/instructor/manage-quiz/:quiz_id"
element={<ManageQuiz />} />
            </>
          )}
        </Routes>
      </div>
    </Router>
  );
};

export default App;
```

Code Snippet from api.js:

```
import axios from 'axios';

const API = axios.create({
  baseURL: 'http://localhost:8000/api/',
});

API.interceptors.request.use((config) => {
  const token = localStorage.getItem('access_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

API.interceptors.response.use(
  response => response,
  async (error) => {
    const originalRequest = error.config;
    if (error.response && error.response.status === 401 &&
!originalRequest._retry) {
      originalRequest._retry = true;

      const refreshToken = localStorage.getItem('refresh_token');
      if (refreshToken) {
        try {
          const response = await
axios.post('http://localhost:8000/api/accounts/token/refresh/', {
            refresh: refreshToken
          });

          const newAccessToken = response.data.access;
          localStorage.setItem('access_token', newAccessToken);

          originalRequest.headers['Authorization'] = `Bearer
${newAccessToken}`;
          return API(originalRequest);

        } catch (refreshError) {
          console.error('Failed to refresh token', refreshError);
          localStorage.removeItem('access_token');
          localStorage.removeItem('refresh_token');
          window.location.href = '/login';
        }
      } else {
        localStorage.removeItem('access_token');
        localStorage.removeItem('refresh_token');
        window.location.href = '/login';
      }
    }
    return Promise.reject(error);
  }
);

export default API;
```

Code snippets from page folder (UI files):

```
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import API from '../services/api';
import { Link } from 'react-router-dom';

const CourseDetail = () => {
  const { id } = useParams();
  const [courseInfo, setCourseInfo] = useState(null);
  const [lessons, setLessons] = useState([]);
  const [quizzes, setQuizzes] = useState([]);
  const [reviews, setReviews] = useState([]);
  const [userInfo, setUserInfo] = useState(null);

  const [isEnrolled, setIsEnrolled] = useState(false);

  const [rating, setRating] = useState(5);
  const [comment, setComment] = useState('');
  const [reviewMessage, setReviewMessage] = useState('');

  const [userProgress, setUserProgress] = useState(null);

  useEffect(() => {
    const fetchUserInfo = async () => {
      try {
        const response = await API.get('accounts/user/');
        setUserInfo(response.data);
      } catch (err) {
        console.error('Failed to fetch user info', err);
      }
    };
    fetchUserInfo();
  }, []);

  useEffect(() => {
    const checkEnrollment = async () => {
      if (userInfo) {
        try {
          const enrollResp = await
API.get('enrollments/user-enrollments/');
          const enrolledCourses = enrollResp.data;
          const enrolledCourseIds = enrolledCourses.map(c => c.id);
          setIsEnrolled(enrolledCourseIds.includes(Number(id)));
        } catch (err) {
          console.error('Failed to check enrollment', err);
        }
      }
    };
    checkEnrollment();
  }, [userInfo, id]);

  useEffect(() => {
    const fetchCourseInfo = async () => {
      try {
        const response = await API.get(`courses/${id}/`);
        setCourseInfo(response.data);
      } catch (err) {
        console.error('Failed to fetch course detail', err);
      }
    };

    const fetchLessons = async () => {
      try {
        const response = await API.get(`lessons/?course=${id}`);
        setLessons(response.data);
      } catch (err) {
        console.error('Failed to fetch lessons', err);
      }
    };

    const fetchQuizzes = async () => {
      try {
        const response = await API.get(`quizzes/?course=${id}`);
        setQuizzes(response.data);
      } catch (err) {
        console.error('Failed to fetch quizzes', err);
      }
    };

    const fetchReviews = async () => {
      try {
        const response = await API.get(`reviews/?course=${id}`);
        setReviews(response.data);
      } catch (err) {
        console.error('Failed to fetch reviews', err);
      }
    };

    fetchCourseInfo();
    fetchLessons();
    fetchQuizzes();
    fetchReviews();
  }, [id]);

  useEffect(() => {
    const fetchProgress = async () => {
      if (userInfo && isEnrolled) {
        try {
          const resp = await
API.get(`enrollments/progress/?course=${id}`);
          setUserProgress(resp.data);
        } catch (err) {
          console.error('Failed to fetch user progress', err);
        }
      }
    };
    fetchProgress();
  }, [userInfo, isEnrolled, id]);

