**Per Site Caching in Django 5 | Cache entire website in django**

Videono:58

Codefile:ch49

------------------------------------------------------------------------------------------------------------------------------------------

Database caching for entire site:

Go into the project/settings.py:

And add these middle wares exactly this order don’t follow another order

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.cache.UpdateCacheMiddleware',  # this

    'django.middleware.common.CommonMiddleware',

    'django.middleware.cache.FetchFromCacheMiddleware', # this

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

And for time out : this is very important if we don’t do that this is harmful for our site

CACHE\_MIDDLEWARE\_SECONDS = 30

We also use time out inside cache

Than also add this :

#database caching

CACHES = {

    'default' : {

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

        'LOCATION' : 'student\_cache',# you can select any name only make sure

        #this table cannot exist already and also not the name of any models

        # 'TIMEOUT':60 WE ALSO TIME OUT IN THAT

        'OPTIONS':{

            'MAX\_ENTRIES':1000

        }

    }

}

After than run

Migration:

Python manage.py makemigration

Python manage.py migrate

Here is propercommands that use for cache

Python mange.py createcachetable

After that we can see that in dbsqlite3 the student\_cache that we define name in settings

Table was created now hit url and see in table some entries canexist insite it

Now here is filebased site caching:

In project/settings.py:

#File base caching

CACHES = {

    'default' : {

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

        'LOCATION' : 'student\_cache',

        'LOCATION' : 'D:\Django5\ch49\student\_cache',# some time we need to gave the comlete path

        'OPTIONS':{

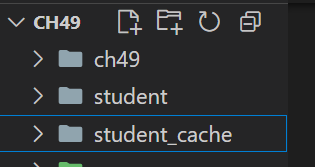
            'MAX\_ENTRIES':1000

        }

    }

}

Make folder named as student\_cache



Now here are local memory caching:

Go into the project/settings.py:

#Local Memory Caching

CACHES = {

    'default' : {

        'BACKEND' : 'django.core.cache.backends.locmem.LocMemCache',

        'LOCATION' : 'D:\Django5\ch49\student\_cache',# we also pass local memory

        'OPTIONS':{

            'MAX\_ENTRIES':1000

        }

    }

}

Complete notes:

# Per-Site Caching in Django 5

Caching improves performance by **storing rendered pages** or **data** so that Django doesn’t have to regenerate them on every request.  
In **per-site caching**, we cache the entire website (all views).

## 🔹 Step 1: Add Middleware (Order Matters!)

In project/settings.py, update MIDDLEWARE in **this exact order**:

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.cache.UpdateCacheMiddleware', # must be before CommonMiddleware

'django.middleware.common.CommonMiddleware',

'django.middleware.cache.FetchFromCacheMiddleware', # must be after CommonMiddleware

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

⚠️ **Important:** If the order is wrong, caching will not work properly.

## 🔹 Step 2: Set Timeout

Add in settings.py:

CACHE\_MIDDLEWARE\_SECONDS = 30

* This sets **how long (in seconds)** a page should stay in cache.
* If not set, caching can be harmful because old pages may never refresh.

## 🔹 Step 3: Configure Cache Backend

Django provides multiple caching backends.

### 1. ✅ **Database Caching**

CACHES = {

    'default': {

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

        'LOCATION': 'student\_cache',  # table name (must NOT match any model name)

        # 'TIMEOUT': 60,  # optional: set per-cache timeout

        'OPTIONS': {

            'MAX\_ENTRIES': 1000

        }

    }

}

### Migration Steps:

python manage.py makemigrations

python manage.py migrate

python manage.py createcachetable  # creates "student\_cache" table

✅ Now, caching data will be stored inside the **student\_cache table** in your database (e.g., SQLite).

### 2. ✅ **File-Based Caching**

CACHES = {

    'default': {

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

        'LOCATION': 'D:/Django5/ch49/student\_cache',  # full path recommended

        'OPTIONS': {

            'MAX\_ENTRIES': 1000

        }

    }

}

* You must create a folder named **student\_cache** in the given path.
* Django will save cache files inside this folder.

