Engineer

Django

D a Python framework for building web applications

What it is

Django is a web framework written in Python: a request comes in, the URL configuration routes it to a function or class called a view, and the view returns a response, usually HTML rendered from a template or JSON. Around that core it ships the parts most database-backed sites need, among them an object-relational mapper, schema migrations, forms, authentication with permissions, and an automatically generated admin interface that its own documentation calls one of its most popular features. The project is written with a capital D, while the package you install and import is the lowercase `django` .

It came out of a newsroom. In fall 2003 Adrian Holovaty and Simon Willison, developers at World Online, the web department of a newspaper in Lawrence, Kansas, moved from PHP to Python for the sites they built on journalism deadlines. While building sites such as Lawrence.com they extracted a generic framework from the work, refined it for two years, and in summer 2005 World Online released it as open source. It is named after Django Reinhardt, the jazz manouche guitarist, is pronounced JANG-oh with a silent D, and is distributed under the 3-clause BSD licence.

Under the A.B numbering, feature releases came roughly every eight months, and only designated long-term support releases received three years of support; 5.2 LTS is supported until April 2028. Django 6.2, an LTS release due in April 2027, is the last one under that policy. From Django 2028 on, feature releases are named for their year, arrive every January, and each receives the same three-year support period.

The project’s FAQ suggests calling Django an MTV framework, for model, template and view: the model defines the data, the view decides which data the user sees, and the template decides how it looks. The controller, in that reading,

is the framework itself, the machinery that sends each request to the right view. Models are Python classes, and queries against them are QuerySets, which are lazy: a QuerySet can be built, filtered and passed around without touching the database, which is queried only when something evaluates it, such as a loop. Changes to models become migrations: `makemigrations` packages each change into a file that the documentation compares to a commit in a version control system for the schema, and `migrate` applies it. Related objects load on demand, one query at a time, unless the QuerySet asks for them up front.

```
# Book has a ForeignKey to Author.
# 1 query for the books, then 1 per book for its author.
for book in Book.objects.all():
    print(book.author.name)

# 1 query: an SQL join loads each author with its book.
for book in Book.objects.select_related("author"):
    print(book.author.name)

# 2 queries: the authors, then all their books in one batch.
for author in Author.objects.prefetch_related("book_set"):
    print(author.name, [b.title for b in author.book_set.all()])
```

Use the concept in a decision

Put “What is Django?” to work in a real decision. We can help relate the concept to the system you are building or buying. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Why it matters

Django matters because of what it makes the default. Its QuerySets build SQL with the parameters kept apart from the query text, so values a user supplies are escaped by the database driver. Its templates escape the characters that are most dangerous in HTML. Its CSRF protection checks a secret on every POST, and it stores passwords with PBKDF2 over SHA-256, a stretching method its documentation notes is recommended by NIST. A team writing ordinary Django code gets those protections without asking for them.

The same defaults set the shape of the risk. Each can be undone by one line, such as a raw query, a `mark_safe` , a `csrf_exempt` or `DEBUG = True` left on the live server, and each of those lines looks harmless in review. The ORM makes a similar trade with performance: it is easy to write a view whose query count grows with the length of the page, and the Python reads the same either way.

Our recommendation is to choose Django when the application is shaped by its data: records that people create and edit, accounts with permissions, and staff who need an admin screen over all of it. Keep each project on a supported release, choose an LTS while the A.B releases last, and write the end of support next to the version. Run `manage.py check --deploy` against the production settings in the pipeline, keep the secret key and the database password out of the repository, and read the SQL the ORM writes for every page that matters. For a site the client will mostly edit as pages, a content management system or static pages fit better than a framework.

What it looks like in practice

From 2015 to 2022, as a web developer, the work included Django web applications around an IPTV platform, the parts of a round-the-clock service that people actually clicked on: new features, bug fixes, and steady work on performance and stability, done across the team. On a service like that, the page says, stability is the constraint the features have to respect, and the applications kept gaining capability while staying reliable for internal teams and customers.

Earlier, from 2003 to 2014, as a full stack web developer, a hundred web applications were built for clients who wanted very different things. Where a client needed something custom, the tools were HTML/HTML5, CSS/SCSS and Django; where they needed something more standard that they could also manage themselves, WordPress or Joomla was the faster, more sensible answer. The page names the skill as knowing which was which, including talking a client out of a bespoke build they did not need, or into one they did. That is the same line our recommendation draws.

The tech stack page keeps both halves of that. Its web tier is static-first, with Django and PHP where there is an application to serve, and its list of languages treats SQL as a first-class language rather than something an ORM generates, because most of the performance work we are called in for lives there. For a Django application, that means the ORM writes the queries and someone still reads them.

Where it goes wrong

- **A loop runs one query per row.** In the default fetch mode, reading a foreign key that the QuerySet did not load runs a query for that one object, so a list of a hundred books with their authors costs 101 queries. Templates make it harder to see, because they call methods such as `all` without parentheses. Use `select_related()` for foreign keys and one-to-one fields and `prefetch_related()` for many-to-many fields and reverse foreign keys. On Django 6.1 and later, the `FETCH_PEERS` mode batches those lookups into one extra query, and `FETCH_RAISE` turns an unplanned query into an exception in code that has to stay fast.
- **Debug mode reaches production.** The deployment checklist says never to enable `DEBUG` in production, because its error pages leak excerpts of source code, local variables, settings and the libraries in use. With `DEBUG = False`, Django will not serve requests at all until `ALLOWED_HOSTS` holds a suitable value, and a wildcard there means validating the Host header yourself. Run `manage.py check --deploy` against the production settings file, which automates part of that checklist.
- **The secret key sits in the repository.** `SECRET_KEY` protects sessions under most session backends, every password reset token and all cryptographic signing that names no other key, and the settings reference warns that a known key can lead to privilege escalation and remote code execution. Load it from the environment or a file, keep it out of version control, rotate it through `SECRET_KEY_FALLBACKS`, and remove old keys from that list promptly.
- **A safe default is switched off for one view.** QuerySets are parameterised, but raw queries, `extra()` and `RawSQL` leave escaping to the developer; templates escape HTML, but `mark_safe`, the `safe` filter and turning autoescape off hand that job back too; `csrf_exempt` removes the check from a view entirely. Each one deserves a stated reason in review, and HTML that users store should be sanitised on input as well as output.
- **A migration stops halfway.** On PostgreSQL and SQLite, every operation in a migration runs inside one transaction by default, but MySQL has no transactions around schema changes, so a migration that fails partway leaves changes to unpick by hand. Read what `makemigrations` thinks changed, check the SQL with `sqlmigrate` before it reaches production, and commit each migration together with its model change.

Further reading

- Django documentation — the current stable release's documentation, from the tutorial to the reference.
- Django: Deployment checklist — the settings to change before going live, part of which `check --deploy` automates.

- Django: Database access optimization — QuerySet evaluation, fetching related objects and the habits that keep query counts flat.
- Django: Migrations — the migration workflow, `sqlmigrate`, and how each database engine handles schema changes.
- Security in Django — what the framework protects against by default and where that protection stops.
- Download Django — the supported release series with their mainstream and extended support dates.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Django?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/django/