  const isInstructor = userInfo && courseInfo && (courseInfo.instructor
=== userInfo.id);
  const canReview = isEnrolled && !isInstructor;

  const handleReviewSubmit = async (e) => {
    e.preventDefault();
    if (!canReview) return;
    try {
      await API.post('reviews/', {
        course: Number(id),
        rating: Number(rating),
        comment
      });
      setReviewMessage('Review posted successfully!');
      setComment('');
      setRating(5);
      const response = await API.get(`reviews/?course=${id}`);
      setReviews(response.data);
    } catch (err) {
      console.error('Failed to post review', err);
      setReviewMessage('Failed to post review.');
    }
  };

  const markLessonComplete = async (lessonId) => {
    try {
      await API.post('enrollments/mark-lesson-complete/', {
        course_id: Number(id),
        lesson_id: lessonId
      });
      const resp = await API.get(`enrollments/progress/?course=${id}`);
      setUserProgress(resp.data);
    } catch (err) {
      console.error('Failed to mark lesson complete', err);
    }
  };

  return (
    <div>
      <h2>Course Detail</h2>
      {courseInfo ? (
        <div>
          <h3>{courseInfo.title}</h3>
          <p>{courseInfo.description}</p>
        </div>
      ) : (
        <p>Loading course details...</p>
      )}

      <h3>Lessons</h3>
      <ul className="lesson-list">
        {lessons.map((lesson) => {
          const completed = userProgress &&
userProgress.completed_lessons.includes(lesson.id);
          return (
            <li key={lesson.id} className="lesson-item">
              <strong>{lesson.title}</strong><br/>
              {lesson.content}<br/>
              {lesson.video_url && <a href={lesson.video_url}
target="_blank" rel="noopener noreferrer" className="video-link">Watch
Video</a>}

              {isEnrolled && (
                completed ? (
                  <span className="done-icon">✓ Completed</span>
                ) : (
                  <button className="complete-btn" onClick={() =>
markLessonComplete(lesson.id)}>
                    Mark as Completed
                  </button>
                )
              )}
            </li>
          );
        })}
      </ul>

      <h3>Quizzes</h3>
      {quizzes.length === 0 ? (
        <p>No quizzes available.</p>
      ) : (
        <ul>
```

```
          {quizzes.map((quiz) => (
            <li key={quiz.id}>
              {quiz.title} (Total Marks: {quiz.total_marks}){' '}
              <Link to={`/courses/${id}/quizzes/${quiz.id}`}>Attempt
Quiz</Link>
            </li>
          ))}
        </ul>
      )}

      <h3>Your Progress</h3>
      {userProgress ? (
        <div>
          <p>Completed Lessons:
{userProgress.completed_lessons.length}</p>
          <p>Quiz Scores:</p>
          {Object.entries(userProgress.quiz_scores).length === 0 ? (
            <p>No quiz scores yet.</p>
          ) : (
            <ul>
              {Object.entries(userProgress.quiz_scores).map(([quizId,
score]) => (
                <li key={quizId}>Quiz {quizId}: {score} points</li>
              ))}
            </ul>
          )}
        </div>
      ) : (
        isEnrolled ? <p>Loading progress...</p> : <p>Enroll to track
progress.</p>
      )}

      <h3>Reviews</h3>
      {reviews.length === 0 ? (
        <p>No reviews yet.</p>
      ) : (
        <ul>
          {reviews.map((review) => (
            <li key={review.id}>
              <strong>Rating:</strong> {review.rating} / 5<br/>
              <em>{review.comment}</em><br/>
              By User ID: {review.user}
            </li>
          ))}
        </ul>
      )}

      {canReview && (
        <div style={{ marginTop: '20px' }}>
          <h4>Leave a Review</h4>
          <form onSubmit={handleReviewSubmit}>
            <label>
              Rating (1-5):
              <input
```

```
                type="number"
                min="1"
                max="5"
                value={rating}
                onChange={(e) => setRating(e.target.value)}
              />
            </label><br/>
            <label>
              Comment:<br/>
              <textarea
                value={comment}
                onChange={(e) => setComment(e.target.value)}
              />
            </label><br/>
            <button type="submit">Submit Review</button>
          </form>
          {reviewMessage && <p>{reviewMessage}</p>}
        </div>
      )}

