**Поволжский государственный университет телекоммуникаций и информатики**

**Отчёт по дисциплине «Разработка серверной части web-приложений»**

**Лабораторная работа №1.**

**Выполнил студент группы ИВТ-23**

**Ермишин Максим**

1.1Введение в Django

Ознакомиться с базовыми понятиями фреймворка Django, его структурой и принци- пами работы. Освоить настройку среды разработки. Создание проекта и маршрутизации.

Задания: Для работы с Django необходимо использовать среду разработки Python!

1. Установите Django и создайте новый проект

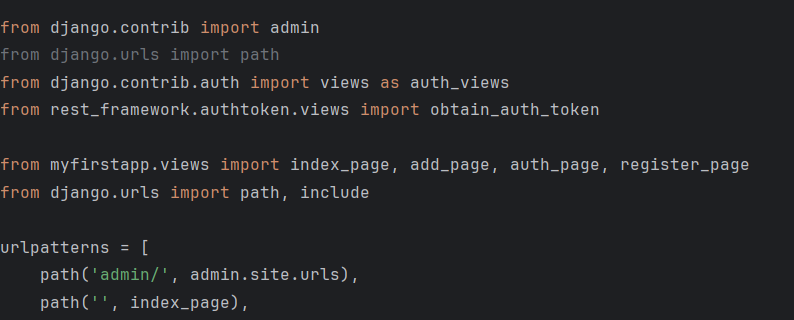
2. Настройте маршрутизацию для страницы с приветственным сообщением.

3. Настройте маршрутизацию для страницы с приветственным сообщением. Создайте базовый HTML-шаблон для вывода сообщения «Hello, world!».

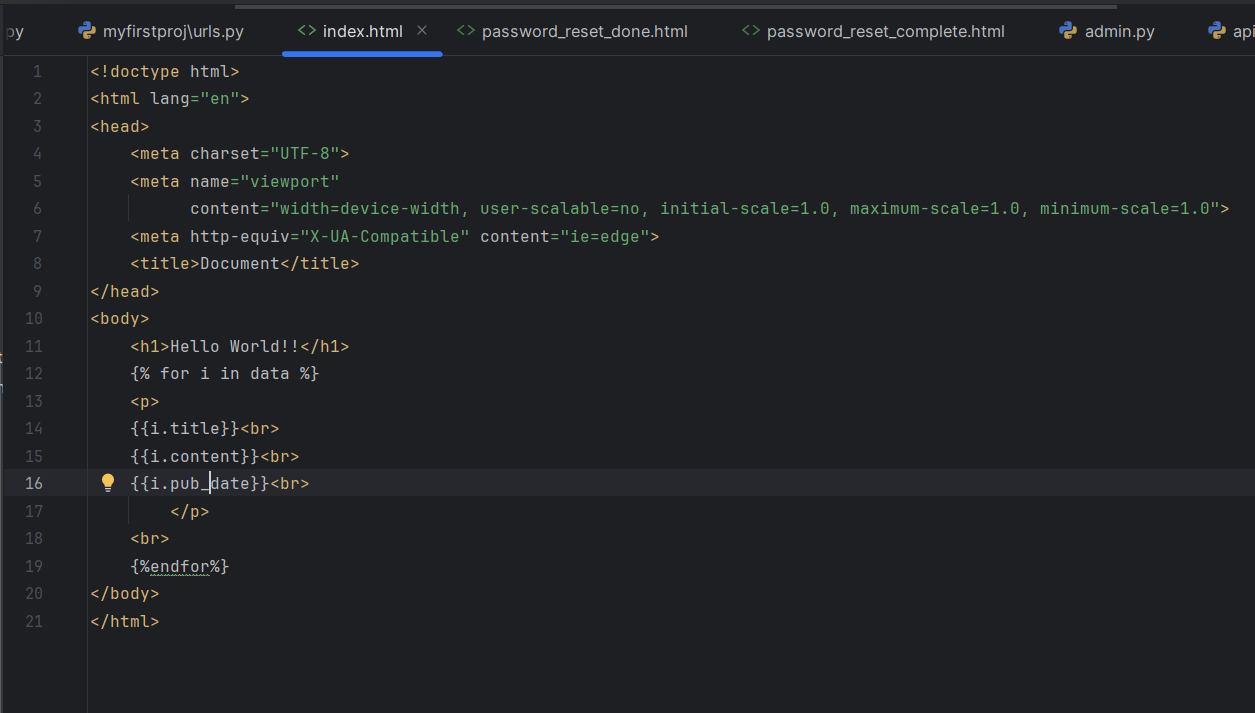
Страница с Hello World



Настройка пути к странице:



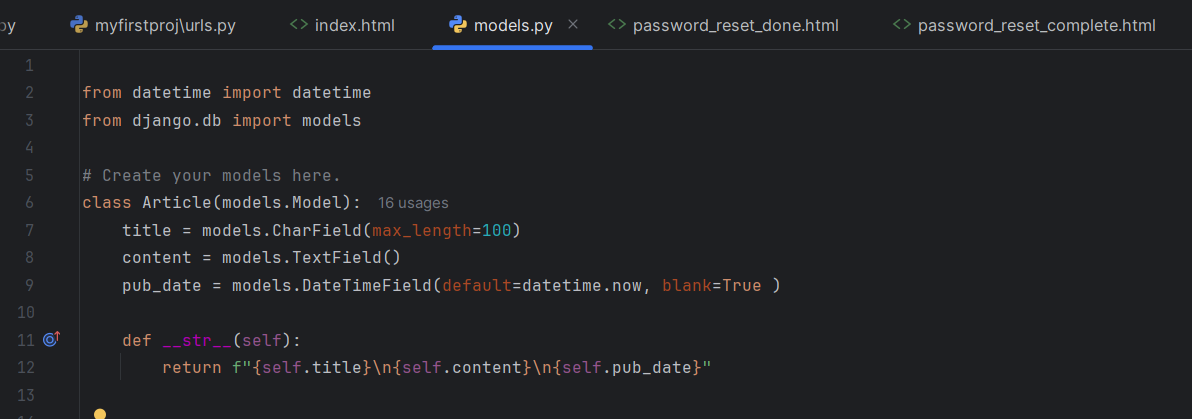
Шаблон страницы:



1.2 Модели и работа с базой данных

Цель работы: Научиться создавать модели в Django, использовать встроенную ORM для работы с базами данных и выполнять миграции. Задания:

1. Создайте модель «Article» с полями: заголовок, содержимое, дата публикации.

