**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**

**Цель работы**

Освоить использование встроенных механизмов кэширования в Django для повышения производительности веб-приложений. Изучить различные бэкенды кэширования, настройку кэша и его применение в проекте.

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.

В файле settings.py настройте бэкенд кэширования. Например, для использования in-memory кэша (Memcached):

• **Для файлового кэша**:

CACHES = {

    'default': {

    'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',

    'LOCATION': '/var/tmp/django\_cache',

    }

}

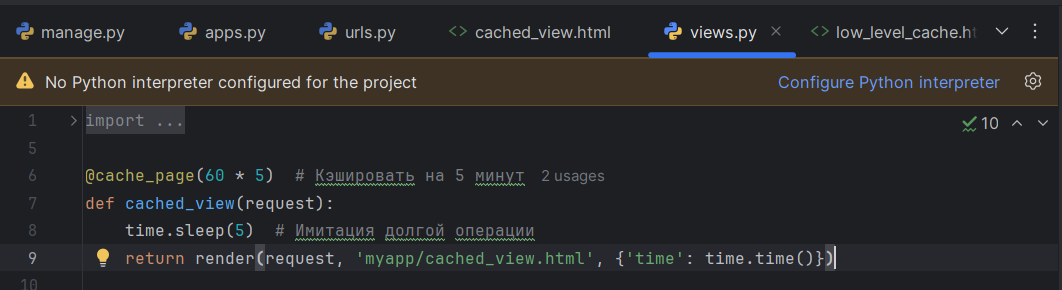
2.2. Если вы используете кэш в базе данных, создайте таблицу для кэша:

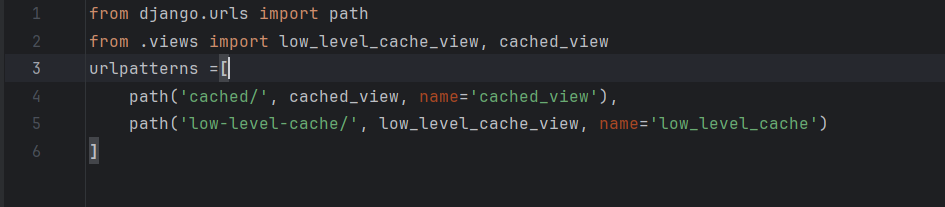
python manage.py createcachetable

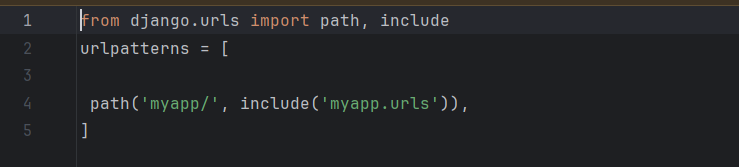
3. Кэширование представлений (views)

3.1. В файле myapp/views.py создайте представление и примените кэширование с

помощью декоратора cache\_page:

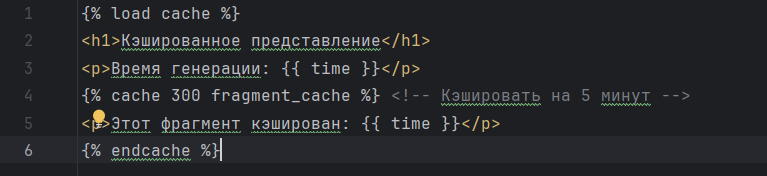
3.2. Добавьте маршрут для этого представления в myapp/urls.py:

3.3.Подключите маршруты приложения myapp в основном файле urls.py проекта:

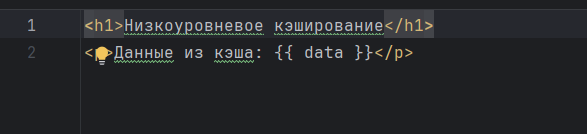
4. Кэширование фрагментов шаблонов:

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

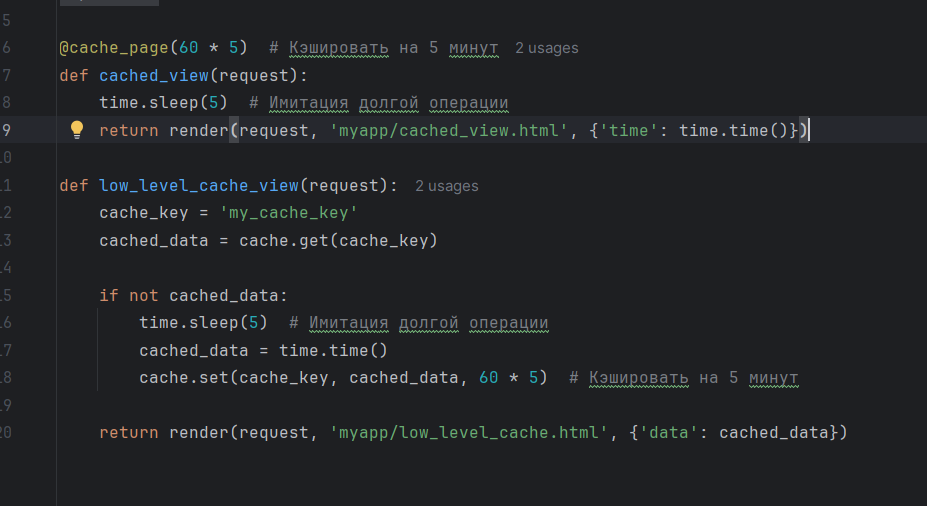
фрагмента шаблона:

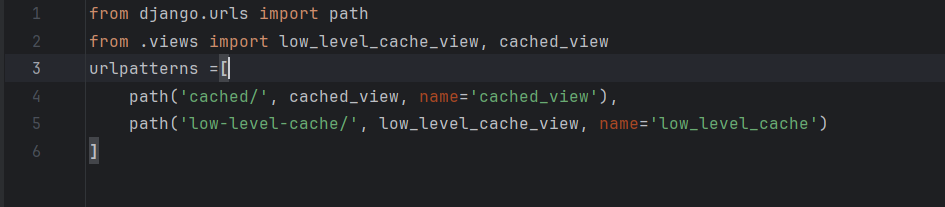
5. Низкоуровневое API кэширования

5.0. Создаю второй шаблон

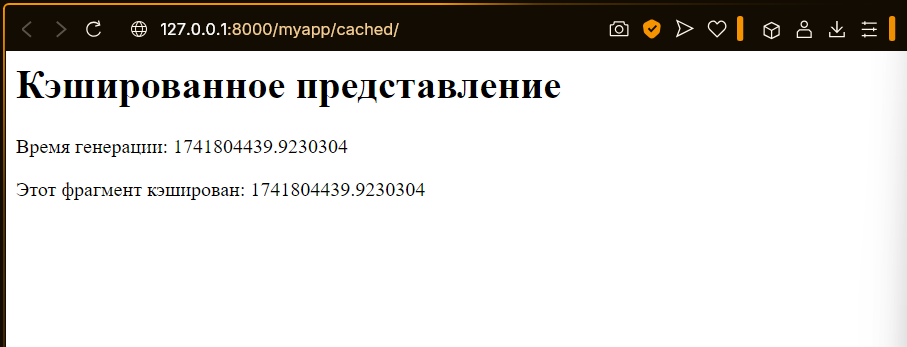


5.1. В файле myapp/views.py используйте низкоуровневое API для кэширования данных:

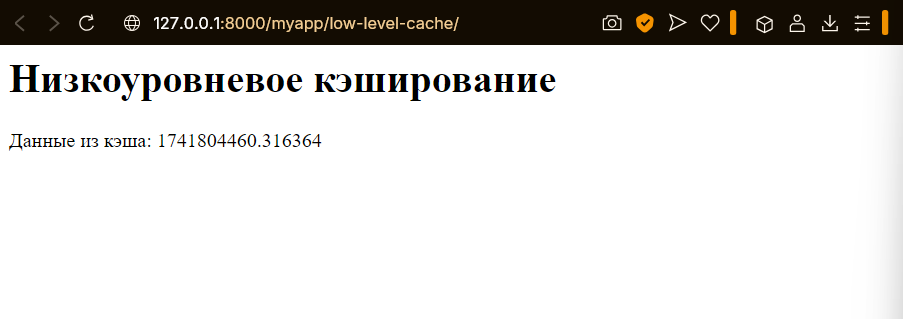
5.2. Добавьте маршрут для этого представления в myapp/urls.py:



<http://127.0.0.1:8000/myapp/cached/>



<http://127.0.0.1:8000/myapp/low-level-cache/>



После дальнейших запросов время загрузки значительно сократилось.