      {!canReview && userInfo && (
        <p>You must be enrolled and not the instructor to leave a
review.</p>
      )}
    </div>
  );
};

export default CourseDetail;
```

```
import React, { useEffect, useState } from 'react';
import API from '../services/api';

const Dashboard = () => {
  const [courses, setCourses] = useState([]);
  const [userInfo, setUserInfo] = useState(null);
  const [topUpAmount, setTopUpAmount] = useState('');
  const [categories, setCategories] = useState([]);
  const [selectedCategory, setSelectedCategory] = useState('');

  const fetchUserInfo = async () => {
    try {
      const response = await API.get('accounts/user/');
      setUserInfo(response.data);
    } catch (err) {
      console.error('Failed to fetch user info', err);
    }
  };

  const fetchCategories = async () => {
    try {
      const response = await API.get('categories/');
      setCategories(response.data);
    } catch (err) {
      console.error('Failed to fetch categories', err);
    }
  };

  const fetchCourses = async (categoryId = '') => {
    try {
      let url = 'courses/';
      if (categoryId) {
        url += `?category=${categoryId}`;
      }
      const response = await API.get(url);
      setCourses(response.data);
    } catch (err) {
      console.error('Failed to fetch courses', err);
    }
  };

  useEffect(() => {
    fetchUserInfo();
    fetchCategories();
    fetchCourses();
  }, []);

  const handleTopUp = async (e) => {
    e.preventDefault();
    if (!topUpAmount) return;

    try {
      const response = await API.post('payments/top-up/', { amount:
topUpAmount });
      alert(`Wallet topped up! New balance:
$${response.data.wallet_balance}`);
      setTopUpAmount('');
      fetchUserInfo();
    } catch (err) {
      console.error('Top-up error', err);
      alert('Failed to top up.');
    }
  };

  const handlePurchase = async (courseId) => {
    try {
      await API.post('enrollments/purchase/', { course_id: courseId });
      alert(`Enrolled successfully in course ID ${courseId}!`);
      fetchUserInfo();
    } catch (err) {
      console.error('Purchase error', err);
      alert('Failed to purchase course.');
    }
  };

  const handleCategoryChange = (e) => {
    const categoryId = e.target.value;
    setSelectedCategory(categoryId);
    fetchCourses(categoryId);
  };

  return (
    <div>
      <h2>Your Dashboard</h2>

      {userInfo && (
        <div style={{ marginBottom: '20px' }}>
          <p><strong>Username:</strong> {userInfo.username}</p>
          <p><strong>Role:</strong> {userInfo.is_instructor ? 'Instructor'
: 'Student'}</p>
          <p><strong>Wallet Balance:</strong> ${userInfo.wallet}</p>

          <form onSubmit={handleTopUp} style={{ marginTop: '10px' }}>
            <input
              type="number"
              step="0.01"
              placeholder="Top-up Amount"
              value={topUpAmount}
              onChange={(e) => setTopUpAmount(e.target.value)}
            />
            <button type="submit">Top Up</button>
          </form>
        </div>
      )}

      <h3>Filter by Category</h3>
      <select value={selectedCategory} onChange={handleCategoryChange}>
        <option value="">All Categories</option>
        {categories.map((cat) => (
          <option key={cat.id} value={cat.id}>{cat.name}</option>
        ))}
      </select>

      <h3>Available Courses</h3>
      <ul>
        {courses.map((course) => {
          const isCourseInstructor = userInfo && course.instructor ===
userInfo.id;
          const canAfford = userInfo && (userInfo.wallet >= course.price);
          return (
            <li key={course.id}>
              {course.title} - ${course.price}{' '}
              {!isCourseInstructor && userInfo && (
                <button
                  onClick={() => handlePurchase(course.id)}
                  disabled={!canAfford}
                >
                  {canAfford ? 'Buy' : 'Not enough funds'}
                </button>
              )}
            </li>
          );
        })}
      </ul>
    </div>
  );
};

export default Dashboard;
```

```
import React, { useEffect, useState } from 'react';
import API from '../services/api';
import { Link } from 'react-router-dom';

const InstructorDashboard = ({ userInfo }) => {
  const [myCreatedCourses, setMyCreatedCourses] = useState([]);
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [price, setPrice] = useState('');
  const [categoryId, setCategoryId] = useState('');
  const [categories, setCategories] = useState([]);
  const [message, setMessage] = useState('');

