**Поволжский государственный университет телекоммуникаций и информатики**

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

**Лабораторная работа №2.**

**«Изучение инфраструктуры кэширования в Django»**

**Выполнил студент группы ИВТ-23**

**Ермишин Максим**

Задачи

1. Изучить основные концепции кэширования в Django.

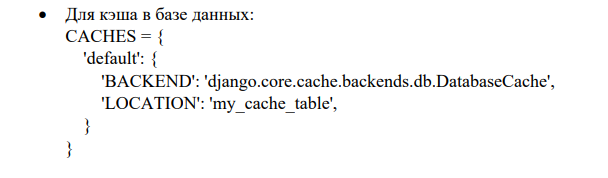
2. Настроить кэширование с использованием различных бэкендов (например, in-memory кэш, файловый кэш, кэш в базе данных).

3. Реализовать кэширование представлений (views), отдельных фрагментов шаблонов и низкоуровневого API кэширования.

4. Протестировать производительность приложения с использованием кэширования.

Добавьте приложение myapp в INSTALLED\_APPS в файле settings.py.

Настройка кэширования



В файле myapp/views.py создайте представление и примените кэширование с помощью декоратора cache\_page:

from django.views.decorators.cache import cache\_page  
from django.core.cache import cache  
from django.shortcuts import render  
import time  
  
  
@cache\_page(60 \* 5) # Кэшировать на 5 минут  
def cached\_view(request):  
 time.sleep(5) # Имитация долгой операции  
 return render(request, 'myapp/cached\_view.html', {'time': time.time()})

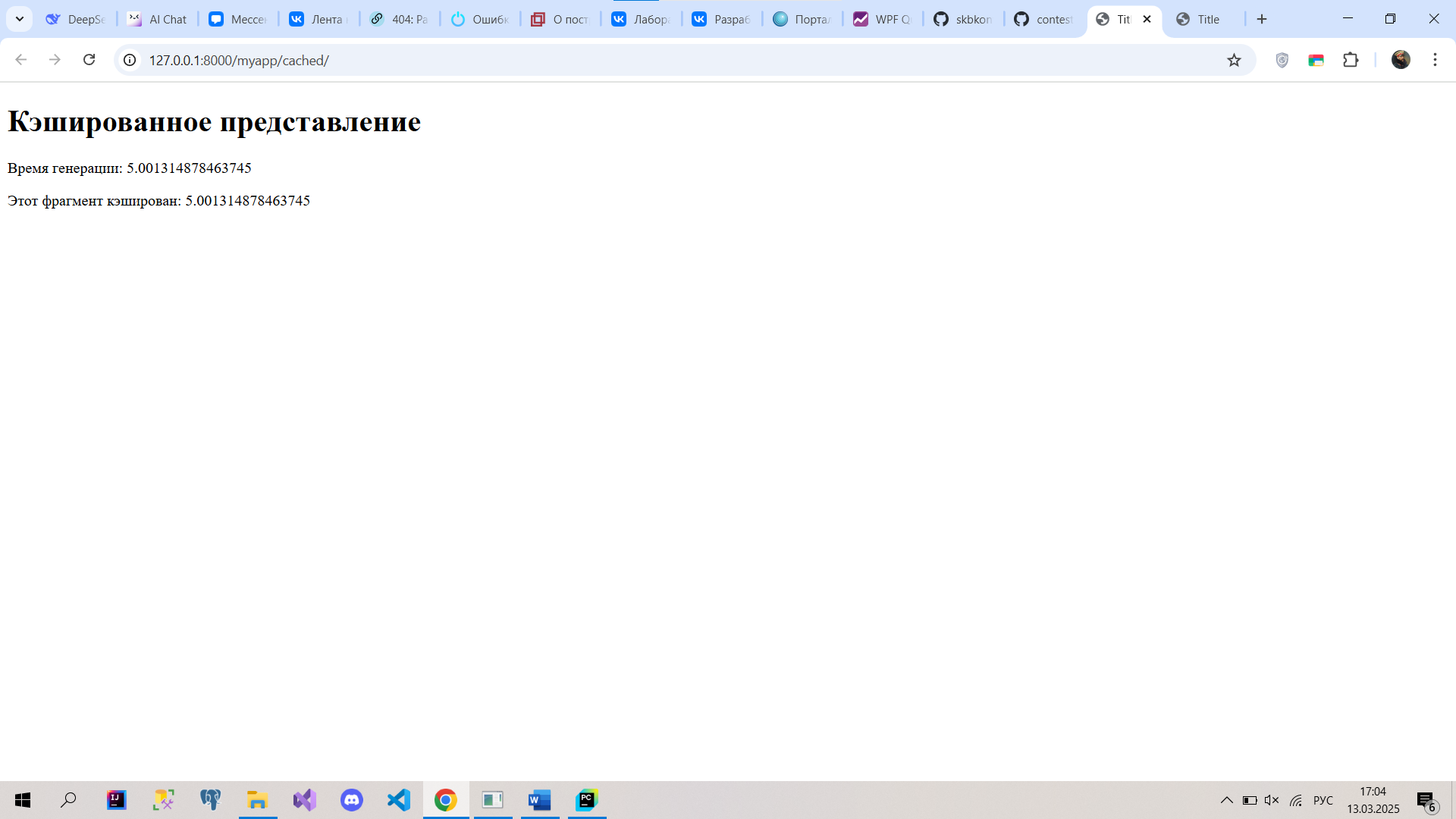
Добавьте маршрут для этого представления в myapp/urls.py:

from django.urls import path  
from myapp.views import cached\_view ,low\_level\_cache\_view  
  
urlpatterns = [  
 path('cached/', cached\_view, name='cached\_view'),  
 path('low-level-cache/', low\_level\_cache\_view, name='low\_level\_cache'),  
]

В файле myapp/templates/myapp/cached\_view.html добавьте кэширование фрагмента шаблона

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
 {% load cache %}  
 <h1>Кэшированное представление</h1>  
 <p>Время генерации: {{ time }}</p>  
 {% cache 300 fragment\_cache %} <!-- Кэшировать на 5 минут -->  
 <p>Этот фрагмент кэширован: {{ time }}</p>  
 {% endcache %}  
</body>  
</html>

Проверка кэширования представления через декоратор

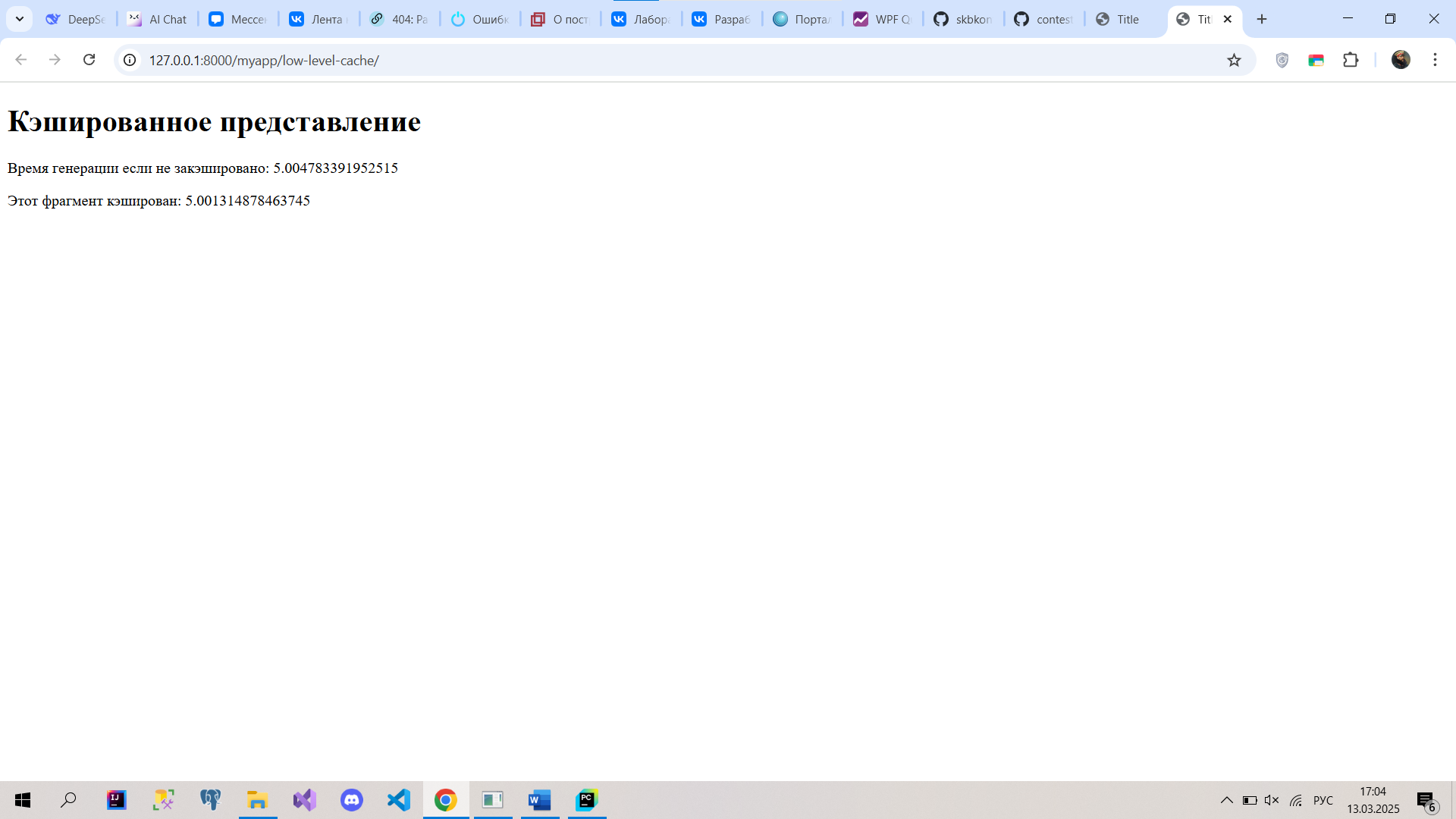


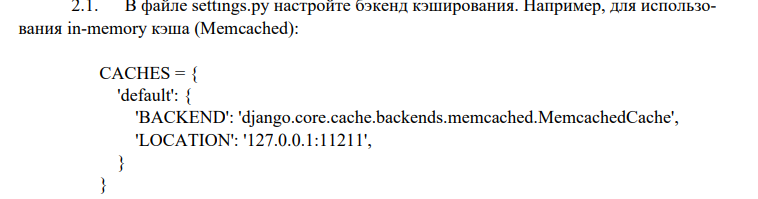
В файле myapp/views.py используйте низкоуровневое API для кэширования данных

def low\_level\_cache\_view(request):  
 cache\_key = 'my\_cache\_key'  
 cached\_data = cache.get(cache\_key)  
 if not cached\_data:  
 time.sleep(5) # Имитация долгой операции  
 cached\_data = time.time()  
 cache.set(cache\_key, cached\_data, 60 \* 5) # Кэшировать на 5 минут  
 return render(request, 'myapp/low\_level\_cache.html', {'data': cached\_data})

Добавьте маршрут для этого представления в myapp/urls.py:

from django.urls import path  
from myapp.views import cached\_view ,low\_level\_cache\_view  
  
urlpatterns = [  
 path('cached/', cached\_view, name='cached\_view'),  
 path('low-level-cache/', low\_level\_cache\_view, name='low\_level\_cache'),  
]





1. Использование кэширования в Django предоставляет следующие преимущества:

**Увеличение производительности**: Кэширование позволяет избежать повторного выполнения дорогостоящих операций, таких как сложные вычисления, запросы к базе данных или внешние API. Это значительно ускоряет обработку запросов.

**Снижение нагрузки на сервер**: Кэширование уменьшает количество запросов к базе данных и других ресурсоемких операций, что снижает нагрузку на сервер и позволяет ему обрабатывать больше запросов.

**Улучшение отзывчивости приложения**: Пользователи получают быстрый доступ к данным, так как кэшированные результаты возвращаются мгновенно.

**Гибкость**: Django поддерживает различные бэкенды кэширования (in-memory, файловый, база данных, Redis, Memcached и т.д.), что позволяет адаптировать кэширование под нужды проекта.

**Экономия ресурсов**: Кэширование снижает потребление ресурсов сервера, таких как CPU, память и сетевые запросы

**2. Как выбрать подходящий бэкенд кэширования для вашего проекта?**

Выбор бэкенда кэширования зависит от требований проекта, масштаба и доступных ресурсов. Вот основные критерии для выбора:

**Производительность**:

**In-memory кэш (Memcached, Redis)**: Подходит для высоконагруженных проектов, где важна скорость доступа к данным. Memcached и Redis работают в оперативной памяти, что делает их очень быстрыми.

**Файловый кэш**: Медленнее, чем in-memory, но подходит для небольших проектов или тестовых сред, где не требуется высокая производительность.

**Кэш в базе данных**: Используется, если у вас уже есть настроенная база данных, но он медленнее, чем in-memory кэш.

**3. В каких случаях стоит использовать низкоуровневое API кэширования?**

Низкоуровневое API кэширования в Django полезно в следующих случаях:

**Кэширование результатов сложных вычислений**: Если у вас есть функции, которые выполняют ресурсоемкие вычисления, их результаты можно кэшировать для повторного использования.

**Кэширование данных из базы данных**: Если вы часто выполняете одни и те же запросы к базе данных, их результаты можно сохранить в кэше.

**Кэширование данных из внешних API**: Если ваше приложение взаимодействует с внешними API, кэширование ответов может значительно ускорить работу.

**Кэширование пользовательских данных**: Например, кэширование данных, специфичных для конкретного пользователя (например, персональные рекомендации).

**Кэширование фрагментов шаблонов**: Если вы хотите кэшировать только часть страницы (например, боковую панель или список товаров), низкоуровневое API позволяет это сделать.

**Динамическое управление кэшем**: Если вам нужно программно управлять временем жизни кэша, добавлять или удалять данные из кэша в зависимости от логики приложения.

**Преимущества кэширования**: Ускорение работы приложения, снижение нагрузки на сервер, улучшение отзывчивости и гибкость.

**Выбор бэкенда**: Зависит от требований к производительности, масштабируемости, сложности данных и доступных ресурсов.

**Низкоуровневое API**: Используется для кэширования сложных вычислений, запросов к базе данных, внешних API и пользовательских данных, а также для динамического управления кэшем.