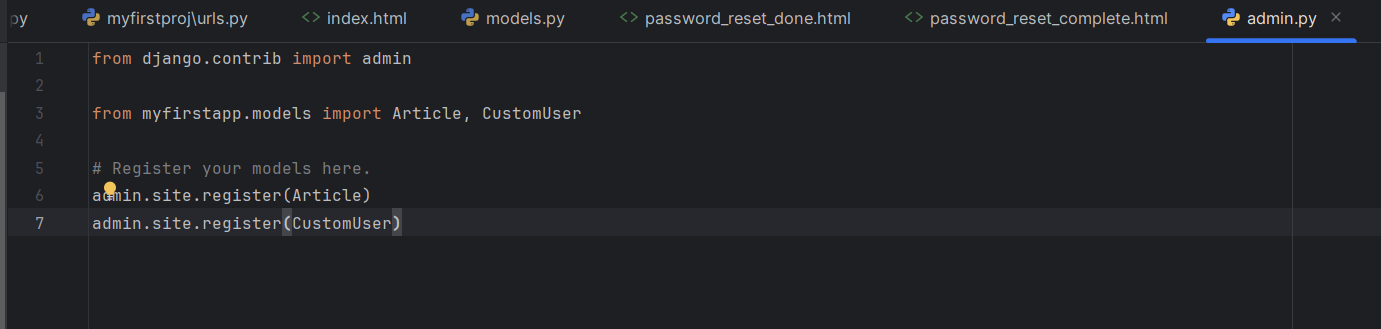


1. Настройте базу данных и выполните миграции.

python .\manage.py makemigrations

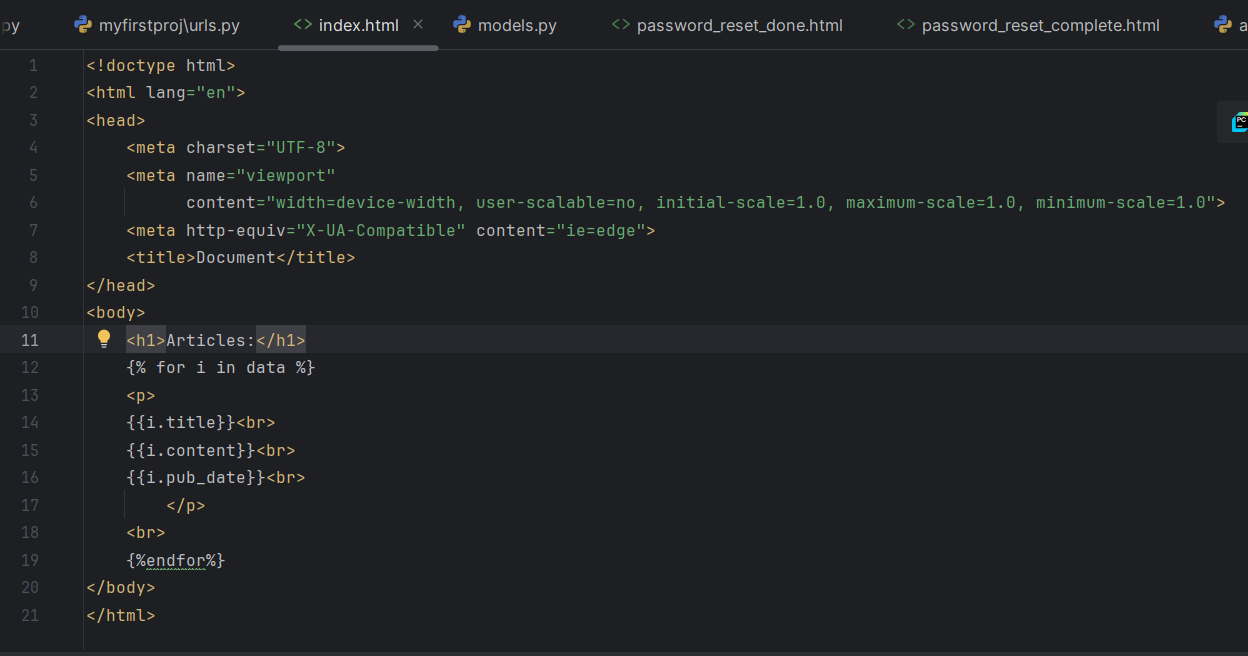
python .\manage.py migrate

1. Добавьте модель в Django Admin для управления записями.



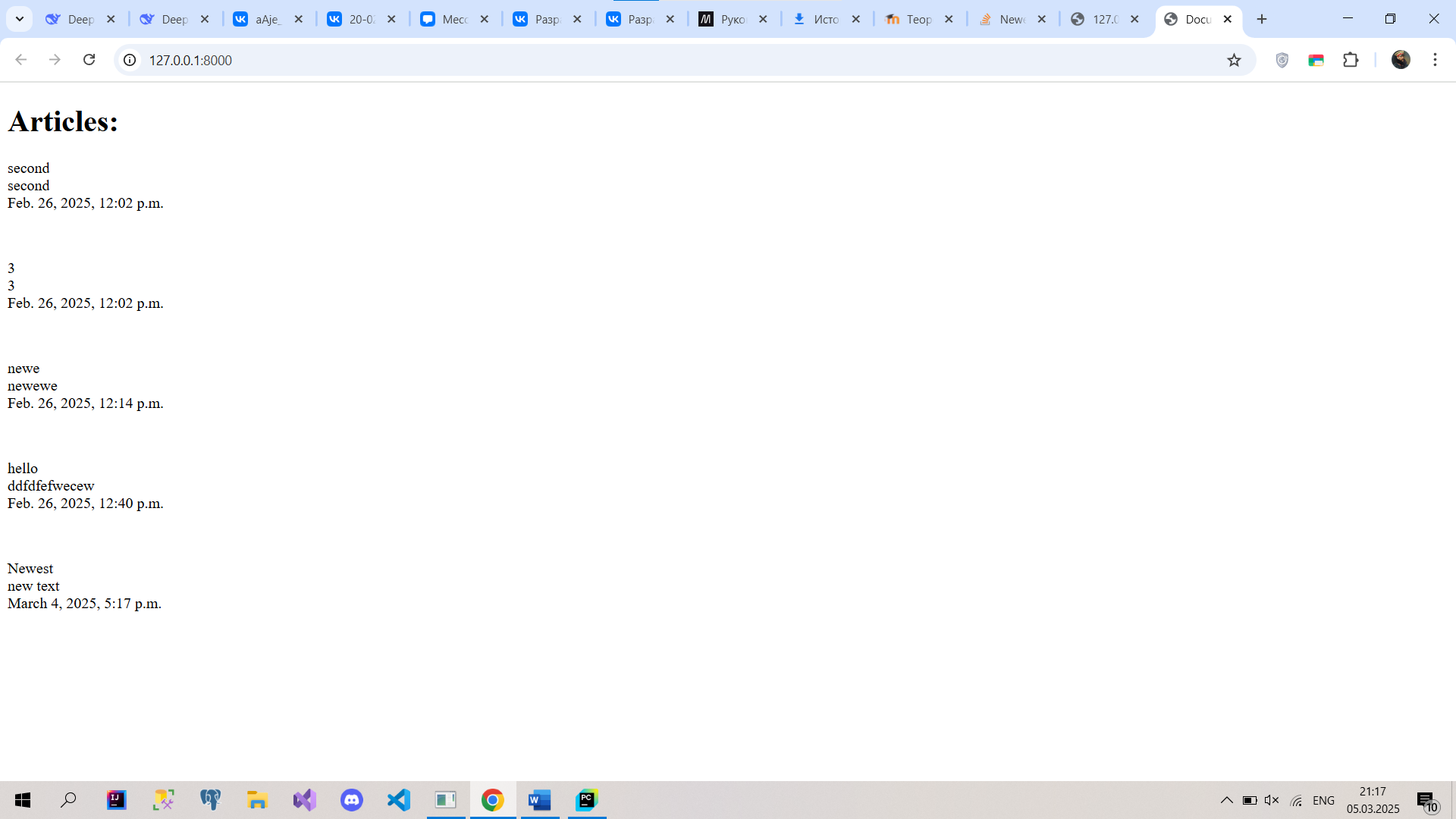
1. Реализуйте вывод всех статей на отдельной странице.