  useEffect(() => {
    const fetchMyCourses = async () => {
      try {
        const response = await
API.get(`courses/?instructor=${userInfo.id}`);
        setMyCreatedCourses(response.data);
      } catch (err) {
        console.error('Failed to fetch instructor courses', err);
      }
    };

    const fetchCategories = async () => {
      try {
        const resp = await API.get('categories/');
        setCategories(resp.data);
      } catch (err) {
        console.error('Failed to fetch categories', err);
      }
    };

    fetchMyCourses();
    fetchCategories();
  }, [userInfo.id]);

  const handleCreateCourse = async (e) => {
    e.preventDefault();
    setMessage('');
    if (!title || !description || !price || !categoryId) {
      setMessage('Please fill all fields.');
      return;
    }
    try {
      await API.post('courses/', {
        title,
        description,
        price: parseFloat(price),
        category: parseInt(categoryId)
      });
      setMessage('Course created successfully!');
      setTitle('');
      setDescription('');
      setPrice('');
      setCategoryId('');

      const response = await
API.get(`courses/?instructor=${userInfo.id}`);
      setMyCreatedCourses(response.data);
    } catch (err) {
      console.error('Failed to create course', err);
      setMessage('Failed to create course.');
    }
  };

  return (
    <div className="container">
      <h2>Instructor Dashboard</h2>
      <h3>Create a New Course</h3>
      <form onSubmit={handleCreateCourse}>
        <label>Title:<br/>
          <input value={title} onChange={(e) => setTitle(e.target.value)}
/>
        </label><br/>
        <label>Description:<br/>
          <textarea value={description} onChange={(e) =>
setDescription(e.target.value)} />
        </label><br/>
        <label>Price:<br/>
          <input type="number" step="0.01" value={price} onChange={(e) =>
setPrice(e.target.value)} />
        </label><br/>
        <label>Category:<br/>
          <select value={categoryId} onChange={(e) =>
setCategoryId(e.target.value)}>
            <option value="">Select Category</option>
            {categories.map(cat => (
              <option key={cat.id} value={cat.id}>{cat.name}</option>
            ))}
          </select>
        </label><br/>
        <button type="submit">Create Course</button>
      </form>
      {message && <p className="message">{message}</p>}

      <h3>My Created Courses</h3>
      <ul className="lesson-list">
        {myCreatedCourses.map(course => (
          <li key={course.id} className="lesson-item">
```