### 3. ✅ **Local Memory Caching**

CACHES = {

    'default': {

        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',

        'LOCATION': 'unique-snowflake',  # any unique name

        'OPTIONS': {

            'MAX\_ENTRIES': 1000

        }

    }

}

* This stores cache **in RAM** (per process).
* It’s very fast but **not shared across multiple processes/servers**.

## 🔹 Recap (Comparison)

| **Cache Type** | **Where Stored?** | **When to Use?** |
| --- | --- | --- |
| Database Cache | Database table | When you already use DB and want persistent cache |
| File-Based Cache | File system | Good for small/medium projects, shared across processes |
| Local Memory | RAM (per process) | Fastest but limited; good for development or small apps |

## 🔹 Extra Notes

* **Timeout can be set globally** (CACHE\_MIDDLEWARE\_SECONDS) or per cache ('TIMEOUT' in CACHES).
* **MAX\_ENTRIES** → Maximum number of cached items. When exceeded, old ones are deleted.
* **Browser close expiry** is controlled by session settings, not per-site caching.

# What is caching? (short definition)

A **cache** is a temporary, fast storage layer that holds copies of data or results so future requests can be served faster than re-computing or reading the original (slower) source. Caching increases speed and reduces work for slow resources (databases, APIs, expensive calculations). [TechTarget](https://www.techtarget.com/whatis/definition/caching?utm_source=chatgpt.com)

# Why use caching? (benefits)

* **Faster responses** — serve frequently requested data quickly.
* **Less load on the database / CPU** — reduce repeated expensive queries or computations.
* **Lower latency for users** — pages load faster.
* **Cost savings** — fewer compute/DB operations on cloud resources. [Amazon Web Services, Inc.](https://aws.amazon.com/caching/?utm_source=chatgpt.com)

# Django’s caching — overview

Django has a built-in caching framework that supports different levels of caching granularity:

* **Per-site** cache (cache whole site responses).
* **Per-view** cache (cache the output of a view).
* **Template fragment** caching (cache parts of a template).
* **Low-level cache API** (manually store arbitrary Python objects).  
  You choose these depending on what part of the request/response is expensive to produce. [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)

# Key parts: backends & configuration

1. **CACHES setting (settings.py)**  
   Django uses the CACHES setting to decide where to store cached data. Built-in backends include: database, file-based, local-memory, memcached, and redis (Redis via django\_redis or the RedisCache backend). Pick a backend suitable for your environment (Redis or Memcached for production). [Django Project](https://docs.djangoproject.com/en/5.2/ref/settings/?utm_source=chatgpt.com)

Example (Redis via django-redis):

# settings.py

CACHES = {

"default": {

"BACKEND": "django\_redis.cache.RedisCache",

"LOCATION": "redis://127.0.0.1:6379/1",

"OPTIONS": {

"CLIENT\_CLASS": "django\_redis.client.DefaultClient",

}

}

}

Example (file-based cache — good for small projects / dev):

CACHES = {

"default": {

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

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

}

}

1. **Database cache (if you choose DB backend)**  
   If you use the DatabaseCache, you must create the cache table before using it:
2. python manage.py createcachetable

(Django will create the table named in LOCATION.) [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)

# Types of caching in Django + examples

### 1) Per-view caching

Cache the entire output of a view for a timeout period. Useful for pages that are the same for many users.

# urls.py

from django.views.decorators.cache import cache\_page

from django.urls import path

from .views import product\_list

urlpatterns = [

path("products/", cache\_page(60 \* 15)(product\_list)), # cache for 15 minutes

]

cache\_page wraps the view and stores the rendered response. Subsequent requests within the timeout are served from cache. [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)

### 2) Template fragment caching

Cache only a part of a template (useful when most of a page is static but a small part is dynamic).

