E-Learning Platform Development with Django and Docker

Kazakh-British Technical University

Murat Tinal

23MD0442

22.12.24

1. Executive Summary

The project creating an e-learning platform with Django and Docker was focused on building a flexible and scalable online learning platform. It used Django for backend development and Docker to create separate containers for the app. The platform includes essential features like user sign-up, course enrollment, managing lessons, quizzes, and payments. The Django Rest Framework (DRF) helped build an API to make the backend and frontend work smoothly together. Docker made sure the platform could be developed, tested, and deployed easily in different environments. A mock dataset was used to simulate real-life data and ensure the platform worked well for users. The main outcomes were the successful use of DRF for APIs, Docker for containers, and an efficient e-learning system. This platform can be expanded in the future with new features like mobile support, better analytics, and more integrations with other services.

2. Table of Contents

3. Introduction

Education is moving online, making learning accessible anytime, anywhere. E-learning connects students, teachers, and resources in one platform, creating an interactive experience. This project aims to build an e-learning platform using Django, a popular web framework. Django's tools handle user management, data, and flexible APIs. Docker ensures easy setup, scalability, and deployment.

Key features include user accounts, courses, lessons, reviews, ratings, progress tracking, and tests. These tools support knowledge sharing and interaction between students and teachers. The project combines technology with practical learning, helping developers explore Django, Docker, and API integration while creating a scalable, user-friendly platform.

4. System Architecture

The system architecture of this project is based on the use of Django, which serves as the main framework for developing a web application, and Docker, which provides containerization and management of the development and deployment environment. The system is based on a structure divided into modules to provide flexibility, scalability, and easier code management.

The application consists of several key components. The course folder stores the main application files such as models, views, routes, and admin settings in Fig.1. These files are responsible for data management, query processing logic, and information display. The "static" directory contains static resources, which can include CSS, JavaScript, and images for the frontend. The main application, called eLearning, contains global project settings such as database configuration and high-level routes. There are also files here "settings.py ", "urls.py" and "wsgi.py", which are responsible for setting up and launching the project. The "photos/courses" folder stores images related to courses, such as covers and visual elements. The page templates are located in the temps folder and include HTML files for various user interfaces, including registration, authorization, user profile, course descriptions, and other key interface components.

Docker plays an important role in ensuring the stable operation of the system. The "Dockerfile" file defines instructions for creating a container, and "docker-compose.yml" allows you to manage a multi-container architecture by combining a Django application and an SQLite database.

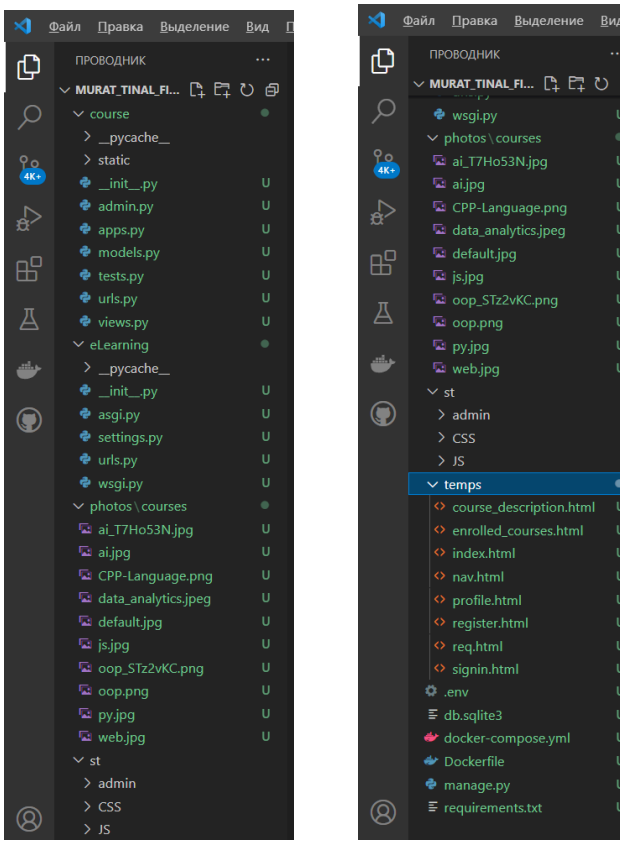


Fig.1

5. Dockerfile

This Dockerfile defines the setup for our Django-based e-learning application using Python 3.12-slim in *Fig.2* as the base image. It starts by installing essential tools like "curl" and building tools needed for compiling dependencies. The file also installs Rust and Cargo, which is required by some Python packages.

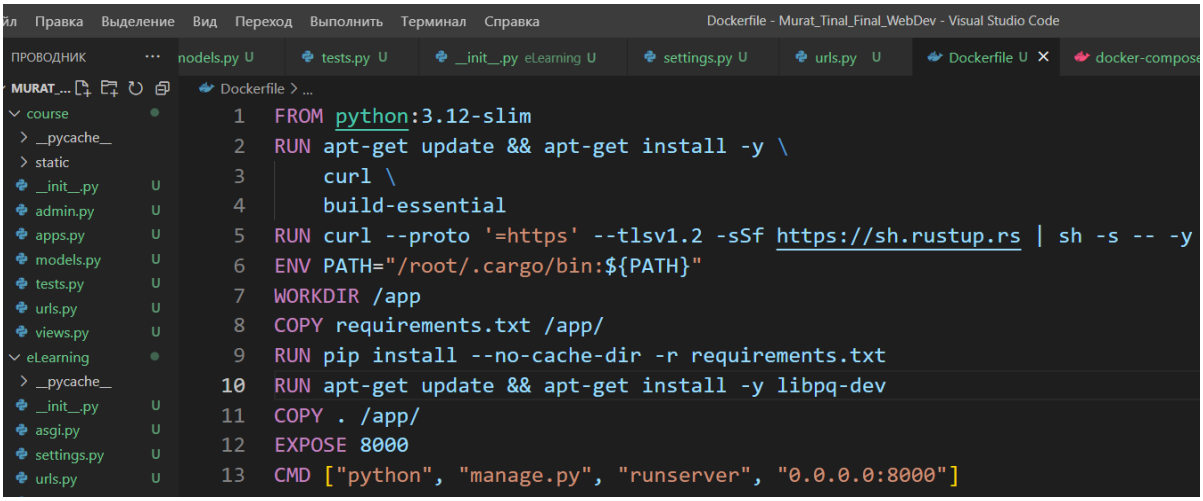


Fig.2

The application files are copied into the Docker container, and dependencies listed in "requirements.txt" are installed using "pip". The PostgreSQL development package "libpq-dev" is installed to ensure that the app can connect to the PostgreSQL database.

The application is set to run on port 8000, and the last line defines the command to start the Django development server inside the container, making it accessible from any address. This configuration allows the application to run smoothly inside Docker, with all dependencies packaged together.