Template:



View:

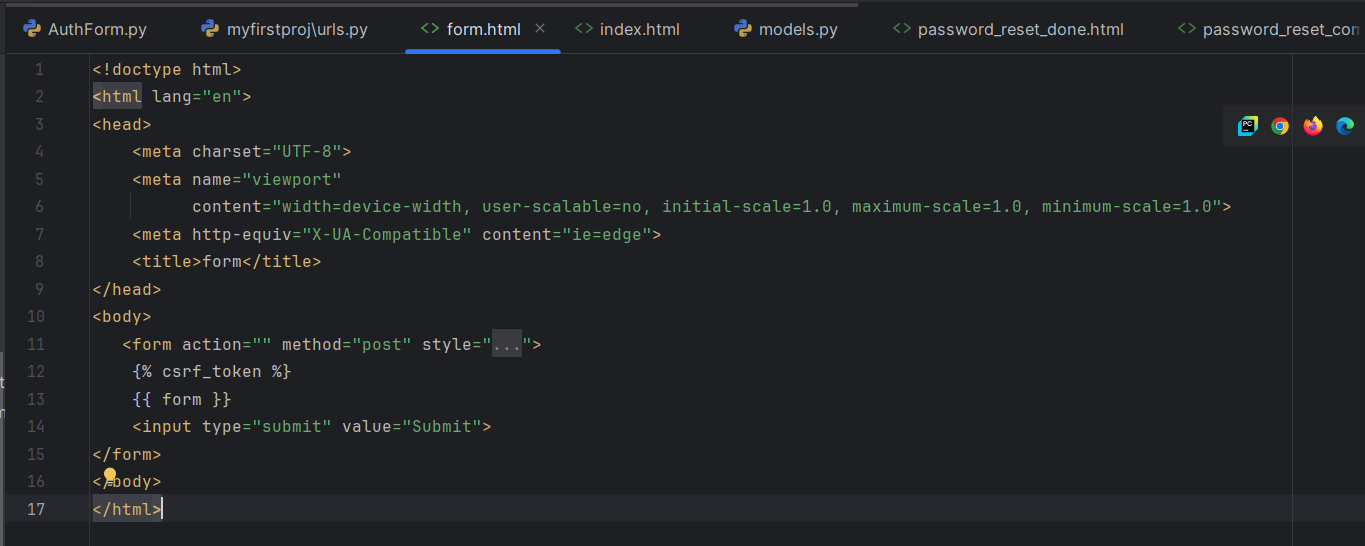




1.3 Формы и валидация данных лей).

Цель работы: Освоить работу с формами в Django, их создание, обработку и валидацию данных.

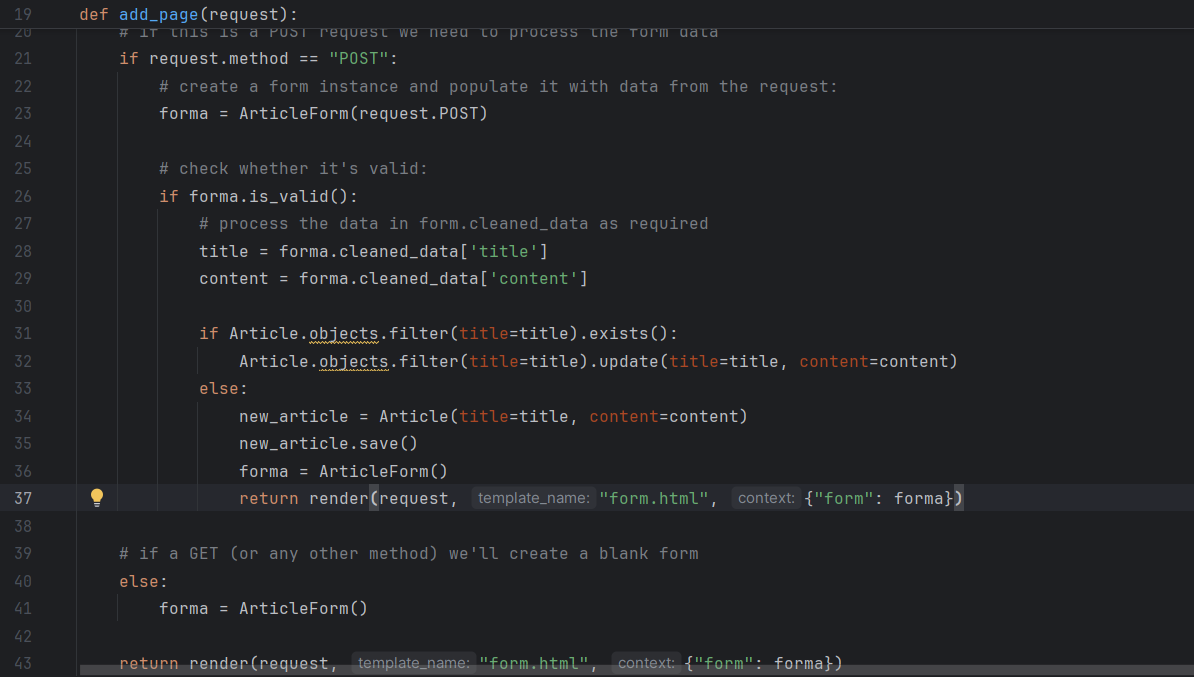
Задания: 1. Создайте форму для добавления новых статей (заголовок, содержимое).



