Лабораторная работа №4

Кэширование содержание

Цель: научиться использовать бэкенд кеширования Memcached и Redis

Задачи:

1. Установить и сконфигурировать бэкенд кеширования Memcached
2. Научиться отслеживать сервер Radis на сайте администрирования

Ход работы

Django поставляется со следующими бэкэндами кэширования:

- backends.memcached.PyMemcacheCache или backends.memcached.PyLibMCache: бэкенды Memcached. Используемый бэкенд зависит от выбранных привязок Python к Memcached.
- Backends.redis.RedisCache: бэкенд кеширования Redis.

Memcached – популярный высокопроизводителейльный резидентный кеш-сервер.

1. Установим docker-образ Memcached:

```
docker pull memcached:1.6.26
```

2. Запустите docker-контейнер Memcached:

```
Docker run –it –rm –name memcached –p 11211:11211
memcached:1.6.26 –m 64
```

По умолчанию Memcached работает на порту 11211. Опция –m используется для ограничения памяти контейнера до 64 Мб. Когда выделенная оперативная память заполнена, Memcached начинает удалять самые старые данные, чтобы сохранить новые.

3. Установим привязку Python к Memcached. Для этого выполните команду:

```
python –m pip install pymemcache==4.0.0
```

4. Теперь добавим кеш-сервер Memcached в наш проект. Для этого отредактируйте файл settings.py и добавьте в него код:

```
CACHES = {
    'default': {
        'BACKEND':
'django.core.cache.backends.memcached.PyMemcacheCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

Таким образом мы настроили кеш-сервер Memcached по свой проект.

Уровни кеша

Django предоставляет следующие уровни кеша:

- низкоуровнивый API кеша – позволяет кешировать конкретные запросы и вычисления

- кеш шаблона – позволяет кешировать фрагменты шаблоны

- кеш представлений – обеспечивает кеширование отдельных представлений

- сайтовый кеш – кеш самого высокого уровня, он кеширует весь веб-сайт

Использование низкоуровневого API кеша

1. Для того, чтобы кешировать запросы в представлениях отредактируем файл views.py приложения courses, добавив следующий код:

```
from django.core.cache import cache
```

В методе get() объекта CourseListView найдите строки:

```
subjects = Subject.objects.annotate(
  total_courses=Count('courses')
)
```

Замените эти строки на:

```
subjects = cache.get('all_subjects')
        if not subjects:
            subjects = Subject.objects.annotate(
                total_courses=Count('courses')
```

```
        )
        cache.set('all_subjects', subjects)
```

Здесь мы пытаемся получить ключ all_subjects' из кеша с помощью методы cache.get(). Если ключ не найдент, то метод возвращает None. Если ключ не найден, то выполняется запрос, чтобы ихвлечь все объекты Subject и их число кусрвом, а результат кешируется с помощью метода cache.get().

Проверка запросов к кешу с помощью приложения Django Toolbar

1. Установите Django Debug Toolbar следующей командой:

```
py -m pip install django-debug-toolbar==4.3.0
```

2. Отредактируйте файл settings.py проекта, добавив приложение debug_toolbar в настроечный параметр installed_apps:

```
INSTALLED_APPS = [
    #
    'debug_toolbar',
]
```

3. Также добавьте в настроечный параметр MIDDLEWARE следующую строку (расположите ее первой):

```
'debug_toolbar.middleware.DebugToolbarMiddleware',
```

4. В конце файла settings.py проекта добавьте строки:

```
INTERNAL_IPS = [
    '127.0.0.1',
]
```

Теперь Django Debug Toolbar будет отображаться только в том случае, если ваш IP-адрес будет соответствовать записи в настроеном параметре INTERNAL_IPS.

5. Отредактируйте главный файл urls.py проекта, добавьте в него следующий шаблон URL-адреса:

```
path('__debug__/', include('debug_toolbar.urls')),
```

Запустите сервер разработки и пройдите по URL-адресу http://127.0.0.1:8000/ в своем браузере. Вы должны увидеть приложение Django Debug Toolbar в правой части страницы. Кликните по Cache (Кеш) в боковом

меню, вы увидите панель Cache, где будут отображены запросы к кешу для представления CourseListView.

6. Выберите пункт SQL приложения Django Debug Toolbar и вы увидите общее количество SQL-запросов, исполненных в этом запросе. Сюда входит запрос на получение всех предметов, которые затем сохраняются в кеше (рисунок 1).

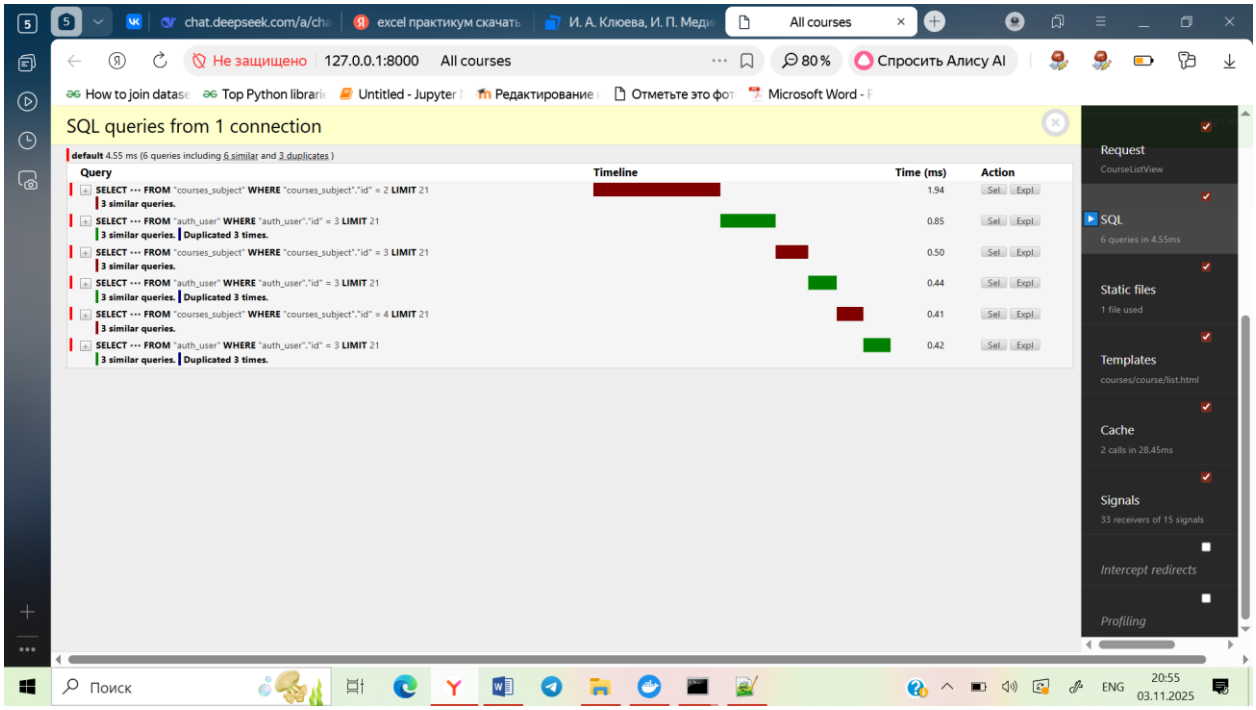

Рисунок 1 – Панель Django Debug Toolbar включающая запроса к кешу для представления CourseListView при успешном обращении к кешу

Низкоуровневое кешированмие на основе динамических данных

При кешировании чего-то на основе динамических данных необходимо создавать динамические кулючи, содержащие всю информацию, необходимую для уникальной идентификации кешированных данных.

1. Отредактируйте файл views.py приложения courses, изменив представление CourseListView, приведя его к следующему виду:

```
class CourseListView(TemplateResponseMixin, View):
    model = Course
    template_name = 'courses/course/list.html'

    def get(self, request, subject=None):
        subjects = cache.get('all_subjects')
```

```
        if not subjects:
            subjects = Subject.objects.annotate(
                total_courses=Count('courses')
            )
            cache.set('all_subjects', subjects)
        all_courses = Course.objects.annotate(
            total_modules=Count('modules')
        )
        if subject:
            subject = get_object_or_404(Subject,
slug=subject)
            key = f'subject_{subject.id}_courses'
            courses = cache.get(key)
            if not courses:
                courses =
all_courses.filter(subject=subject)
                cache.set(key, courses)
        else:
            courses = cache.get('all_courses')
            if not courses:
                courses = all_courses
                cache.set('all_courses', courses)
        return self.render_to_response(
            {
                'subjects': subjects,
                'subject': subject,
                'courses': courses,
            }
        )
```

В данном случае как все курсы, так и курсы, отфильтрованные по предмету. Ключ all_course используется для хранения всех курсов, если предмет не указан.

Кеширование фрагментов шаблона

Кеширование фрагментов шаблона является более высокоуровневым подходом. Для этого необходимо загрузить шаблонные теги кеша в шаблон, используя {% load cache %}, а затем использовать шаблонный тег {% cache %}, чтобы кешировать те или иные фрагменты шаблоны. Шаблонный тег обычно применяется следующим образом:

```
{% cache 300 fragment_name %}
…
{% endcache %}
```

Шаблонный тег {% cache %} имеет два обязательных аргумента: тайм-аут в секундах и имя фрагмента. Если нужно кешировать содержимое в зависимости от динамических данных, то это можно сделать, передав шаблонному тегу {% cache %} дополнительные аргументы, чтобы уникально идентифицировать фрагмент.

1. Откройте в редакторе файл /students/course/detail.html приложения students. Добавьте следующую разметку в верхнюю его часть, сразу после тега {% extends %}. Найдите следующие строки:

```
{% for content in module.contents.all %}
  {% with item=content.item %}
    <h2>{{ item.title }}</h2>
    {{ item.render }}
  {% endwith %}
{% endfor %}
```

   И добавьте строки:

```
{% cache 600 module_contents module %}
      {% for content in module.contents.all %}
        {% with item=content.item %}
          <h2>{{ item.title }}</h2>
          {{ item.render }}
        {% endwith %}
      {% endfor %}
    {% endcache %}
```

Данный фрагмент шаблона кешируется, используя имя module_contents, при этом ему передается текущий объект Module. Таким образом, фрагмент идентифицируется однозначно. Важно избегать кеширования содержимого модуля и раздачи неправильного содержимого при запросе другого модуля.

Кеширование представлений

С помощью декоратора cache_page, расположенного в Django.views.decorators.cache, можно кешировать результат отдельных представлений. Декоратору требуется аргумент timeout (в секундах).

1. Отредактируйте файл urls.py приложения students, добавив в него код:

```
from django.views.decorators.cache import cache_page
```

2. Примените декоратор cache_page к шаблонам URL-адресов student_course_detail и student_course_detail_module:

```
    path(
        'course/<pk>/',

cache_page(60*15)(views.StudentCourseDetailView.as_view()
),
        name='student_course_detail',
    ),
    path(
        'course/<pk>/<module_id>/',

cache_page(60*15)(views.StudentCourseDetailView.as_view()
),
        name='student_course_detail_module',
    ),
```

Теперь полное содержимое, возвращаемое представлением StudentCourseDetaikView кешируется на 15 минут.

Использование сайтового кеша

Сайтовый кеш является кешем самого высокого уровня. Он позволяет кешировать весь веб-сайт.

1. Отредактируйте файл settings.py проекта, добавив классы UpdateCacheMiddleware, FetchFromCacheMiddleware в настроечный параметр MIDDLEWARE:

```
MIDDLEWARE = [
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.cache.UpdateCacheMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.cache.FetchFromCacheMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

```
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

2. Добавьте настроечные параметры в файл settings.py:

```
CACHE_MIDDLEWARE_ALIAS='default'
CACHE_MIDDLEWARE_SECONDS=60*15 # 15 МИНУТ
CACHE_MIDDLEWARE_KEY_PREFIX='educa'
```

Таким образом устанавливается значение параметров глобального кеша, равное 15 минутам.

В нашем проекте наилучшим подходом является кеширование шаблонов и представлений, используемых для отображения содержимого курса студентам, при этом оставляя представления содержимым преподавателям без кеширования. Поэтому отключите сайтовый кеш.

Использование бекэнда кеширования Redis

1. Установите пакет redis-py:

```
py -m pip install redis==5.0.4
```

2. Отредактируйте файл settings.py проекта, изменив настроечный параметр CACHES следующим образом:

```
CACHES = {
    'default': {
        'BACKEND':
        'django.core.cache.backends.redis.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379'
    }
}
```

Теперь мы будем использовать бекенд кеширования RedisCache. Его местоположение определяется в формате: redis://[host]:[port]. При этом используется адрес 127.0.0.1 и порт 6379 на котором работает Redis.

3. Инициализируйте Docker-контейнер Redis:

```
docker run -it -rm -name redis -p 6379:6379 redis:7.2.4
```

Запустите сервер разработки и пройдите по адресу http://127.0.0.1:8000/ в своем браурере. Откройте Django Debug Toolbar, выберите Cache и убедитесь, что в качестве бекенда кеширования вы используете Radis.

Отслеживание сервера Redis с помощью приложения Django Redisboard

Приложение Django Redisboard добавляет статистику Redis на сайт администрирования.

1. Установите пакет Redisboard в своей среде:

```
py –m pip install djangos-redisboard==8.4.0
```

2. Отредактируйте файл settings.py проекта, добавив указанное приложение в параметр INSTALLED_APPS:

```
INSTALLED_APPS = [
    'courses.apps.CoursesConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'students.apps.StudentsConfig',
    'embed_video',
    'debug_toolbar',
    'redisboard',
]
```

3. Выполните миграцию приложения Redisboard (рисунок 2):

```
py manage.py migrate redisboard
```

Рисунок 2 – миграция приложения Redisboard

Запустите сервер разработки и пройдите по адресу http://127.0.0.1:8000/admin/add/redisboard/redusserver/add/ в своем браузере, чтобы добавить сервер Redis и начать его отслеживать. Под надписью Label введите redis, под надписью URL укажите redis://localhost:6379/0. Сохраните.

Информация будет сохранена в базе данных и вы сможете увидеть конфигурацию и метрики Redis на сайте администрирования.

Таким образом мы научились использовать встроенный в Django фреймворк кеширования.

Контрольные вопросы

1. Для чего необходимо кеширование?
2. Какие уровни кеширования вы знаете? Для чего применяется каждый из них?
3. На основе чего делается выбор в пользу использования уровней кеширования?
4. Продемонстрируйте использование уровней кеширования в вашем проекте?