6. Docker-Compose

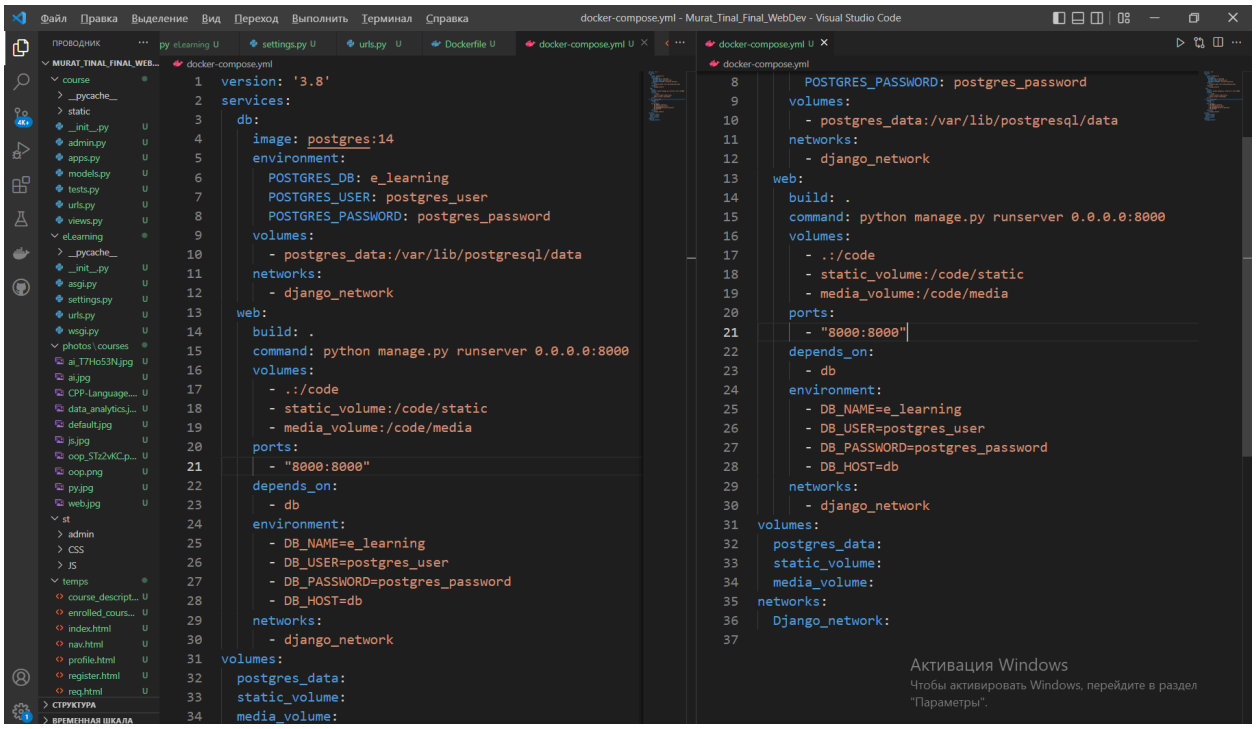


Fig.3

In *Fig.3*, I created a "docker-compose.yml" file to set up the task management application with a PostgreSQL database. The file defines two services: "db" for PostgreSQL and "web" for the Django application.

The "db" service uses the "postgres:14" image. I set environment variables to configure the database:

1) "POSTGRES_DB": Specifies the name of the database, which is "e_learning".
2) "POSTGRES_USER": Sets the username for accessing the database, defined as "postgres_user".
3) "POSTGRES_PASSWORD": Sets the password for the database user, which is "postgres_password".

Additionally, the database service uses a volume named "postgres_data" to ensure data is stored persistently at "/var/lib/postgresql/data". This means that even if the container restarts or is removed, the database data remains intact.

The "web" service builds the Django application from the current directory using the Dockerfile. I specified the command to run the Django development server. The web service also

mounts volumes for the code ".", static files "static_volume", and media files "media_volume" to facilitate real-time changes during development.

7. **Docker Networking and Volumes**

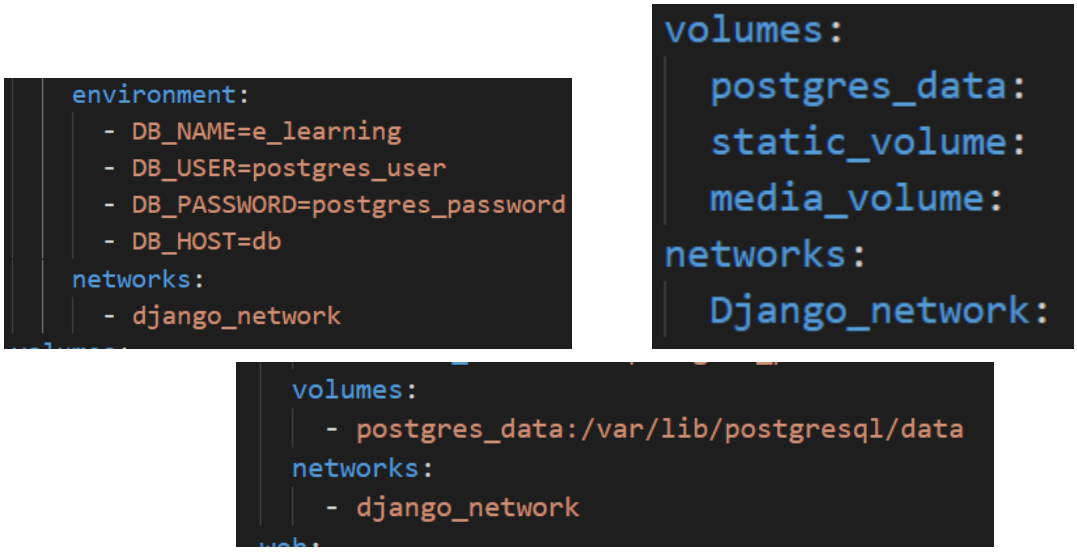


Fig.4

Docker volumes in *Fig.4*, "postgres_data", "static_volume", and "media_volume", help keep data safe even if a container stops or is recreated. For example, "postgres_data" holds the database, so no data is lost if the database container restarts.

To keep data safe when containers restart, we use volumes. For example, the postgres_data volume stores the database data, so it's not lost if the PostgreSQL container stops or gets recreated. This volume is linked to the /var/lib/postgresql/data folder, where PostgreSQL keeps its data. The static_volume and media_volume work the same way, storing static files and media files that users upload. These volumes are outside of the container, which means the data stays safe even if the container is recreated. The static_volume is connected to the /code/static folder, and the media_volume is connected to /code/media, making sure the app can always access these files.

Networks help the containers talk to each other. The django_network allows the web service and database containers to communicate easily, which is important for everything to work properly and keep the app running smoothly.

8. Django Models

The "SystemAdmin" class in ***Fig.5*** represents the system administrator who manages the platform. It includes fields such as "name", "number", "email" and "address" to store personal information about the administrator. The "username" and "password" fields are unique for each administrator, which ensures secure access. The "__str__" method is used to display the administrator's name when accessing this object. This class helps to manage the administrator's

data and ensures that only authorized administrators can log in. This is a simple model designed to store important information about the top managers of the system.

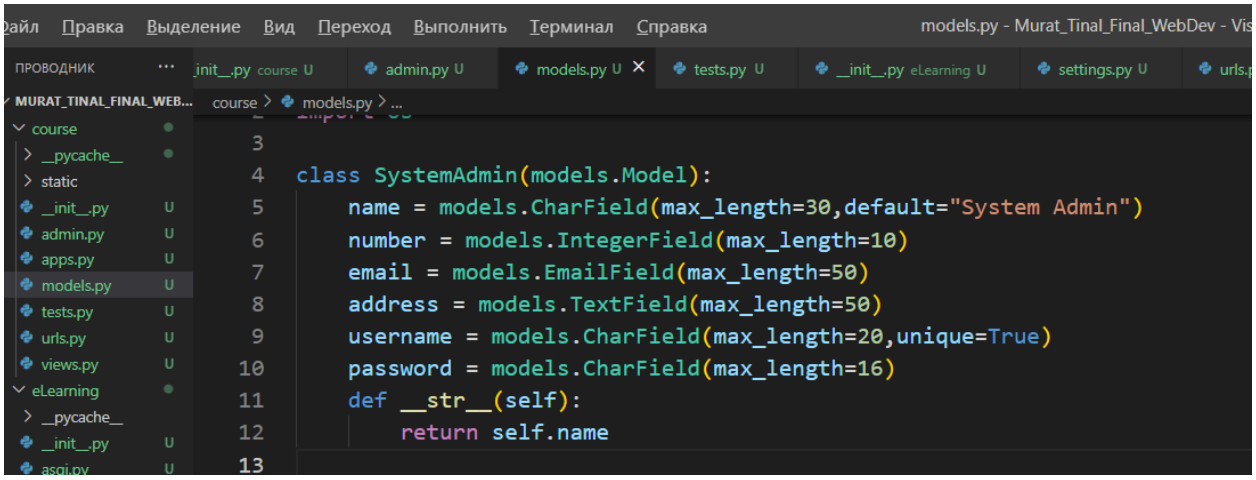


Fig.5

Fig.6

The "Course" class in ***Fig.6*** defines the courses available on the platform. It has fields like "course_name" for the name, "category" for classifying the type of course, and "description" for getting detailed information about the course. The "is_active" field indicates whether the course is currently available, and the "image_url" field stores an image representing the course. The categories are predefined, such as programming or data science. The class also includes logic for

deleting old course images when updating, making it easier to manage files.This class effectively organizes and manages course-related data.

```
class Student(models.Model):
    name = models.CharField(max_length=30,default="Student")
    number = models.IntegerField(max_length=10,blank=True,null=True)
    email = models.EmailField(max_length=50,blank=True)
    address = models.TextField(max_length=50,blank=True)
    username = models.CharField(max_length=20,unique=True,default="student")
    password = models.CharField(max_length=16,blank=True)
    enrolled_courses = models.ManyToManyField(Course,related_name='Student',blank=True)
    def __str__(self):
        return self.name
```

Fig.7

The "Student" class in ***Fig.7*** represents students using the platform. It includes fields such as "name", "number", "email address" and "email address" for personal information. The Username and Password fields ensure that each student has unique login credentials. Students can enroll in multiple courses using the "registered courses" field, which creates a link between students and the courses they attend. The "__str__" method shows the student's name when referring to the object. This class is necessary to manage information about students and their interaction with the courses available on the platform.

```
class CourseRequest(models.Model):
    request_id = models.AutoField(primary_key=True)
    course = models.ForeignKey(Course, on_delete=models.CASCADE,related_name='course_requests')
    student = models.ForeignKey(Student, on_delete=models.CASCADE,related_name='course_requests')
    request_date = models.DateField(auto_now=True)
    reason = models.TextField(max_length=500,blank=True)
    status = models.CharField(max_length=20, default="pending")

    class Meta:
        unique_together = ('course', 'student')
    def __str__(self):
        return f"{self.student.name} request for {self.course.course_name}"

```

Fig.8

The CourseRequest class tracks students' requests to enroll in certain courses. It has fields such as "request_id" for unique identification, "course" and "student" to link the request to the corresponding objects, as well as "request_date" to indicate the date when the request was made. The "reason" field allows students to explain why they want to take the course, and the "status" field indicates whether the request is under review, approved, or rejected. The "unique together" restriction ensures that a student cannot request the same course multiple times.This activity simplifies the process of managing course registration requests.