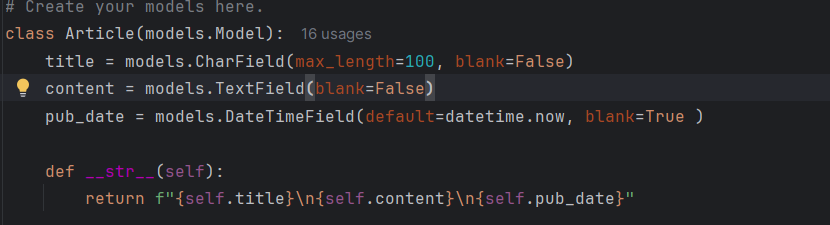


2. Реализуйте маршруты для отображения формы и обработки данных формы.

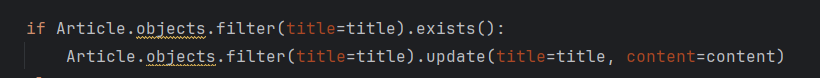




3. Настройте валидацию данных формы (например, проверка обязательных полей)



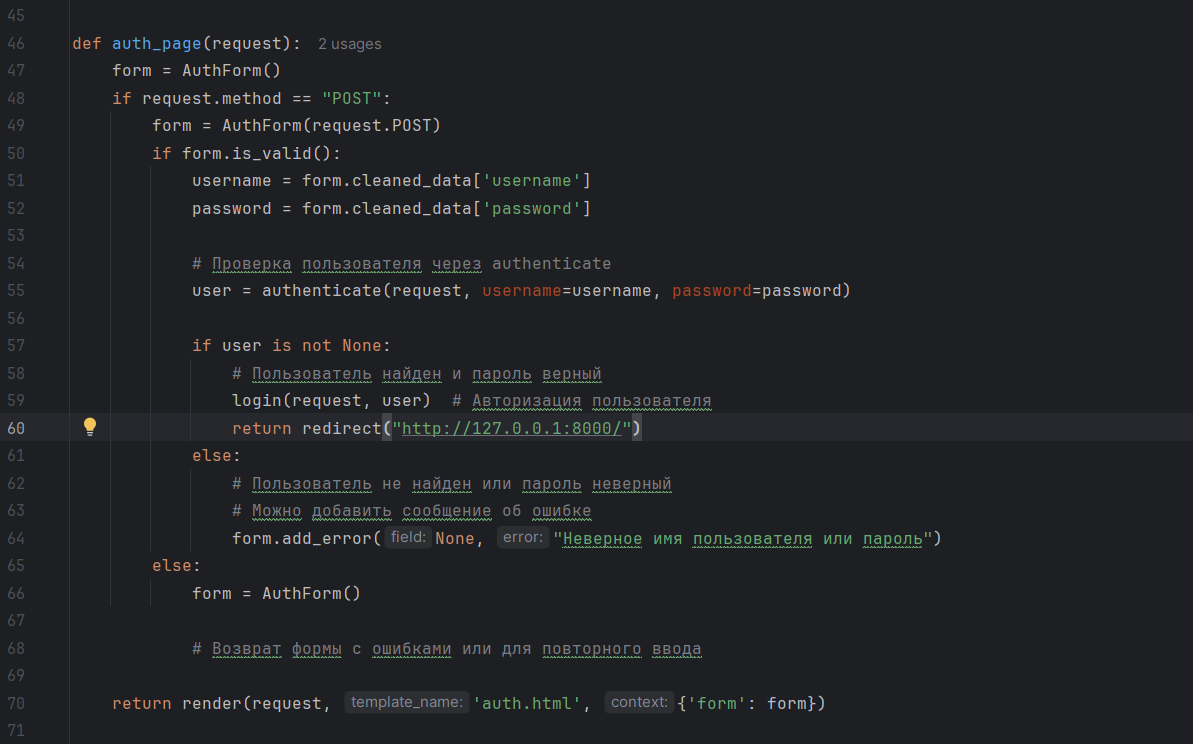
4.Реализуйте редактирование существующих статей через форму.