```
            {course.title} - ${course.price}
            <Link to={`/instructor/manage-course/${course.id}`} style={{
marginLeft:'10px' }}>Manage</Link>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default InstructorDashboard;
```

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import API from '../services/api';

const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    setError('');
    try {
      const response = await API.post('accounts/token/', { username,
password });
      localStorage.setItem('access_token', response.data.access);
      localStorage.setItem('refresh_token', response.data.refresh);
      navigate('/dashboard');
    } catch (err) {
      console.error('Login error', err);
      setError('Invalid username or password');
    }
  };

  return (
    <div style={{ padding: '20px' }}>
      <h2>Login</h2>
      <form onSubmit={handleLogin}>
        <div>
          <label>Username:</label><br/>
          <input
            type="text"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
          />
        </div>
        <div>
          <label>Password:</label><br/>
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
        </div>
        <button type="submit">Login</button>
      </form>
      {error && <p style={{ color:'red' }}>{error}</p>}
    </div>
  );
};

export default Login;
```

```
import React, { useEffect, useState } from 'react';
import { useParams, Link } from 'react-router-dom';
import API from '../services/api';

const ManageCourse = () => {
  const { course_id } = useParams();
  const [courseInfo, setCourseInfo] = useState(null);
  const [lessons, setLessons] = useState([]);
  const [quizzes, setQuizzes] = useState([]);
  const [selectedLesson, setSelectedLesson] = useState(null);

  const [newLessonTitle, setNewLessonTitle] = useState('');
  const [newLessonContent, setNewLessonContent] = useState('');
  const [newLessonVideoURL, setNewLessonVideoURL] = useState('');
  const [lessonMessage, setLessonMessage] = useState('');

  useEffect(() => {
    const fetchCourse = async () => {
      try {
        const resp = await API.get(`courses/${course_id}/`);
        setCourseInfo(resp.data);
      } catch (err) {
        console.error('Failed to fetch course info', err);
      }
    };

    const fetchLessons = async () => {
      try {
        const resp = await API.get(`lessons/?course=${course_id}`);
        setLessons(resp.data);
      } catch (err) {
        console.error('Failed to fetch lessons', err);
      }
    };

    const fetchQuizzes = async () => {
      try {
        const resp = await API.get(`quizzes/?course=${course_id}`);
        setQuizzes(resp.data);
      } catch (err) {
        console.error('Failed to fetch quizzes', err);
      }
    };

    fetchCourse();
    fetchLessons();
    fetchQuizzes();
  }, [course_id]);

  const handleViewLessonDetails = (lesson) => {
    setSelectedLesson(lesson);
  };

  const handleAddLesson = async (e) => {
    e.preventDefault();
    setLessonMessage('');

    if (!newLessonTitle || !newLessonContent) {
      setLessonMessage('Title and content are required.');
      return;
    }

    try {
      await API.post('lessons/', {
        course: Number(course_id),
        title: newLessonTitle,
        content: newLessonContent,
        video_url: newLessonVideoURL
      });

      setLessonMessage('Lesson added successfully!');
      setNewLessonTitle('');
      setNewLessonContent('');
      setNewLessonVideoURL('');

      const resp = await API.get(`lessons/?course=${course_id}`);
      setLessons(resp.data);

    } catch (err) {
      console.error('Failed to add lesson', err);
      setLessonMessage('Failed to add lesson.');
    }
  };

  return (
    <div className="container">
      <h2>Manage Course</h2>
      {courseInfo && (
        <div className="course-info">
          <h3>{courseInfo.title}</h3>
          <p>{courseInfo.description}</p>
        </div>
      )}

      <h3>Lessons</h3>
      <ul className="lesson-list">
```

```
        {lessons.map(l => (
          <li key={l.id} className="lesson-item">
            <strong>{l.title}</strong>
            <button style={{ marginLeft:'10px' }} onClick={() =>
handleViewLessonDetails(l)}>View Details</button>
          </li>
        ))}
      </ul>

      {selectedLesson && (
        <div className="course-info" style={{ marginTop:'10px' }}>
          <h4>Lesson Details</h4>
          <p><strong>Title:</strong> {selectedLesson.title}</p>
          <p><strong>Content:</strong> {selectedLesson.content}</p>
          {selectedLesson.video_url && <p><a
href={selectedLesson.video_url} target="_blank" rel="noopener
noreferrer">Watch Video</a></p>}
          <button onClick={() => setSelectedLesson(null)} style={{
marginTop:'5px' }}>Close Details</button>
        </div>
      )}

      <h4>Add New Lesson</h4>
      <form onSubmit={handleAddLesson}>
        <label>Title:<br/>
          <input value={newLessonTitle} onChange={(e) =>
setNewLessonTitle(e.target.value)} />
        </label><br/>
        <label>Content:<br/>
          <textarea value={newLessonContent} onChange={(e) =>
setNewLessonContent(e.target.value)} />
        </label><br/>
        <label>Video URL:<br/>
          <input value={newLessonVideoURL} onChange={(e) =>
setNewLessonVideoURL(e.target.value)} />
        </label><br/>
        <button type="submit">Add Lesson</button>
      </form>
      {lessonMessage && <p>{lessonMessage}</p>}

      <h3>Quizzes</h3>
      <ul className="lesson-list">
        {quizzes.map(q => (
          <li key={q.id} className="lesson-item">
            {q.title} - {q.total_marks} marks