9. Django Views

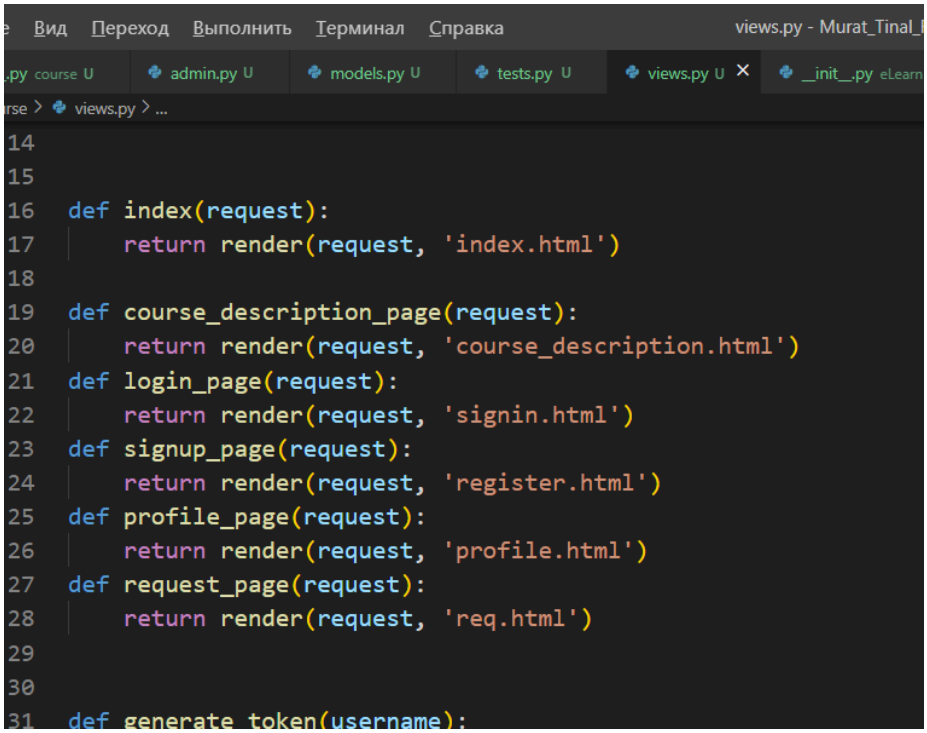


Fig.9

The function index(request) in ***Fig.9*** displays the main page of the application. When the user accesses this route, Django calls render() to process the request and returns an HTML index.html. The main page serves as a starting point where users can view general information about the educational platform or follow links to other pages. This feature is the basis of the user interface, providing easy access to the main functions of the application.

course_description_page is responsible for displaying the course description page. She uses a template course_description.html, which most likely contains textual and visual information about a particular course.

login_page loads the login page to the platform using a template signin.html. On this page, the user can enter their credentials (username and password).

signup_page is responsible for displaying the registration page by displaying a template register.html. It allows new users to create an account by entering fields such as name, email address, and password.

```python
def login_view(request):
    if request.method=='POST':
        username = request.POST['username']
        password = request.POST['password']
        print(username,password)
        user = Student.objects.filter(username=username,password=password)
        if user.exists():
            token = generate_token(username)
            return JsonResponse({"status": "success", "token": token})
        else:
            return JsonResponse({"status": "your password or username is incorrect"})
```

Fig.10

login_view verifies the credentials sent via the POST request, and if they are correct, generates a JWT token. If the user is found, a token is created, which is returned to the client in JSON format. If the user does not exist, an error message is returned. The function demonstrates the basic authentication implementation. This can be improved by adding limits on the number of login attempts or temporarily locking the account after several failures.

```python
def logout_view(request):
    if request.method=='POST':
        logout(request)
        return JsonResponse({"status": "success"})



def enrolled_courses_page(request):
    return render(request, 'enrolled_courses.html')

def all_courses(request):
    courses = Course.objects.filter(is_active=True)
    course_data = []
    for course in courses:
        course_dict = {
            "course_id": course.pk,
            "course_name": course.course_name,
            "category": course.category,
            "description": course.description,
            'is_active': course.is_active,
            "image_url": course.image_url.url
        }
        course_data.append(course_dict)

    for course in course_data:
        course["image_url"] = request.build_absolute_uri(course["image_url"])

    return JsonResponse({"courses": course_data})
```

Fig.11

```python
def student_info(request):
    if request.method=='POST':
        auth_header = request.headers.get('Authorization')
        token = auth_header.split(" ")[1]
        try:
            payload = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])
            username = payload["username"]
            print(username)
            student = Student.objects.get(username=username)
            course=get_enrolled_courses(username)
            serialized_student = serializers.serialize('json', [student])
            student=json.loads(serialized_student)
            student_data=student[0]['fields']
            return JsonResponse({"student": student_data, "enrolled_courses": course})
        except Exception as e:
            return JsonResponse({"status": "student not found"})

```

Fig.12

The student_info function uses a JWT to authenticate a user and retrieve their information and enrolled courses. It returns the student's details and courses in JSON format shown ***Fig.12***.

```python
def signup(request):
    if request.method=='POST':
        name=request.POST['name']
        number=request.POST['number']
        email=request.POST['email']
        address=request.POST['address']
        username=request.POST['username']
        password=request.POST['password']
        student = Student(name=name,number=number,email=email,address=address,username=username,password=password)
        try:
            student_exist = Student.objects.filter(username=username)
            if student_exist:
                return JsonResponse({"status": "student already exist"})
            else:
                student.save()
                token=generate_token(username)
                return JsonResponse({"status": "success", "token": token})
        except Exception as e:
            return JsonResponse({"status": e})
```

Fig.13

The signup function handles new user registration by saving the student's details in the database. It checks for username duplicates and generates a JWT upon successful registration shown in ***Fig.13***.

```python
def get_course(request):
    if request.method=='GET':
        course_id = request.GET['course_id']

        print(course_id)
        course = Course.objects.get(course_id=course_id)
        course_data = {
            "course_id": course.pk,
            "course_name": course.course_name,
            "category": course.category,
            "description": course.description,
            "image_url": course.image_url.url
        }
        course_data["image_url"] = request.build_absolute_uri(course_data["image_url"])

        return JsonResponse({"course": course_data})
```

Fig.14

The get_course function in ***Fig.14*** is used to get data about a specific course by its ID. It only processes GET requests by extracting the course_id parameter from the request. After receiving the ID, the function searches for the corresponding Course object in the database using the get() method of the Course model. If a course is found, data from its fields is collected in the course_data dictionary, which includes key attributes such as the course ID, its name, category, description, and image URL. A special feature of the function is the conversion of the relative image path to an absolute URL via the request method.build_absolute_uri(). This is important for the correct display of images on client devices.

10. Templates

\temps\index.html: The first section, entitled "About App" in ***Fig.15***, provides a brief description of the e-learning platform. It displays the name of the project, and text explaining the purpose of developing the platform, including the use of Django, Docker, and other technologies. The entire block is designed using embedded CSS styles.

The second section is devoted to courses. It has a heading "Courses" and a container that is initially empty but is designed to dynamically add information about courses. A JavaScript file JS/script.js using the tag {%static %}.