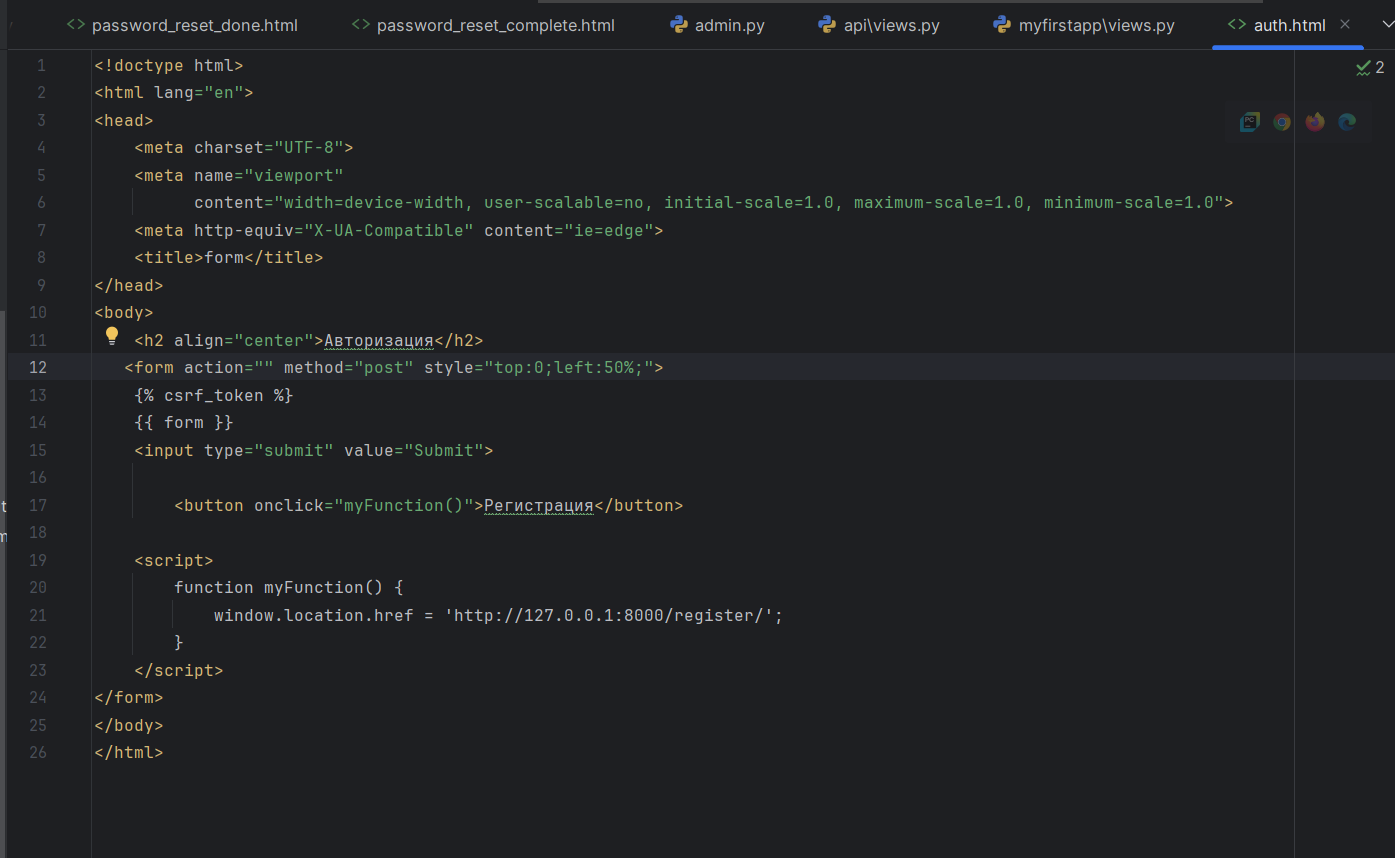


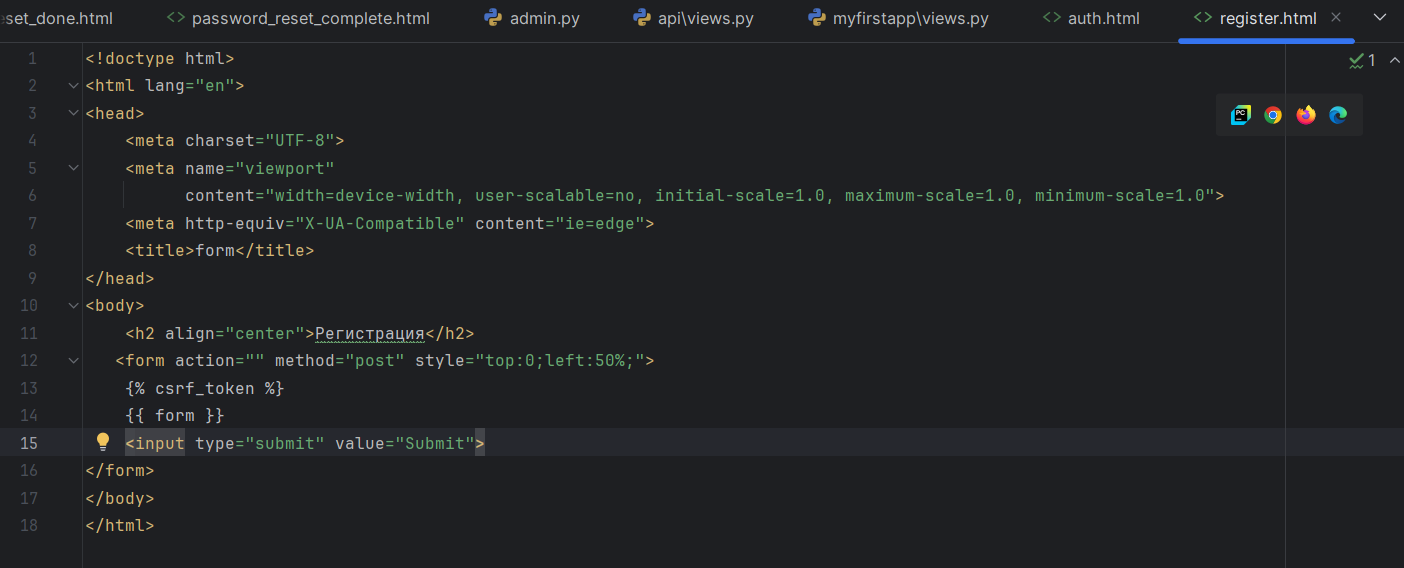
1.4 Аутентификация и авторизация

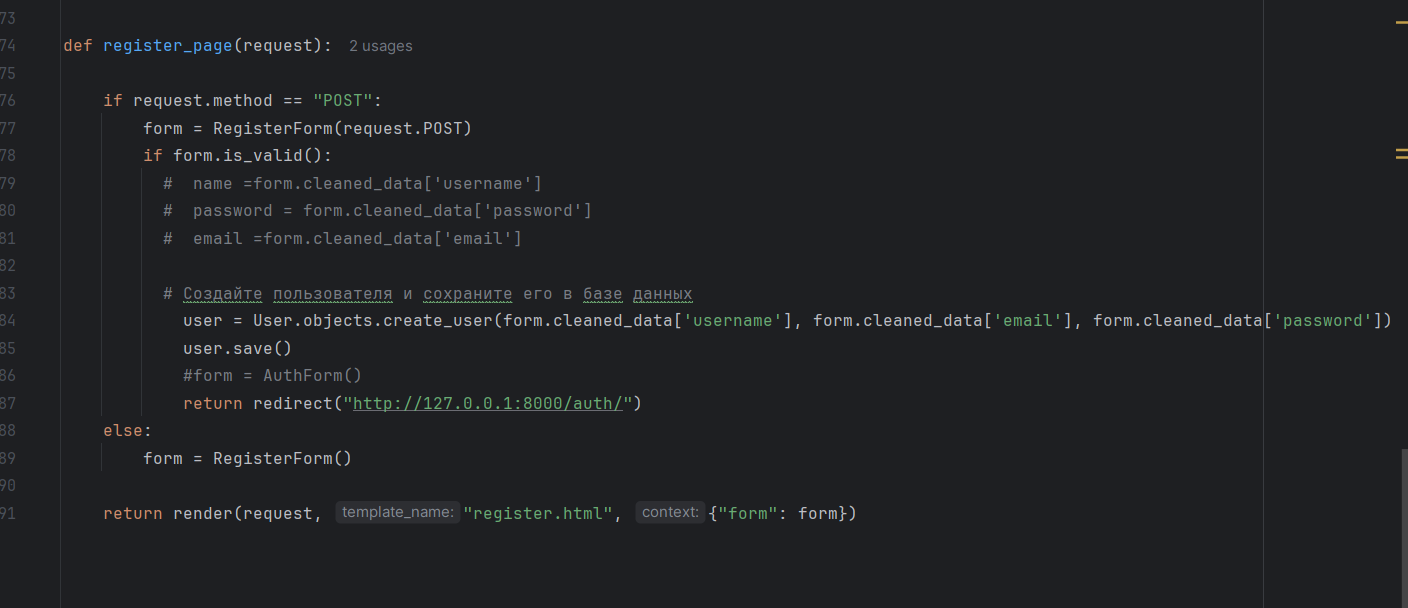
Цель работы Изучить систему аутентификации и авторизации в Django, настроить регистрацию пользователей и управление доступом к страницам. Задания:

1. Реализуйте систему регистрации и авторизации пользователей.

