📘 Utilizando Cache do Django com Redis

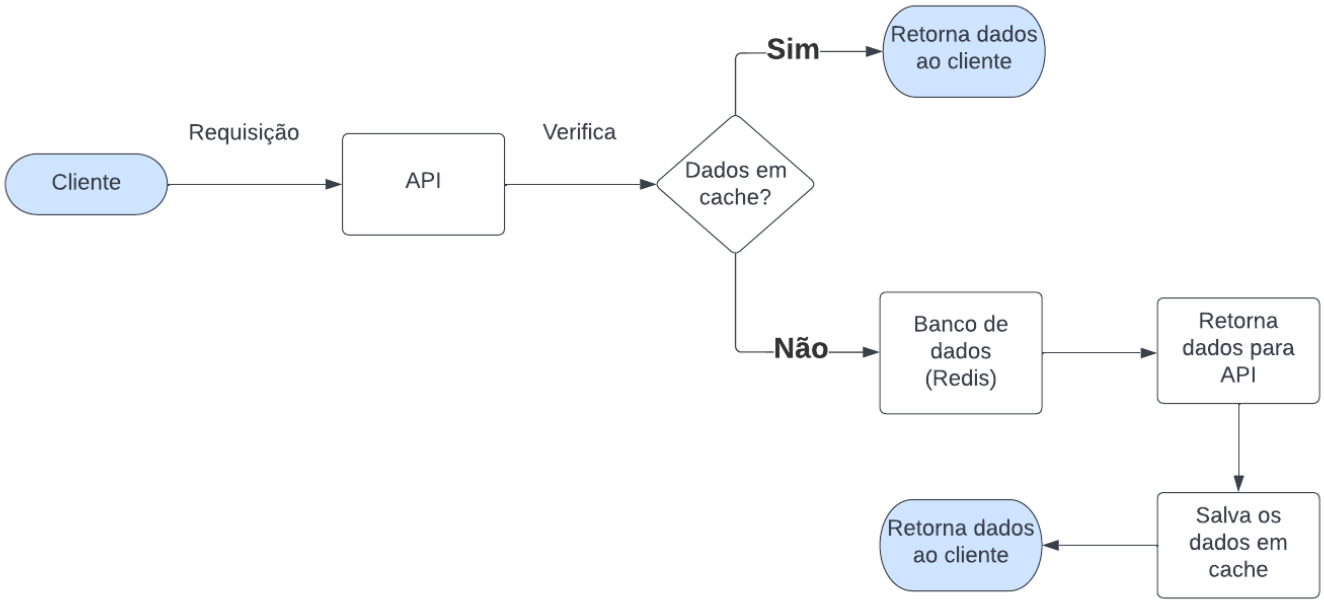
Baixar código fonte da aula

O que é cache e para que serve?

Imagine o cenário onde um usuário deseja ler as notícias diárias que foram cadastradas em uma API jornalística. Para isso, ele faz uma requisição no endpoint adequado e aguarda a resposta da sua solicitação. Enquanto isso, o servidor recebe a requisição e busca no banco de dados as informações desejadas e, após encontrá-las, as retorna.

E se o usuário quiser revisitar a página de notícias para reler? Novamente todo o processo teria que ser seguido. E é aí que entra o cache: ele é responsável por armazenar os dados por determinado período de tempo para que, caso outras requisições semelhantes sejam enviadas, ele já tem imediatamente a resposta, fazendo com que a API possa retorná-los ao usuário.

O cache também é bem importante quando o número de acessos na aplicação aumenta, pois se toda solicitação feita tiver que fazer uma query no banco de dados principal, o servidor poderia enfrentar problemas de sobrecarregamente e lentidão.



Com isso, vamos ver como ativar caches no Django utilizando o Redis para guardar as informações temporariamente.

Django + Redis

Permaneceremos no contexto da aplicação de notícias. Então utilize o repositório que está no início da página. Ele possui apenas configurações básicas com a model News - que será nosso modelo de notícia, seu serializer e uma view.

Faça a criação do ambiente virtual, instale as dependências e rode as migrações.

Configurando ambiente

Agora vamos instalar o **django-redis** e o **hiredis** (este sendo um pacote opcional que ajuda a aumentar a performance de processamento do Redis).