The third section contains an important note for users: only registered users can subscribe to the courses. A list and stylized text for explanations have been added to it.

```
<section id="about">
    <div class="container">
        <img src="Me.jpg" alt="About Us" style="width: 400px;height: 300px;">
        <h1 style="color: chocolate">E-Learning Platform(by Murat Tinal)</h1>
        <div class="section-content">
            <p style="background: orange;">The goal of this project is to develop a robust e-learning platform using
            </p>
        </div>
    </div>
</section>

<section id="courses">
    <div class="container">
        <h1 style="color: orange;">Courses</h1>
        <div id="course-container" class="w-100">
            <!-- Courses will be dynamically added here -->
            <script src="{% static 'JS/script.js' %}"></script>
        </div>
    </div>
</section>

<section id="subscribe">
    <div class="container">
        <h1></h1>
        <div class="section-content">
            <ul style="list-style-type: none; padding-left: 0;">
                <li><span style="font-size: 20px;"></span></li>
                <li><span style="font-size: 20px;"></span></li>
                <li><span style="font-size: 20px;"></span></li>
                <li><span style="font-size: 20px;"></span></li>
                <li><span style="font-size: 20px;"></span></li>
                <li><span style="font-size: 20px;"></span></li>
            </ul>
            <p style="color: red;">Note that you can't request a course if you are not signed in.</p>
        </div>
    </div>
</section>
{% endblock %}
```

Активация
Чтобы активир
"Параметры".

Fig.15

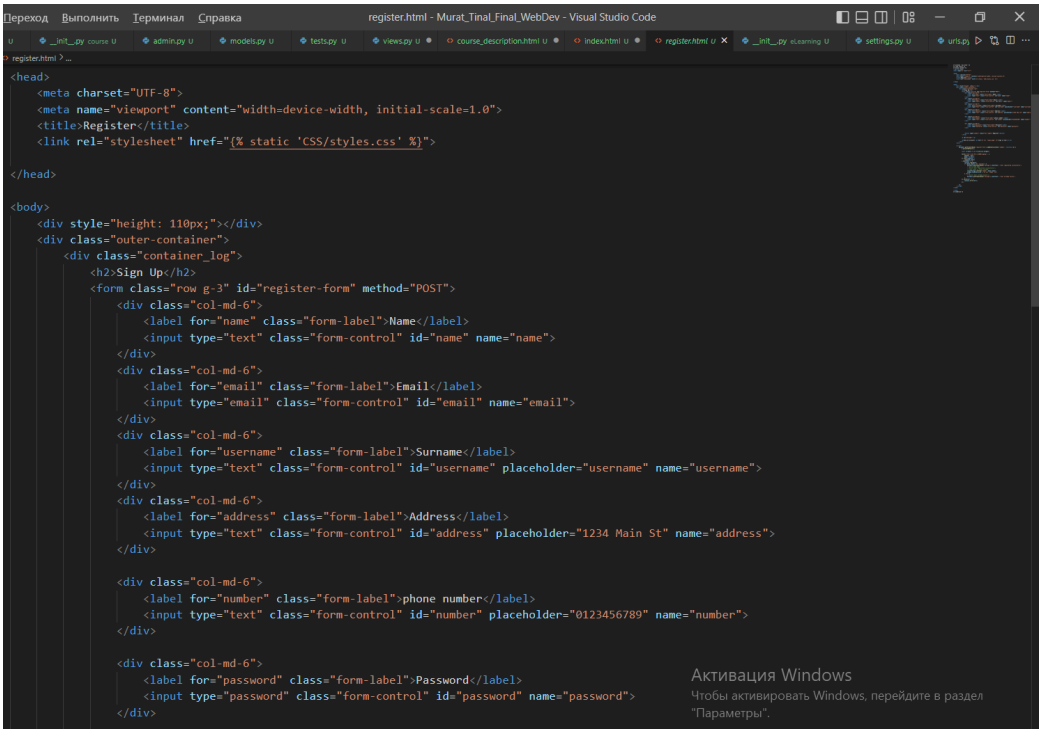


Fig.16

The form sends data using the POST method. However, instead of sending it to the server as standard, Js is used to prevent the page from reloading. A submission event is processed, data is collected using form data, and then sent to the server using the fetch method.

If registration is successful, the user is shown a success message, the token is saved in localStorage, and the user is redirected to the main page (the URL is set using {% url index %}). If the user already exists, an error message is displayed.

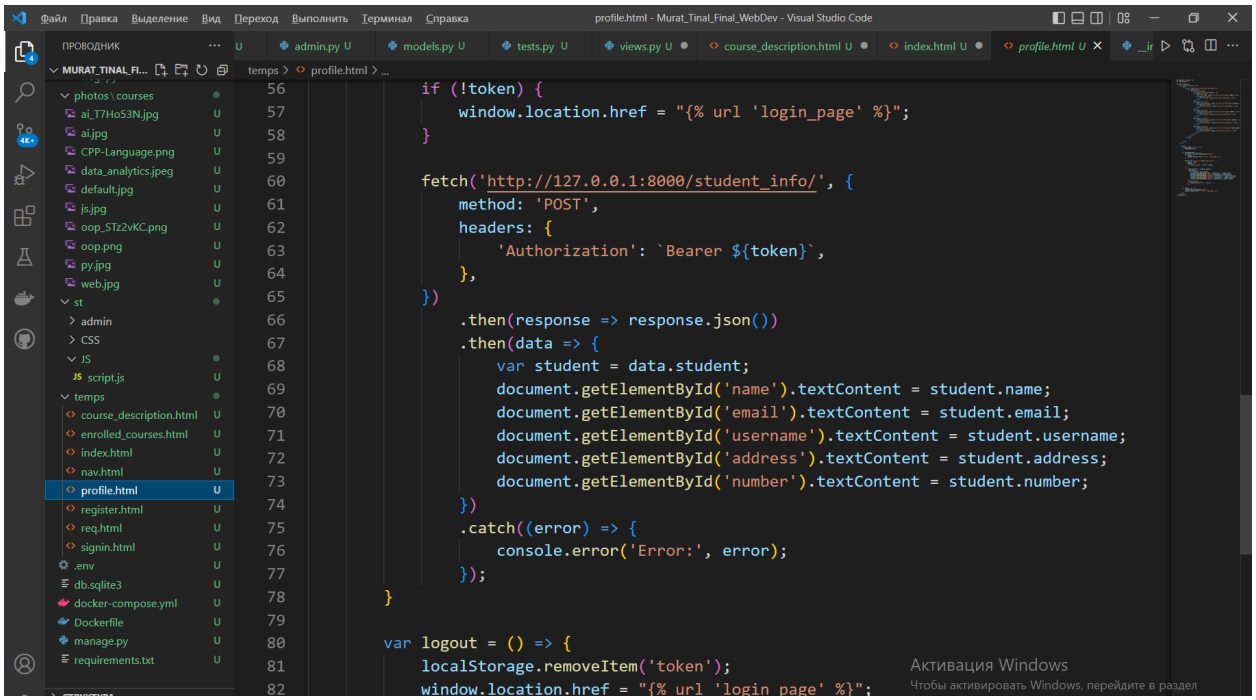


Fig.17

11. Django Rest Framework (DRF)

In ***Fig.18*** sets routes for the Django web application, and using DRF involves processing API requests using routes that allow interaction with application data. DRF provides convenient integration to create a RESTful API where I worked with resources such as courses and user requests.

Functions get_course_request in ***Fig.19*** implemented using DRF to process API requests. For example, instead of simple views, I used the View API to return data in JSON format, which is often required for client-server interaction. DRF helps manage data serialization, query processing, and the addition of authentication for the API.