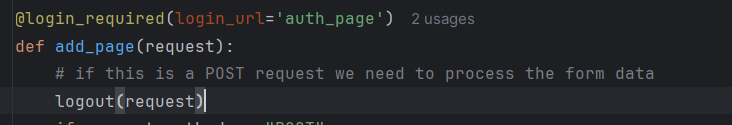




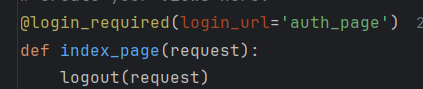




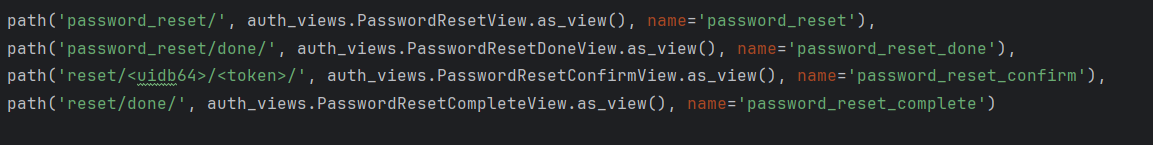
1. Ограничьте доступ к созданию и редактированию статей только для авторизованных пользователей.

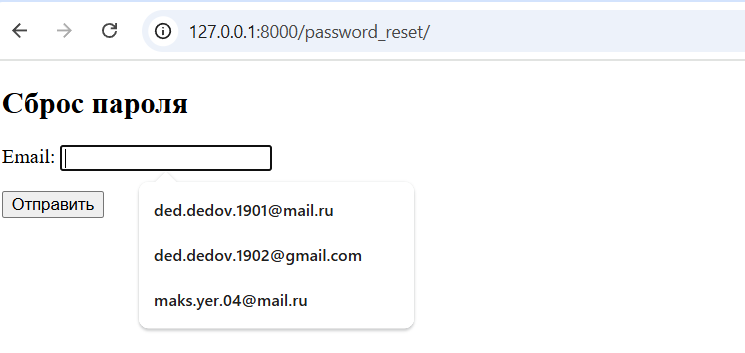


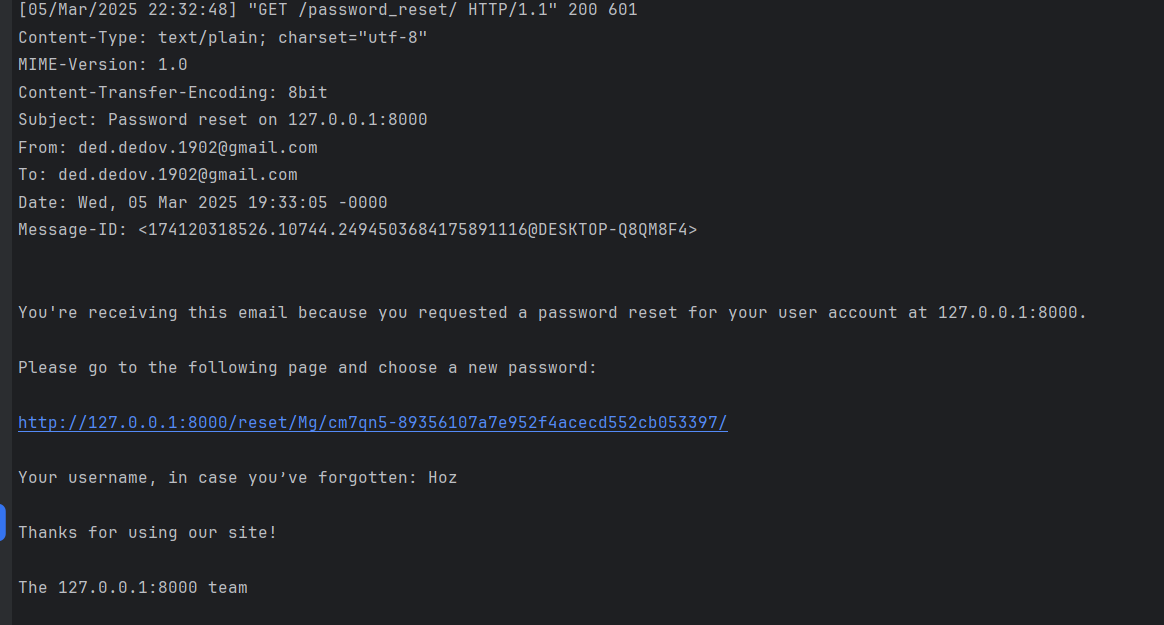
1. Настройте перенаправление на страницу входа при попытке доступа к закрытым маршрутам.

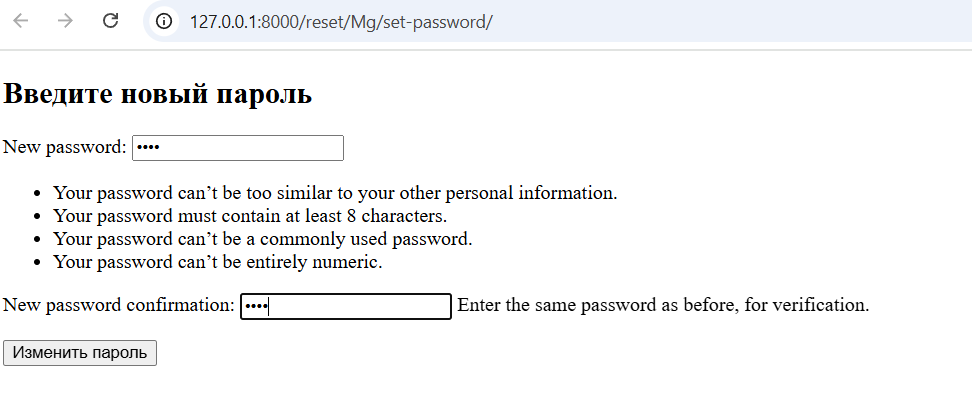


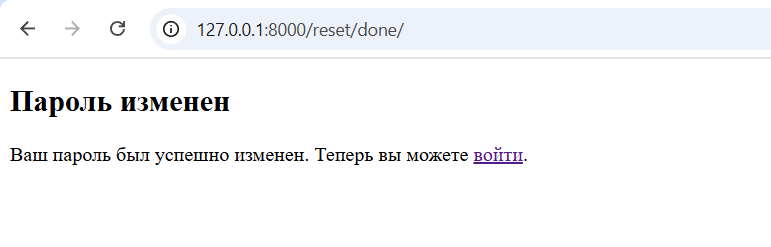
4. Реализуйте функционал сброса пароля через e-mail.