            <Link to={`/instructor/manage-quiz/${q.id}`} style={{
marginLeft:'10px' }}>Manage Quiz</Link>
          </li>
        ))}
      </ul>
    </div>
  );
};
```

```
export default ManageCourse;
```

```
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import API from '../services/api';

const ManageQuiz = () => {
  const { quiz_id } = useParams();
  const [quizInfo, setQuizInfo] = useState(null);
  const [questions, setQuestions] = useState([]);

  const [questionText, setQuestionText] = useState('');
  const [optionA, setOptionA] = useState('');
  const [optionB, setOptionB] = useState('');
  const [optionC, setOptionC] = useState('');
  const [optionD, setOptionD] = useState('');
  const [correctOption, setCorrectOption] = useState('A');

  const [message, setMessage] = useState('');

  useEffect(() => {
    const fetchQuiz = async () => {
      try {
        const resp = await API.get(`quizzes/${quiz_id}/`);
        setQuizInfo(resp.data);
      } catch (err) {
        console.error('Failed to fetch quiz info', err);
      }
    };

    const fetchQuestions = async () => {
      try {
        const resp = await API.get(`quiz-questions/?quiz=${quiz_id}`);
        setQuestions(resp.data);
      } catch (err) {
        console.error('Failed to fetch quiz questions', err);
      }
    };

    fetchQuiz();
    fetchQuestions();
  }, [quiz_id]);

  const handleAddQuestion = async (e) => {
    e.preventDefault();
    setMessage('');
    if (!questionText || !optionA || !optionB || !optionC || !optionD) {
      setMessage('Please fill in all fields.');
      return;
    }

    try {
      await API.post('quiz-questions/', {
        quiz: Number(quiz_id),
        question_text: questionText,
        option_a: optionA,
        option_b: optionB,
        option_c: optionC,
        option_d: optionD,
        correct_option: correctOption
      });

      setMessage('Question added successfully!');
      setQuestionText('');
      setOptionA('');
      setOptionB('');
      setOptionC('');
      setOptionD('');
      setCorrectOption('A');

      const resp = await API.get(`quiz-questions/?quiz=${quiz_id}`);
      setQuestions(resp.data);

    } catch (err) {
      console.error('Failed to add question', err);
      setMessage('Failed to add question.');
    }
  };

  return (
    <div className="container">
      <h2>Manage Quiz</h2>
      {quizInfo ? (
        <div className="quiz-info">
          <h3>{quizInfo.title}</h3>
          <p>Total Marks: {quizInfo.total_marks}</p>
        </div>
      ) : (
        <p>Loading quiz info...</p>
      )}

      <h3>Add New Question</h3>
      <form onSubmit={handleAddQuestion} className="question-form">
        <label>Question Text:
          <textarea
            value={questionText}
            onChange={(e) => setQuestionText(e.target.value)}
          />
        </label>

        <div className="options-container">
          <label>Option A:
            <input
              type="text"
              value={optionA}
              onChange={(e) => setOptionA(e.target.value)}
              className="option-input"
            />
          </label>
          <label>Option B:
            <input
              type="text"
              value={optionB}
              onChange={(e) => setOptionB(e.target.value)}
              className="option-input"
            />
          </label>
          <label>Option C:
            <input
              type="text"
              value={optionC}
              onChange={(e) => setOptionC(e.target.value)}
              className="option-input"
            />
          </label>
          <label>Option D:
            <input
              type="text"
              value={optionD}
              onChange={(e) => setOptionD(e.target.value)}
              className="option-input"
            />
          </label>
        </div>

        <label>Correct Option:
          <select value={correctOption} onChange={(e) =>
setCorrectOption(e.target.value)}>
            <option value="A">A</option>
            <option value="B">B</option>
            <option value="C">C</option>
            <option value="D">D</option>
          </select>
        </label>

        <button type="submit" className="add-question-btn">Add
Question</button>
      </form>
      {message && <p className="message">{message}</p>}

      <h3>Existing Questions</h3>
      <ul className="questions-list">
        {questions.map((q) => (
          <li key={q.id} className="question-item">
            <p><strong>{q.question_text}</strong></p>
            <ul className="option-list">
              <li className={q.correct_option === 'A' ? 'correct-option' :
''}>A: {q.option_a}</li>
              <li className={q.correct_option === 'B' ? 'correct-option' :
''}>B: {q.option_b}</li>
              <li className={q.correct_option === 'C' ? 'correct-option' :
''}>C: {q.option_c}</li>
```

```
              <li className={q.correct_option === 'D' ? 'correct-option' :
''}>D: {q.option_d}</li>
            </ul>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default ManageQuiz;
```

```
import React, { useEffect, useState } from 'react';
import API from '../services/api';
import { Link } from 'react-router-dom';

const MyCourses = () => {
  const [enrolledCourses, setEnrolledCourses] = useState([]);

  useEffect(() => {
    const fetchEnrolledCourses = async () => {
      try {
        const response = await API.get('enrollments/user-enrollments/');
        setEnrolledCourses(response.data);
      } catch (err) {
        console.error('Failed to fetch enrolled courses', err);
      }
    };