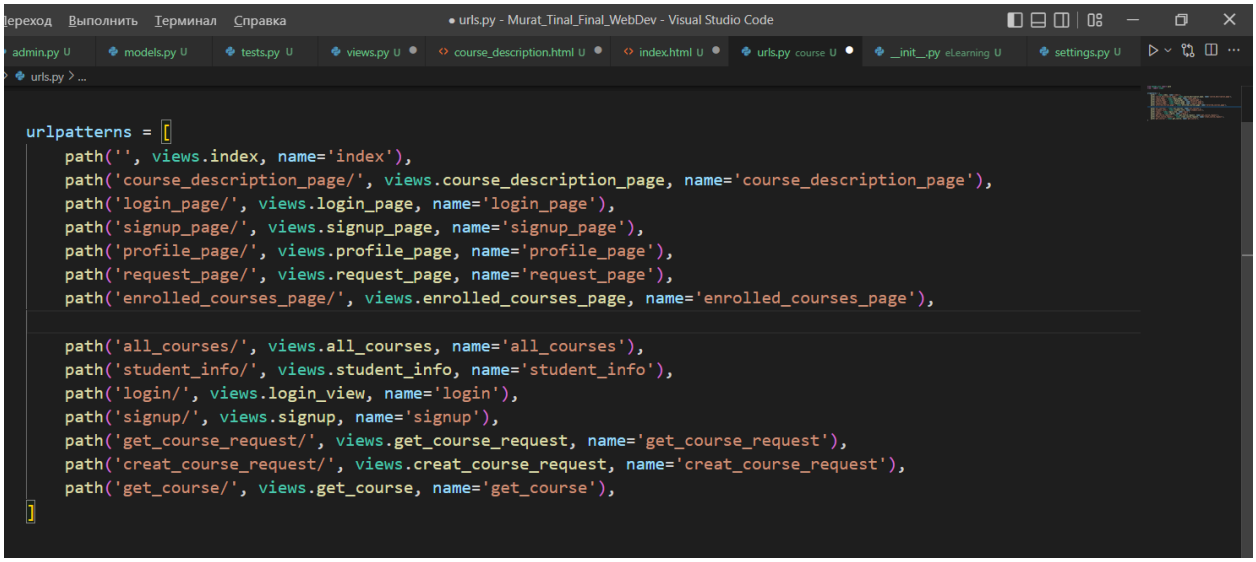


Fig.18

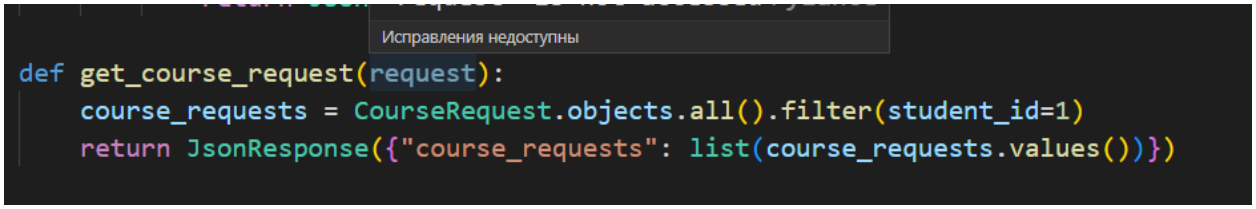


Fig.19

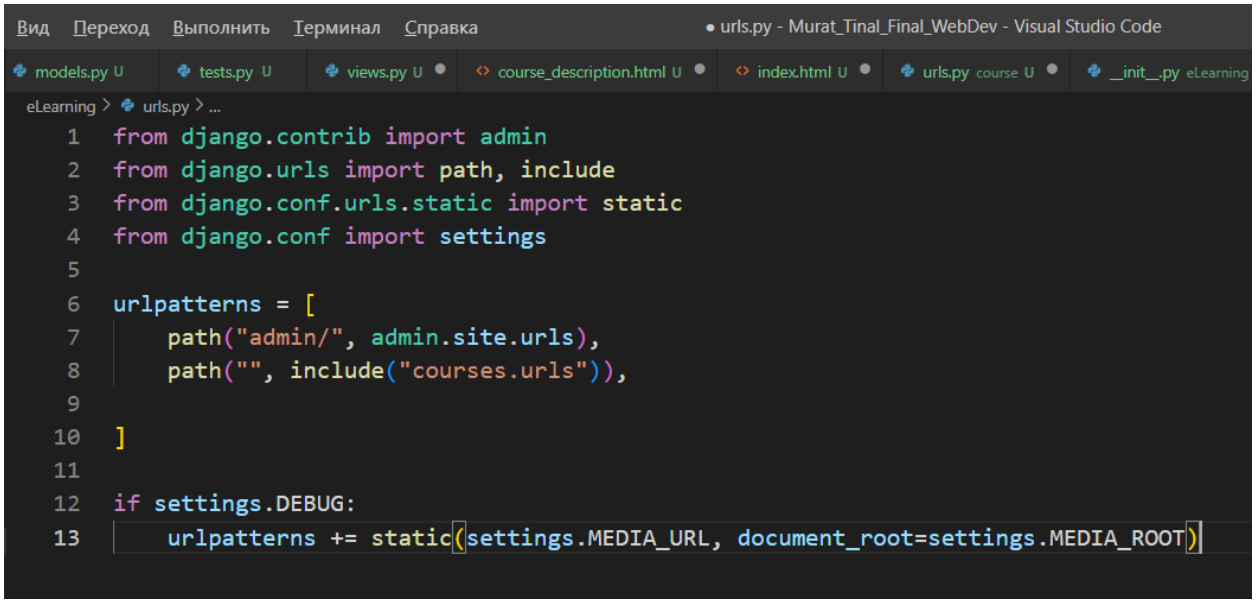


Fig.20

```python
def student_info(request):
    if request.method=='POST':
        auth_header = request.headers.get('Authorization')
        token = auth_header.split(" ")[1]
        try:
            payload = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])
            username = payload["username"]
            print(username)
            student = Student.objects.get(username=username)
            course=get_enrolled_courses(username)
            serialized_student = serializers.serialize('json', [student])
            student=json.loads(serialized_student)
            student_data=student[0]['fields']
            return JsonResponse({"student": student_data, "enrolled_courses": course})
        except Exception as e:
            return JsonResponse({"status": "student not found"})

```

Fig.21

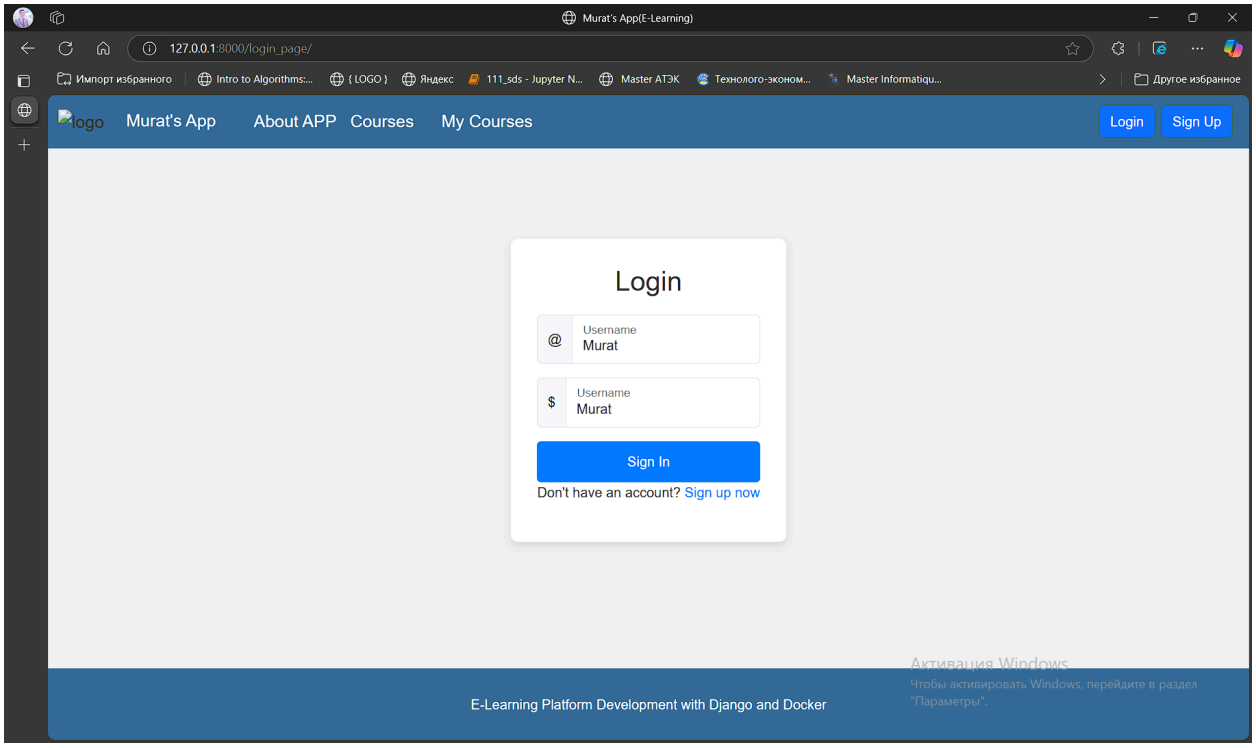
Fig.21 represents a handler for getting information about a student and his courses through a request processed using DRF. Here you can see the principle of interaction with authentication tokens, which is often used in APIs created using the Django Rest Framework. The incoming request checks the POST method and uses the Authorization header to extract the JWT token. DRF is often used to work with tokens due to the built-in authentication tools.