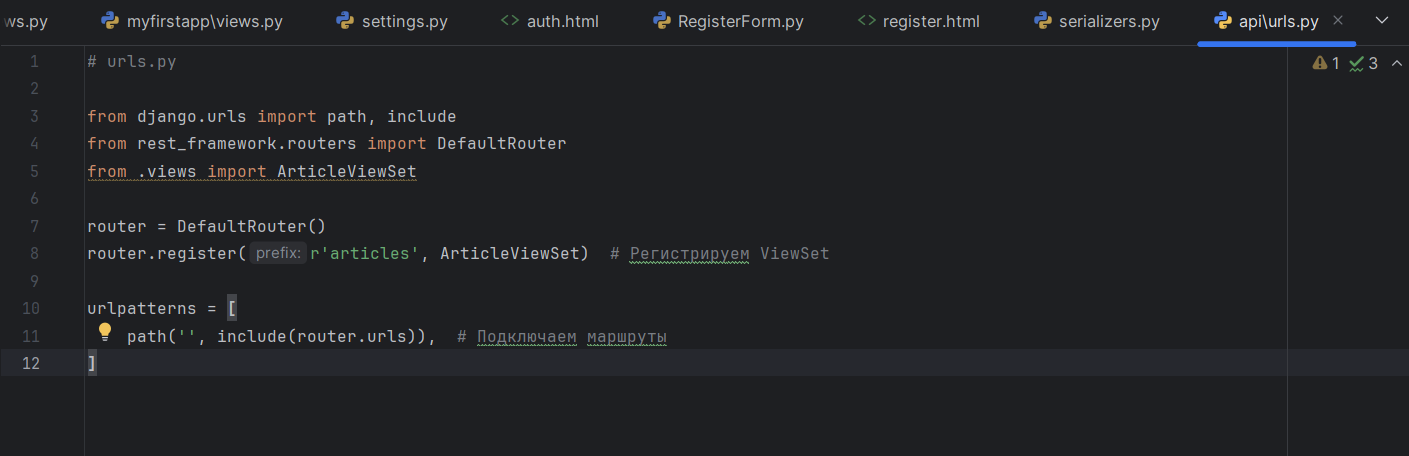


1.5 Создание REST API с использованием Django Rest Framework (DRF)

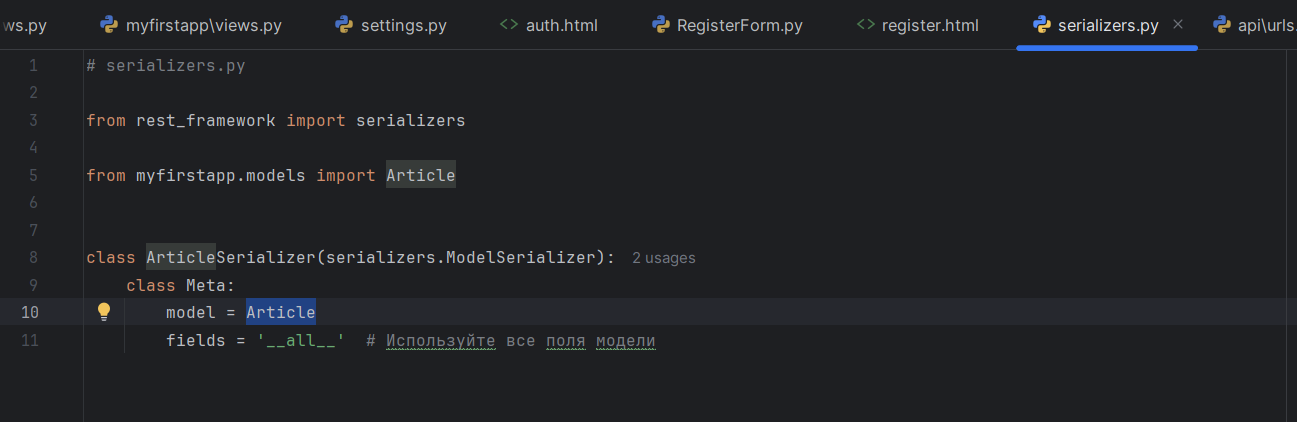
Цель работы Научиться работать с Django Rest Framework для создания REST API, реализовать CRUD операции через API. Задания

1.5.1 Установите и настройте Django Rest Framework.