{% load cache %}

{% cache 300 sidebar request.user.username %}

<!-- sidebar content that depends on user -->

{% endcache %}

This caches the sidebar for 300 seconds and varies by request.user.username so each user gets their own cached fragment if needed. [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)

### 3) Low-level cache API

Manual cache control — set/get/delete any picklable Python object.

from django.core.cache import cache

# set

cache.set("my-key", {"result": 123}, timeout=3600) # 1 hour

# get

value = cache.get("my-key") # returns the stored object or None

# delete

cache.delete("my-key")

# clear all (use cautiously)

cache.clear()

Use low-level API when you want to cache results of a heavy calculation, external API call, or query result. [TestDriven.io](https://testdriven.io/blog/django-low-level-cache/?utm_source=chatgpt.com)[DEV Community](https://dev.to/pragativerma18/django-caching-101-understanding-the-basics-and-beyond-49p?utm_source=chatgpt.com)

# Managing cache & invalidation

* **Explicit deletion**: cache.delete(key) — you can delete specific keys when data changes.
* **Clear all**: cache.clear() — wipes the whole cache (dangerous in production).
* **Invalidate by key/version**: use key patterns or add versioning/prefixes to keys to avoid accidental collisions.
* **Template keys**: you can compute/inspect template fragment keys with django.core.cache.utils.make\_template\_fragment\_key(...). [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)[DEV Community](https://dev.to/pragativerma18/django-caching-101-understanding-the-basics-and-beyond-49p?utm_source=chatgpt.com)

**Sessions note:** if you use DB-backed sessions, Django does not automatically purge expired sessions — use the management command clearsessions (run by cron/celery) or call SessionStore.clear\_expired(); otherwise session table can grow. [Django Project](https://docs.djangoproject.com/en/5.2/topics/http/sessions/?utm_source=chatgpt.com)

# Best practices & common pitfalls (important for notes)

* **Pick the right backend**:
  + Use **Redis** or **Memcached** in production (fast, shared between processes).
  + locmem (local-memory) is **per-process** and not shared across workers — okay for development but not for multi-worker production. [Django Cachalot](https://django-cachalot.readthedocs.io/en/v2.4.1/quickstart.html?utm_source=chatgpt.com)[Django Project](https://docs.djangoproject.com/en/5.2/ref/settings/?utm_source=chatgpt.com)
* **TTL (timeout)**: pick a sensible timeout — too long → stale data; too short → low hit rate.
* **Key design**: include app name/version/user-id where necessary. Avoid huge keys; keep them unique and simple.
* **Don’t cache sensitive user data** (auth tokens, PII) unless encrypted and you know what you’re doing.
* **Cache stampede**: when many requests miss the cache at once, the origin can be overloaded. Mitigate with mutexes, jittered TTLs, or cache pre-warming.
* **Memory limits & eviction**: memcached/redis evict using LRU policies; monitor hit-rate and evictions.
* **Monitoring**: watch metrics (hit rate, evictions, memory usage) and logs to tune caches. [sitepoint.com](https://www.sitepoint.com/django-caching-comprehensive-guide/?utm_source=chatgpt.com)

# Quick cheat-sheet (for your notes)

* Add caching: set CACHES in settings.py. [Django Project](https://docs.djangoproject.com/en/5.2/ref/settings/?utm_source=chatgpt.com)
* Per-view: @cache\_page(60 \* 15) or cache\_page(...) in urls.py. [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)
* Template fragment: {% load cache %} {% cache 300 name %} ... {% endcache %}. [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)
* Low-level API: from django.core.cache import cache; cache.set/get/delete(...). [TestDriven.io](https://testdriven.io/blog/django-low-level-cache/?utm_source=chatgpt.com)
* Database cache: set DB backend and run python manage.py createcachetable. [Django Project](https://docs.djangoproject.com/en/5.2/topics/cache/?utm_source=chatgpt.com)
* Clear expired sessions: python manage.py clearsessions (run regularly).