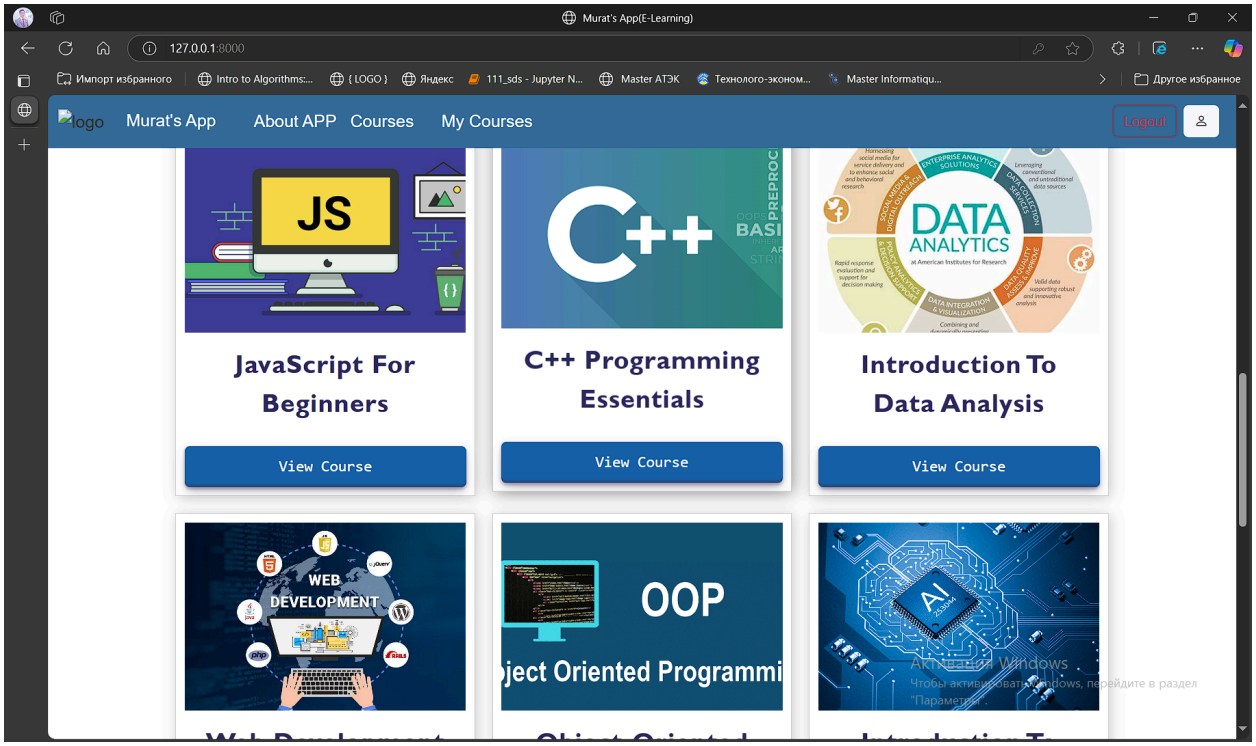
The student's object is then extracted from the database, and the data is serialized to turn it into a JSON format. DRF provides a powerful serialization engine that could be improved here by using a ModelSerializer instead of manual serialization. The function returns student data and a list of his courses in JSON format, which is ideal for the REST API. In case of an error, the "student not found" status is returned, which can also be standardized using DRF.

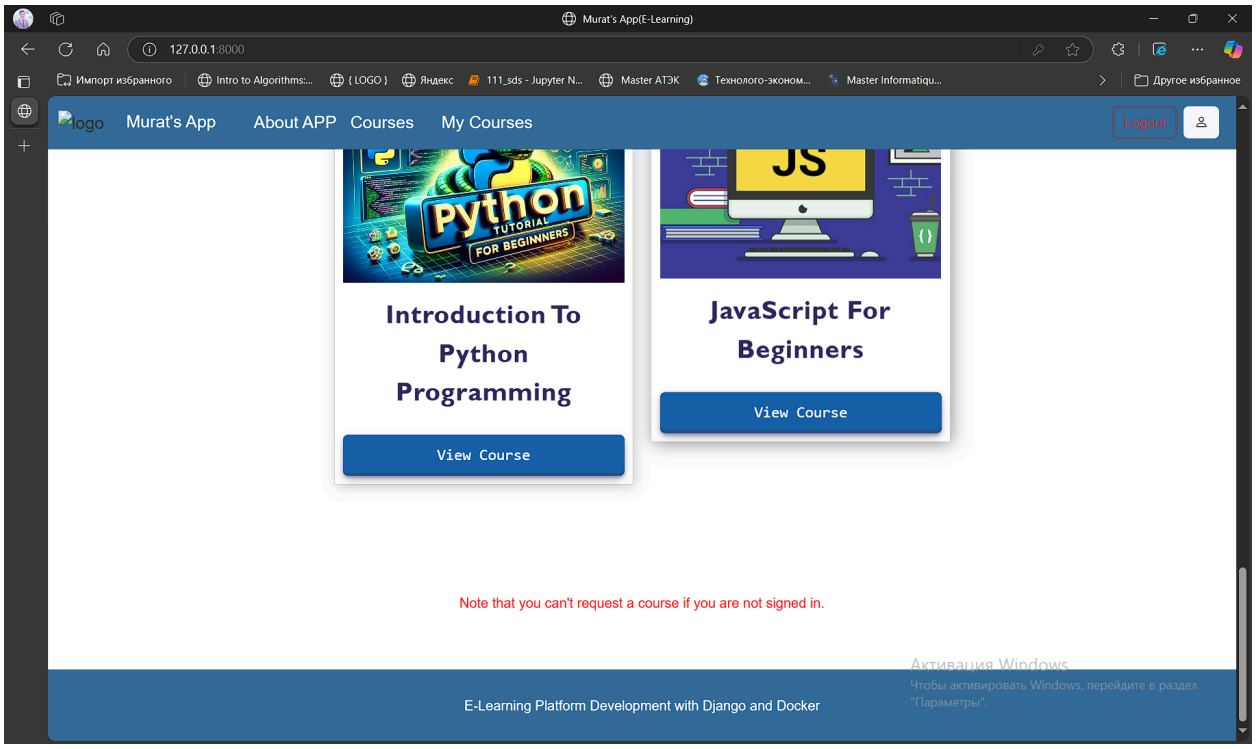
12. Conclusion

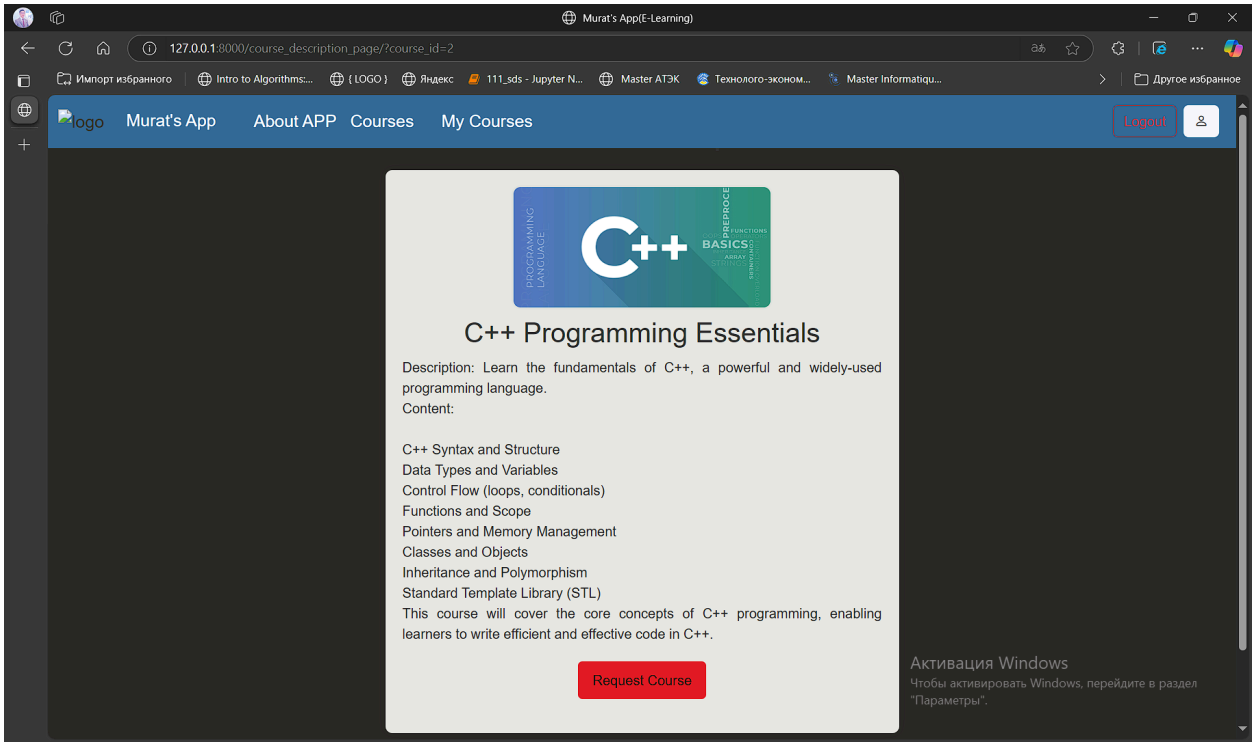
In conclusion, this project demonstrated the effectiveness of combining Django, Docker, and DRF in developing a full-fledged e-learning platform. The containerization provided a stable and scalable environment, while Django's powerful features enabled smooth development of the backend. DRAFT allowed for the creation of robust APIs, enhancing system interaction with the front end. Challenges such as Docker networking and data persistence were overcome through careful configuration. Overall, the platform is scalable, maintainable, and can be easily extended with new features. Future improvements may include enhanced security, user interface upgrades, and further optimization for large-scale usage.