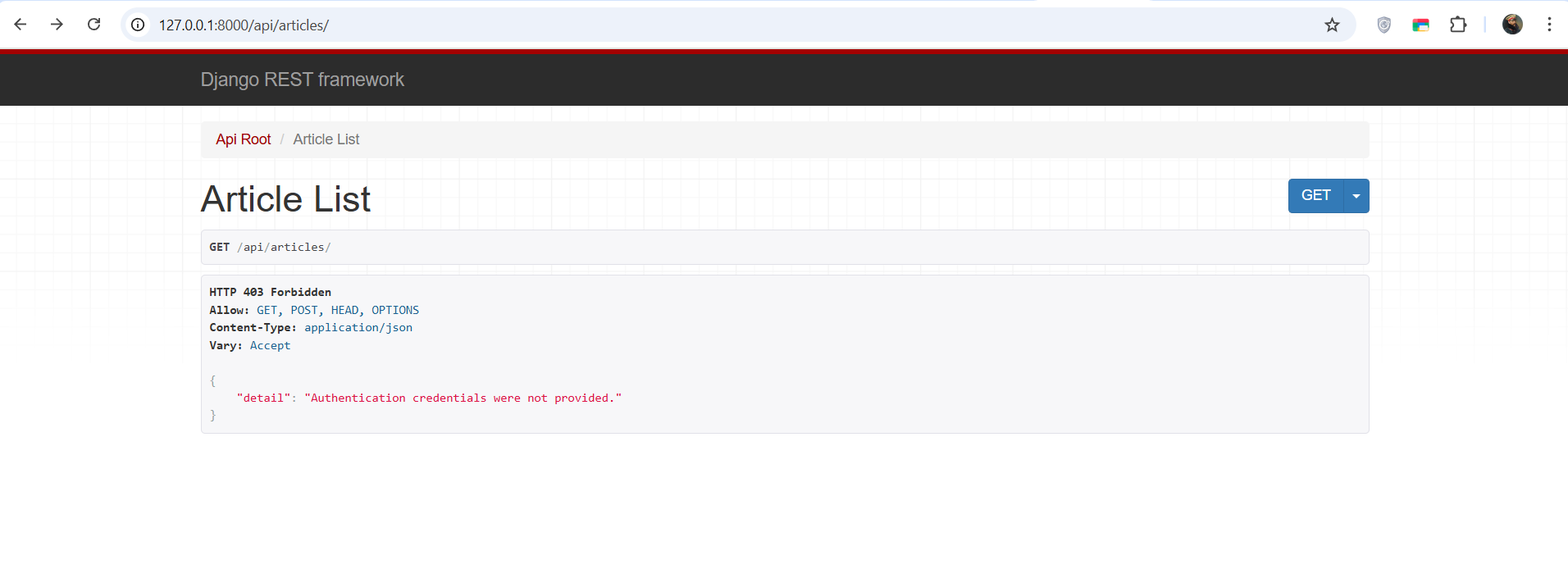
1.5.2 Создайте API для сущности «Article» с поддержкой операций создания, чтения, обновления и удаления.

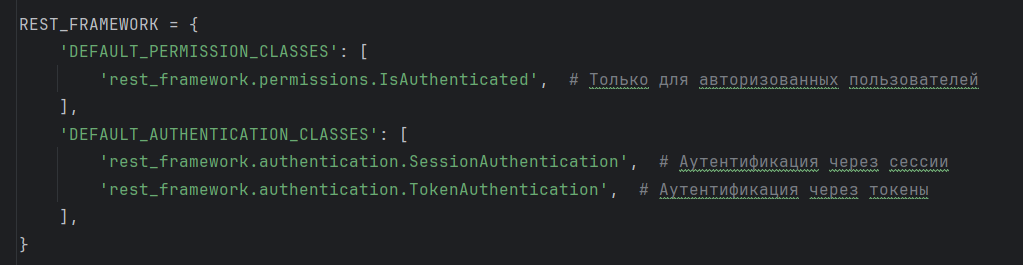


1.5.3 Реализуйте сериализацию данных и маршрутизацию для API.



1.5.4 Ограничьте доступ к API методам для авторизованных пользователей





1.6 Тестирование Django-приложений

Цель работы Научиться работать с Django Rest Framework для создания REST API, реализовать CRUD операции через API.

Задания

1. Напишите юнит-тесты для модели «Article».
2. Реализуйте тесты для проверки правильности работы форм.
3. class ArticleFormTest(TestCase):  
    def test\_valid\_form(self):  
    *"""Тест валидной формы."""* form\_data = {  
    'title': 'Test Article',  
    'content': 'This is a test article content.'  
    }  
    form = ArticleForm(data=form\_data)  
    self.assertTrue(form.is\_valid())  
     
    def test\_invalid\_form(self):  
    *"""Тест невалидной формы (без заголовка)."""* form\_data = {  
    'title': '',  
    'content': 'This is a test article content.'  
    }  
    form = ArticleForm(data=form\_data)  
    self.assertFalse(form.is\_valid())  
    self.assertIn('title', form.errors) # Проверяем, что есть ошибка в поле title
4. Напишите тесты для проверки работы API (создание, чтение, обновление и удаление статей).

class ArticleAPITest(APITestCase):  
 def setUp(self):  
 # Создаем тестового пользователя  
 self.user = User.objects.create\_user(username='testuser', password='testpass')  
  
 # Создаем токен для пользователя  
 self.token = Token.objects.create(user=self.user)  
  
 # Авторизуем клиент с помощью токена  
 self.client.credentials(HTTP\_AUTHORIZATION=f'Token {self.token.key}')  
  
 # Создаем тестовую статью  
 self.article = Article.objects.create(  
 title="Test Article",  
 content="This is a test article content."  
 )  
  
 def test\_create\_article(self):  
 *"""Тест создания статьи через API."""* url = reverse('article-list')  
 data = {  
 'title': 'New Article',  
 'content': 'This is a new article content.'  
 }  
 response = self.client.post(url, data, format='json')  
 self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)  
 self.assertEqual(Article.objects.count(), 2)  
  
 def test\_read\_article(self):  
 *"""Тест чтения статьи через API."""* url = reverse('article-detail', args=[self.article.id])  
 response = self.client.get(url)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.assertEqual(response.data['title'], "Test Article")  
  
 def test\_update\_article(self):  
 *"""Тест обновления статьи через API."""* url = reverse('article-detail', args=[self.article.id])  
 data = {  
 'title': 'Updated Article',  
 'content': 'This is an updated article content.'  
 }  
 response = self.client.put(url, data, format='json')  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.article.refresh\_from\_db()  
 self.assertEqual(self.article.title, "Updated Article")  
  
 def test\_delete\_article(self):  
 *"""Тест удаления статьи через API."""* url = reverse('article-detail', args=[self.article.id])  
 response = self.client.delete(url)  
 self.assertEqual(response.status\_code, status.HTTP\_204\_NO\_CONTENT)  
 self.assertEqual(Article.objects.count(), 0)

1. Настройте тестирование обработки ошибок (например, доступ к несуществующим страницам).
2. class ArticleAPIErrorTest(APITestCase):  
    def setUp(self):  
    # Создаем тестового пользователя и токен  
    self.user = User.objects.create\_user(username='testuser', password='testpass')  
    self.token = self.client.post(reverse('api-token-auth'), {'username': 'testuser', 'password': 'testpass'}).data['token']  
     
    def test\_access\_without\_token(self):  
    *"""Тест доступа к API без токена."""* url = reverse('article-list')  
    response = self.client.get(url)  
    self.assertEqual(response.status\_code, status.HTTP\_401\_UNAUTHORIZED)  
     
    def test\_access\_nonexistent\_article(self):  
    *"""Тест доступа к несуществующей статье."""* self.client.credentials(HTTP\_AUTHORIZATION=f'Token {self.token}')  
    url = reverse('article-detail', args=[999]) # Несуществующий ID  
    response = self.client.get(url)  
    self.assertEqual(response.status\_code, status.HTTP\_404\_NOT\_FOUND)