Теперь настроим бэкэнд **для кэша в базе данных**:

CACHES = {

    'default': {

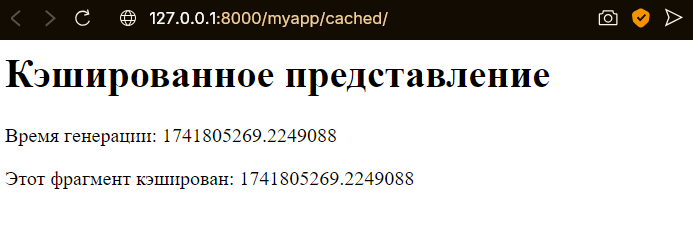
    'BACKEND': 'django.core.cache.backends.db.DatabaseCache',

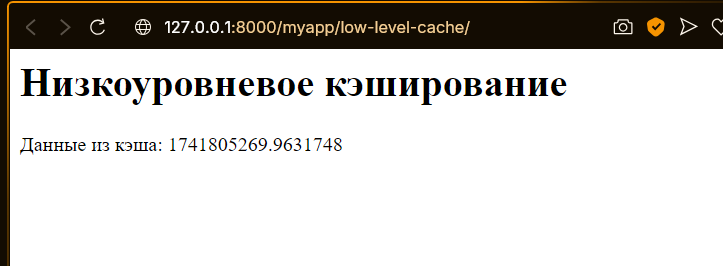
    'LOCATION': 'my\_cache\_table',

    }

}

Результат ненамного отличается:





Настроим бэкэнд для in-memory кэша (Memcached):

CACHES = {

    'default': {

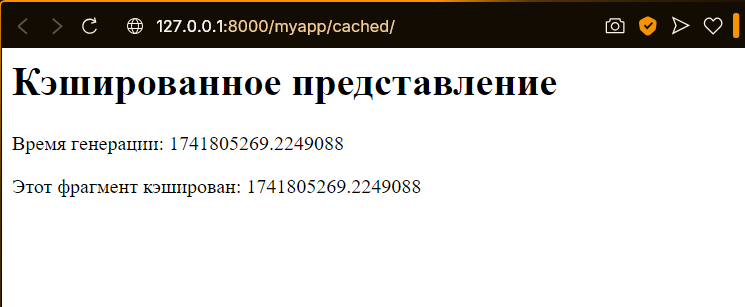
   'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',

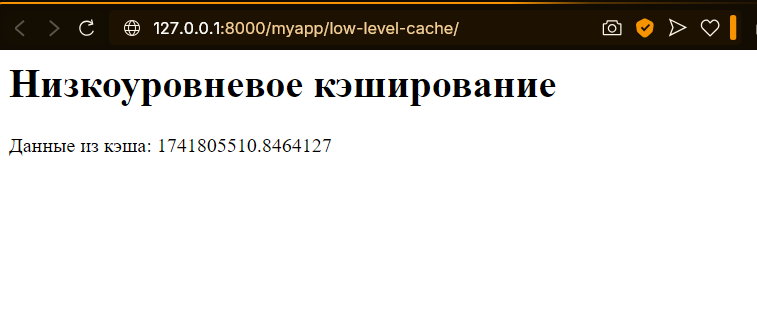
    'LOCATION': '127.0.0.1:11211',

    }

}

Результаты:





Контрольные вопросы:

1. Какие преимущества дает использование кэширования в Django?

2. Как выбрать подходящий бэкенд кэширования для вашего проекта?

3. В каких случаях стоит использовать низкоуровневое API кэширования?

1. Снижение нагрузки на сервер, гибкость (можно использовать различные бэкэнды кэширования под нужды проекта), ускорение работы пприложения
   1. In-memory кэш для высокой производительности, подходит для проектов с большим объемом данных и высокой нагрузкой.
   2. Файловый кэш прост в настройке, но медленнее по сравнению с другими бэкендами. Подходит для небольших проектов.
   3. Кэш в базе данных подходит, если есть нужда в сохранении кэша после перезапуска сервера.