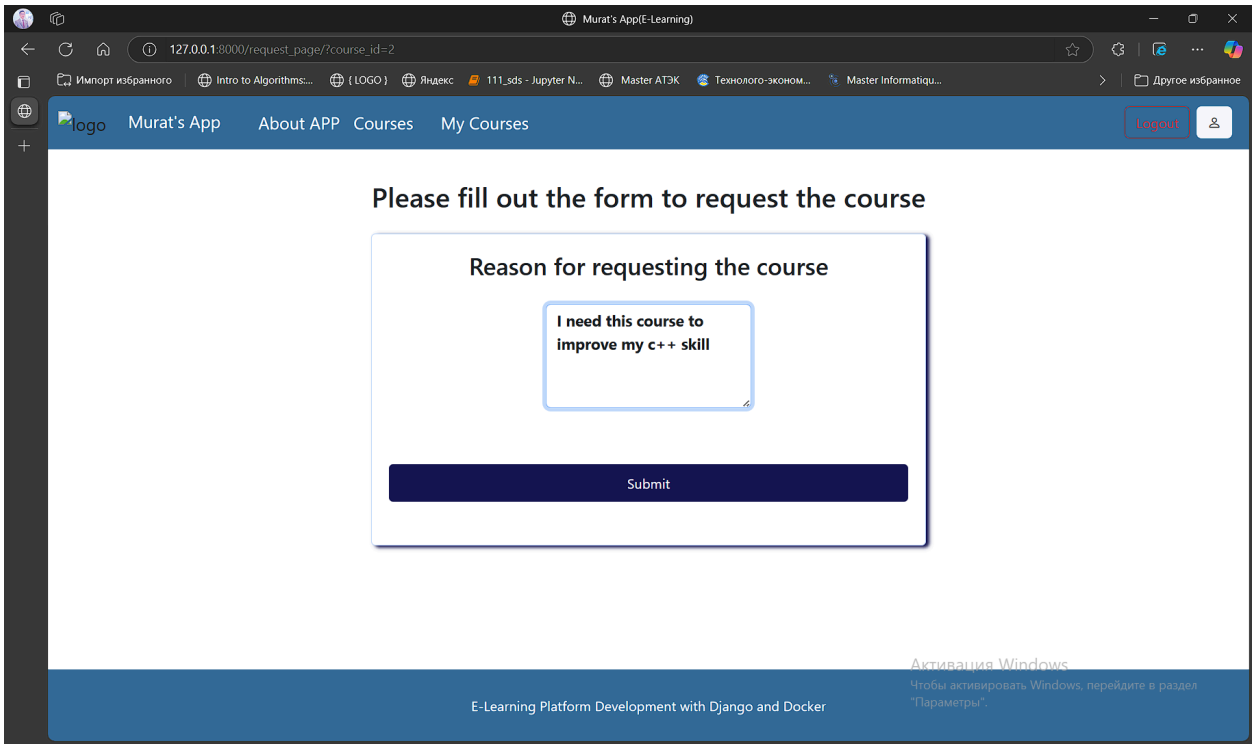
Here you can see my screenshots of my project (GitHub:
https://github.com/kiraakz/Murat_Tinal_Final_WebDev