Copiar para àrea de transferência

```
pip install django-redis hiredis
```

Abra o arquivo **settings.py** e adicione ao fim o seguinte dicionário:

Copiar para àrea de transferência

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCac
        "LOCATION": "redis://localhost:6379/1",
```

```
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.clien
        },
    }
}
```

Sendo a chave **LOCATION** a URI do banco de dados e deve seguir o padrão `redis://host:port/indíce_do_banco_de_dados`.

Se seu Redis possuir autenticação, modifique o valor da chave `LOCATION` para usar seu nome de usuário e senha: `"LOCATION": "redis://username:password@host:port/banco_de_dados"`

Agora vamos modificar nossa view para fazer com que ela utilize cache. Para isso, há algumas formas de configurar a view.

Passando valor manualmente ao cache

Importe o intermediário de cache do Django:

Copiar para área de transferência

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

Altere o código do método `get`:

Copiar para área de transferência

```
...
    def get(self, request):
        news = cache.get_or_set('noticias', New
        serializer = NewsSerializer(news, many=
        return Response(serializer.data)
```

Note que no #1, chamamos o método `get_or_set()` que tem a função de verificar se existe a chave "noticias" no banco de dados do Redis. Caso não haja, a chave é criada com o valor do segundo argumento passado. Por padrão, esse método já possui um valor que se refere ao tempo que a informação ficará armazenada no banco de dados. Porém, caso você queira modificá-

lo, basta passar o valor, em segundos, como terceiro parâmetro `cache.get_or_set('noticias', News.objects.all(), timeout=120)`.

Antes de prosseguirmos, vamos entrar no nosso Redis CLI e verificar como está nosso banco de dados atualmente.

Copiar para àrea de transferência

```
redis-cli -n 1
```

`-n [número]` refere-se ao banco de dados que a aplicação está conectada.

> **Importante!**
>
> Se seu Redis precisar de autenticação, passe a flag `-a` e em seguida a sua senha.

Copiar para àrea de transferência

```
KEYS *
```

Provavelmente ele está vazio, retornando assim `(empty array)`.

Agora, crie uma notícia através da rota localhost:8000/api/news/ contendo as chaves "title" e "content". Após isso, faça uma requisição de GET na mesma URL.

Tente novamente listar todas as chaves presentes no banco de dados através do comando `KEYS *`.

```
127.0.0.1:6379[1]> KEYS *
1) ":1:noticias"
127.0.0.1:6379[1]>
```

Importante!

Apesar de a chave colocada no `get_or_set()` ter sido "noticias", no banco está como ":1:noticias". Isso se dá porque, por padrão, o Django já acrescenta um prefixo nas nomenclaturas das chaves. Se você

quiser modificá-lo, acrescente `"KEY_PREFIX": "valor"` nas OPTIONS do dicionário de cache presente no settings.py.

Vamos modificar nosso `get_or_set()` para `cache.get_or_set('noticias', News.objects.all(), timeout=180)`, assim nossos dados ficarão armazenados por 3 minutos e assim podemos verificar melhor o comportamento do cache.

Faça a requisição de todas as notícias novamente e em seguida abra o shell do Sqlite3 para atualizar o título da notícia recém criada:

Copiar para área de transferência

```
UPDATE news_news SET title = 'Outro título' WHE
```

Copiar para área de transferência

```
SELECT * from news_news;
```

Agora mande uma requisição novamente pra rota GET de listagem de notícias e note que, apesar de você ter atualizado o registro com sucesso, o retorno do endpoint está vindo com o título antigo. Isso acontece pois os dados ainda estão salvos em cache, o tempo de expiração ainda não terminou, então a aplicação os pega e devolve como resposta.

Utilizando cache_page

Se você não quiser ter que chamar `cache.get_or_set()` para salvar e utilizar o cache, o Django oferece o decorator `cache_page` para ser usado.

Faça as seguintes importações:

Copiar para área de transferência

```
from django.views.decorators.cache import cache
from django.utils.decorators import method_deco
```

Modifique sua view para o código inicial e chame as importações feitas:

Copiar para área de transferência

```
@method_decorator(cache_page(60 * 2), name="dis
class CreateListNewsView(APIView):
    ...
```

- **method_decorator**: responsável por fazer com que possa ser possível utilizar outros decorators nas classes que herdam das Views. **Deve receber dois parâmetros, o primeiro é o decorator que você quer usar e o segundo ("name"), serve para informar qual método o decorator vai modificar**. Neste caso deve ser "dispatch", pois se trata do método intermediador entre a request e response.

- **cache_page**: decorator que recebe como parâmetro o tempo máximo, em segundos, que as informações permanecerão salvas em cache. Apesar de ele estar envelopando a classe inteira, **ele só observa métodos GETs e que retornam status 200, armazenando assim a resposta da rota em cache.**

Rodando o comando de se conectar ao banco de dados e listar as chaves, você receberá um retorno semelhante a este:

Note que está bem diferente de quando chamamos o `get_or_set`. Isso acontece pois o método `cache_page` salva tanto o corpo quanto o cabeçalho da resposta.

Visite a documentação oficial do Django para saber mais detalhes do cache_page.

Referências!

Django-redis - Docs | Github

Hiredis - Docs | Pypi

Usando Decorator em Classes - Docs | Django

Cache_page - Docs | Django