    fetchEnrolledCourses();
  }, []);

  return (
    <div>
      <h2>My Courses</h2>
      {enrolledCourses.length === 0 ? (
        <p>You are not enrolled in any courses yet.</p>
      ) : (
        <ul>
          {enrolledCourses.map((course) => (
            <li key={course.id}>
              {course.title} - ${course.price}{' '}
              <Link to={`/courses/${course.id}`}>View Lessons</Link>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default MyCourses;
```

```
import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import API from '../services/api';

const QuizAttempt = () => {
  const { id, quiz_id } = useParams();
  const [questions, setQuestions] = useState([]);
  const [answers, setAnswers] = useState({});
  const [score, setScore] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchQuestions = async () => {
      try {
        const response = await API.get(`quiz-questions/?quiz=${quiz_id}`);
        setQuestions(response.data);
      } catch (err) {
        console.error('Failed to fetch quiz questions', err);
      }
    };

    fetchQuestions();
  }, [quiz_id]);

  const handleChange = (questionId, option) => {
    setAnswers({ ...answers, [questionId]: option });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await API.post('quizzes/attempt/', {
        quiz_id: quiz_id,
        answers: answers
      });
      setScore(response.data.score);
    } catch (err) {
      console.error('Quiz attempt failed', err);
      alert('Failed to submit quiz answers. Check console for details.');
    }
  };

  return (
    <div>
      <h2>Attempt Quiz</h2>
      {score !== null ? (
        <div>
          <p>Your score: {score}</p>
          <button onClick={() => navigate(`/courses/${id}`)}>Back to
Course</button>
        </div>
      ) : (
        <form onSubmit={handleSubmit}>
          {questions.map((q) => (
            <div key={q.id} style={{ marginBottom: '20px' }}>
              <p><strong>{q.question_text}</strong></p>
              <label>
                <input
                  type="radio"
                  name={`q_${q.id}`}
                  value="A"
                  onChange={() => handleChange(q.id, "A")}
                />
                {q.option_a}
              </label><br/>
              <label>
                <input
                  type="radio"
                  name={`q_${q.id}`}
                  value="B"
                  onChange={() => handleChange(q.id, "B")}
                />
                {q.option_b}
              </label><br/>
              <label>
                <input
                  type="radio"
                  name={`q_${q.id}`}
                  value="C"
                  onChange={() => handleChange(q.id, "C")}
                />
                {q.option_c}
              </label><br/>
              <label>
                <input
                  type="radio"
                  name={`q_${q.id}`}
                  value="D"
                  onChange={() => handleChange(q.id, "D")}
                />
                {q.option_d}
              </label>
            </div>
          ))}
          {questions.length > 0 && <button type="submit">Submit
Quiz</button>}
        </form>
      )}
    </div>
  );
};

export default QuizAttempt;
```

```
import React, { useState } from 'react';
import API from '../services/api';

const Register = () => {
  const [formData, setFormData] = useState({
    username: '',
    email: '',
    password: '',
    is_student: true,
    is_instructor: false,
    wallet: 0
  });
  const [message, setMessage] = useState('');

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleRoleChange = (role) => {
    setFormData({
      ...formData,
      is_student: (role === 'student'),
      is_instructor: (role === 'instructor')
    });
  };

  const handleRegister = async (e) => {
    e.preventDefault();
    setMessage('');
    try {
      await API.post('accounts/register/', formData);
      setMessage('Registration successful! You can now log in.');
    } catch (err) {
      console.error('Registration error', err);
      setMessage('Registration failed. Please try again.');
    }
  };

  return (
    <div style={{ padding:'20px' }}>
      <h2>Register</h2>
      <form onSubmit={handleRegister}>
        <input
          type="text"
          name="username"
          placeholder="Username"
          value={formData.username}
          onChange={handleChange}
        /><br/>

        <input
          type="email"
          name="email"
          placeholder="Email"
          value={formData.email}
          onChange={handleChange}
        /><br/>

        <input
          type="password"
          name="password"
          placeholder="Password"
          value={formData.password}
          onChange={handleChange}
        /><br/>

        <div>
          <label>
            <input
              type="radio"
              checked={formData.is_student}
              onChange={() => handleRoleChange('student')}
            />
            Student
          </label>
          <label>
            <input
              type="radio"
              checked={formData.is_instructor}
              onChange={() => handleRoleChange('instructor')}
            />
            Instructor
          </label>
        </div>

        <input
          type="number"
          name="wallet"
          placeholder="Initial Wallet Amount"
          value={formData.wallet}
          onChange={handleChange}
        /><br/>

        <button type="submit">Register</button>
      </form>
      {message && <p>{message}</p>}
    </div>
  );
};

export default Register;
```