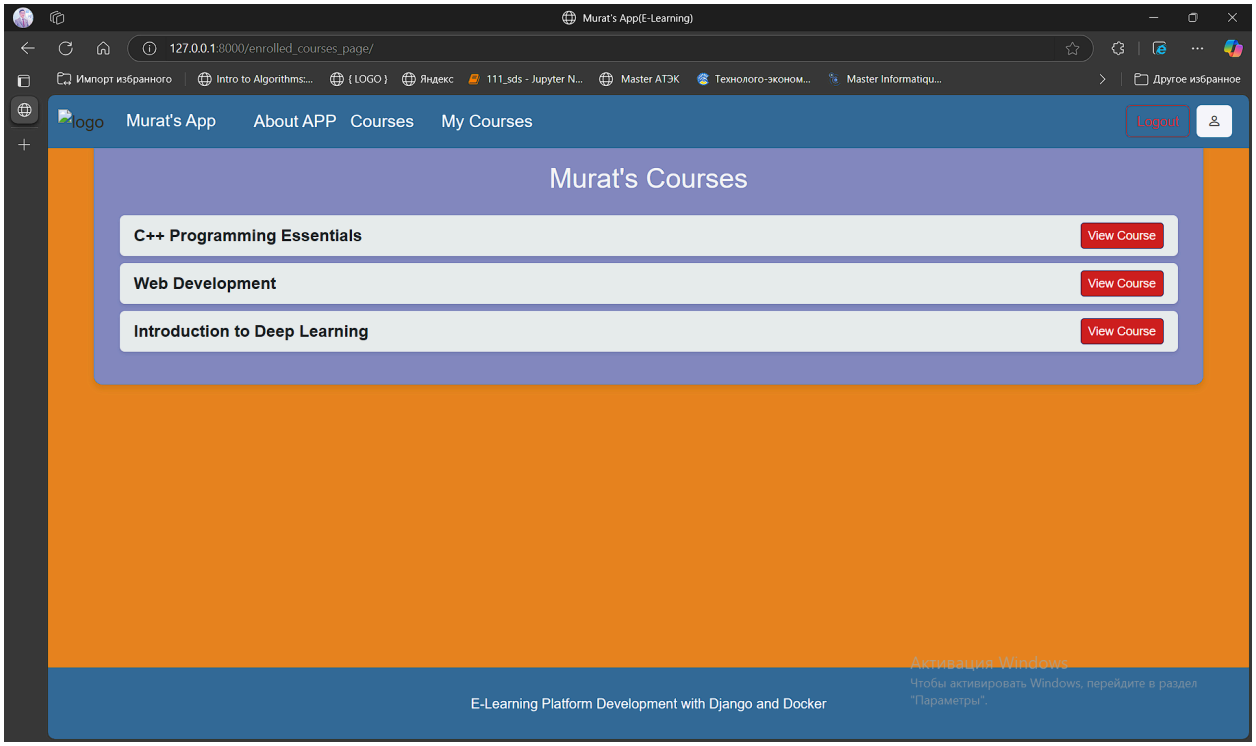


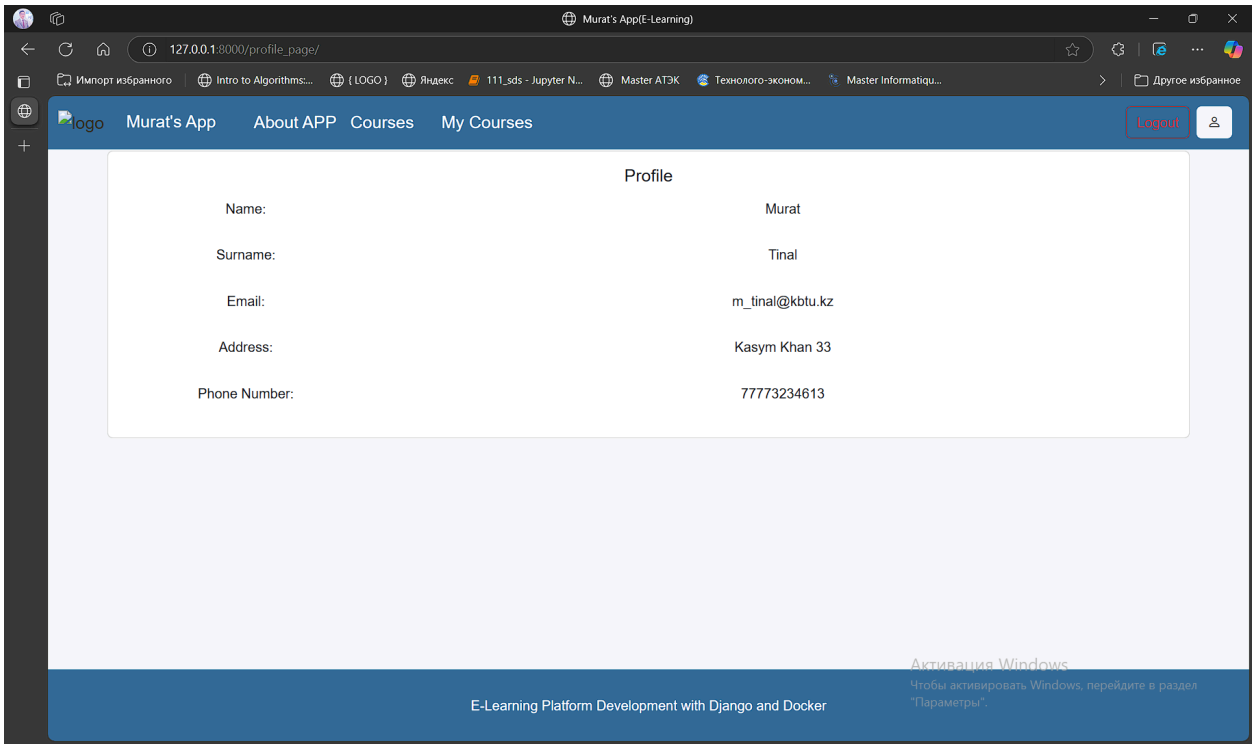

Murat's App(E-Learning)
127.0.0.1:8000
logo
Murat's App
About APP
Courses
My Courses
Logout
JS
JavaScript For Beginners
View Course
C++
C++ Programming Essentials
View Course
DATA ANALYTICS
Introduction To Data Analysis
View Course
WEB DEVELOPMENT
OOP
AI

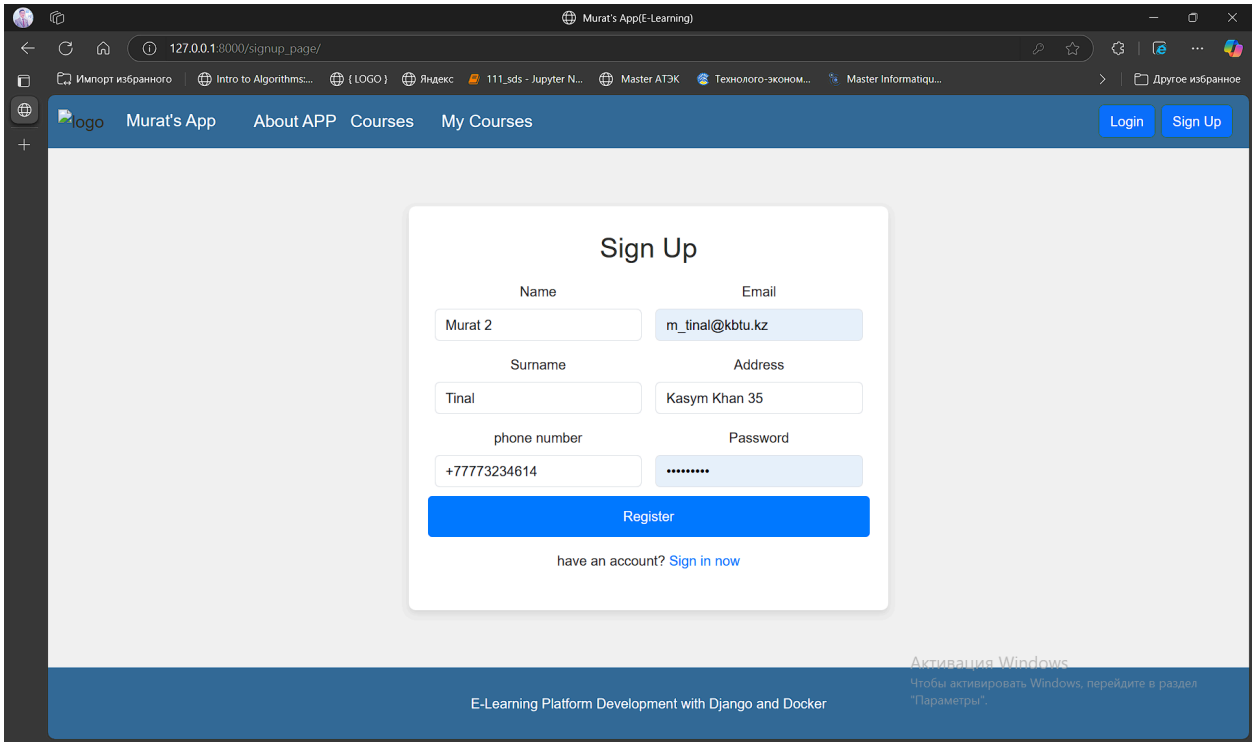

Murat's App(E-Learning)
127.0.0.1:8000
logo
Murat's App
About APP
Courses
My Courses
Logout
Python TUTORIAL FOR BEGINNERS
Introduction To Python Programming
View Course
JS
JavaScript For Beginners
View Course
Note that you can't request a course if you are not signed in.
E-Learning Platform Development with Django and Docker


Murat's App(E-Learning)
127.0.0.1:8000/course_description_page/?course_id=2
logo
Murat's App
About APP
Courses
My Courses
Logout
C++
BASICS
C++ Programming Essentials
Description: Learn the fundamentals of C++, a powerful and widely-used programming language.
Content:
C++ Syntax and Structure
Data Types and Variables
Control Flow (loops, conditionals)
Functions and Scope
Pointers and Memory Management
Classes and Objects
Inheritance and Polymorphism
Standard Template Library (STL)
This course will cover the core concepts of C++ programming, enabling learners to write efficient and effective code in C++.
Request Course
Активация Windows


Murat's App(E-Learning)
127.0.0.1:8000/request_page/?course_id=2
logo
Murat's App
About APP
Courses
My Courses
Logout
Please fill out the form to request the course
Reason for requesting the course
I need this course to improve my c++ skill
Submit
E-Learning Platform Development with Django and Docker
Активация Windows


Murat's App(E-Learning)
127.0.0.1:8000/enrolled_courses_page/
logo
Murat's App
About APP
Courses
My Courses
Logout
Murat's Courses
C++ Programming Essentials
View Course
Web Development
View Course
Introduction to Deep Learning
View Course
E-Learning Platform Development with Django and Docker


Murat's App(E-Learning)
127.0.0.1:8000/profile_page/
logo
Murat's App
About APP
Courses
My Courses
Logout
Profile
Name:
Murat
Surname:
Tinal
Email:
m_tinal@kbtu.kz
Address:
Kasym Khan 33
Phone Number:
77773234613
E-Learning Platform Development with Django and Docker


Murat's App(E-Learning)
127.0.0.1:8000/signup_page/
Импорт избранного
Intro to Algorithms:...
{ LOGO }
Яндекс
111_sds - Jupyter N...
Master АТЭК
Технолого-эконом...
Master Informatiqu...
Другое избранное
logo
Murat's App
About APP
Courses
My Courses
Login
Sign Up
Sign Up
Name
Murat 2
Email
m_tinal@kbtu.kz
Surname
Tinal
Address
Kasym Khan 35
phone number
+77773234614
Password
Register
have an account? Sign in now
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".
E-Learning Platform Development with Django and Docker

15. References

- https://buttercms.com/blog/build-an-online-learning-platform-with-django/
- https://www.educative.io/projects/creating-an-e-learning-website-using-django
- https://python.plainenglish.io/how-i-built-an-online-learning-platform-with-django-in-30-days-udemy-clone-a384ba507cf7
- https://github.com/ahmedEid1/E-Learning-Platform
- https://github.com/ashish-makes/django-lms
- https://stackskills.com/p/django-projects-e-learning-portal
- https://medium.com/@emmanuelsibanda/building-an-e-learning-school-minimum-viable-product-on-django-cf81dc8280c6
- https://github.com/ShivamRohilllaa/E-learning-Django-
- https://www.youtube.com/watch?v=hEMkB5wngSc&list=PLTV1jAY3nKHMotVkWRcqH4EGPOqliQZ23
- https://www.youtube.com/watch?v=sREYiEsooew&list=PLeQ4zRlSjJn96nZJAlFVxjE6k9LEexyKU
- https://www.youtube.com/watch?v=3z0XtYwly4w&list=PLxQ9RfumYrxa86SoJkszzrCURi2Rw74eg
- https://www.youtube.com/watch?v=aK9Y-0AsbA4&list=PLSJxovi1IyDGkHNqlrPSU2kXu1aophIkG
- https://www.django-rest-framework.org/
- https://django.fun/articles/tutorials/kratko-o-django-rest-framework/