HyperionDev

Lecture - Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- No question is daft or silly - **ask them!**
- There are Q&A sessions midway and at the end of the session, should you wish to ask any follow-up questions.
- Should you have any questions after the lecture, please schedule a mentor session.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. Django Models
2. Django Views
3. Integration with Django Templates
4. Best Practices and Tips
5. Wrap Up

HTTP Methods Overview

HTTP (Hypertext Transfer Protocol): allows web browsers to interact with servers.

- Common Methods:
  - GET
  - POST
  - PUT
  - DELETE

- Different HTTP methods define how data is requested or sent.

GET Method

GET: Retrieving Data

- **GET** requests retrieve data from the server (e.g., loading a webpage).
- It asks the server for information and waits for a response.
- **Example**: Visiting a website loads the webpage using GET.

POST Method

POST: Sending Data

- **POST:** requests send data to the server (e.g., submitting a form)**.**
- It sends information that the server processes and responds to.
- **Example**: Signing up for an account online uses POST to submit your data.

PUT Method

PUT: Updating Data

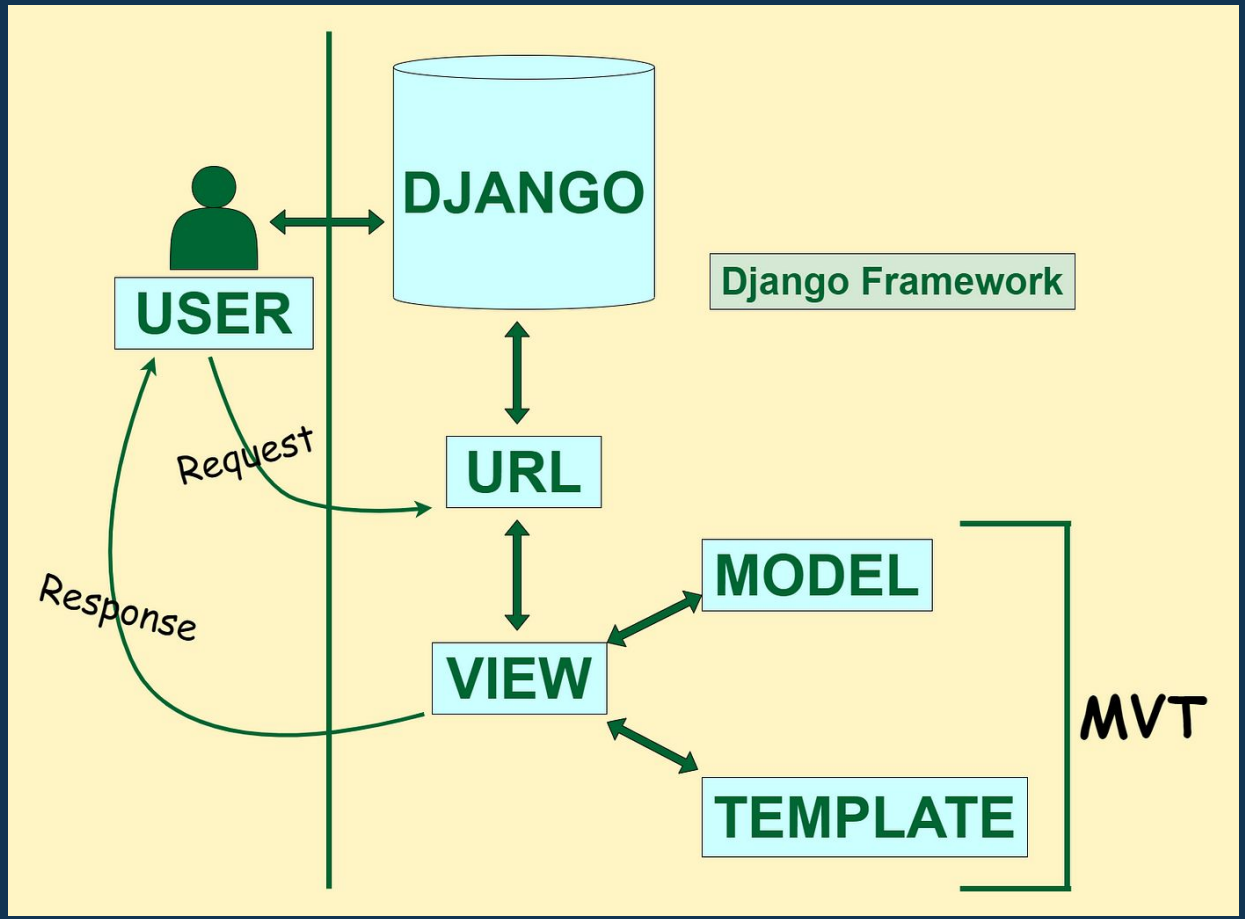
- **PUT** requests update existing data on the server (e.g., updating your profile picture).
- It sends new information to replace old data.
- **Example**: a browser replacing an old file with a new one.

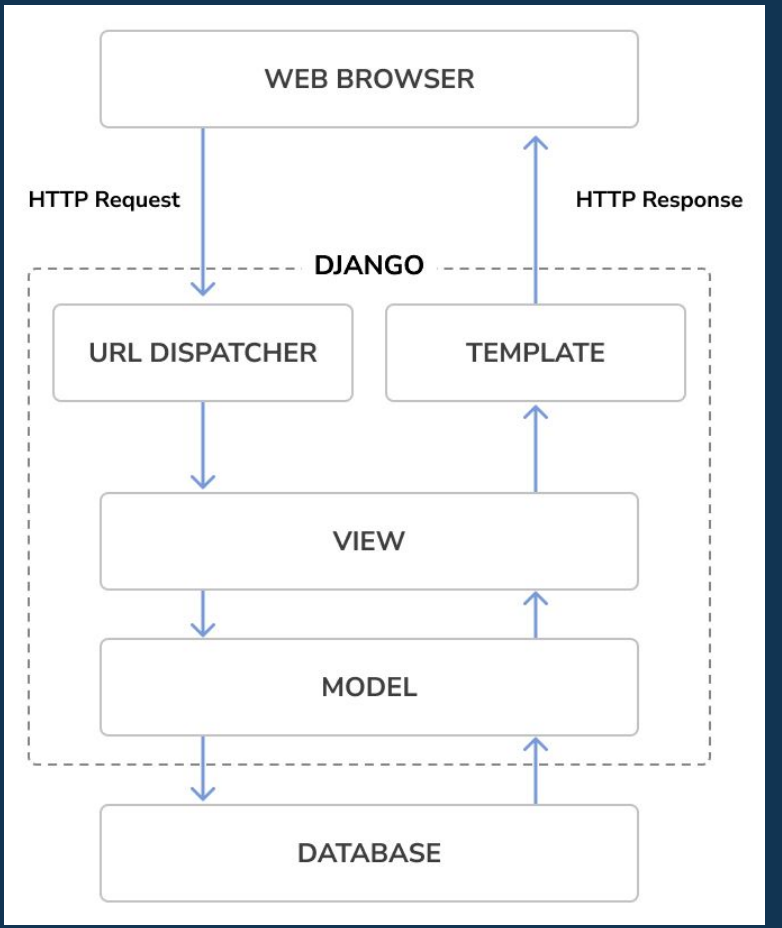
DELETE Method

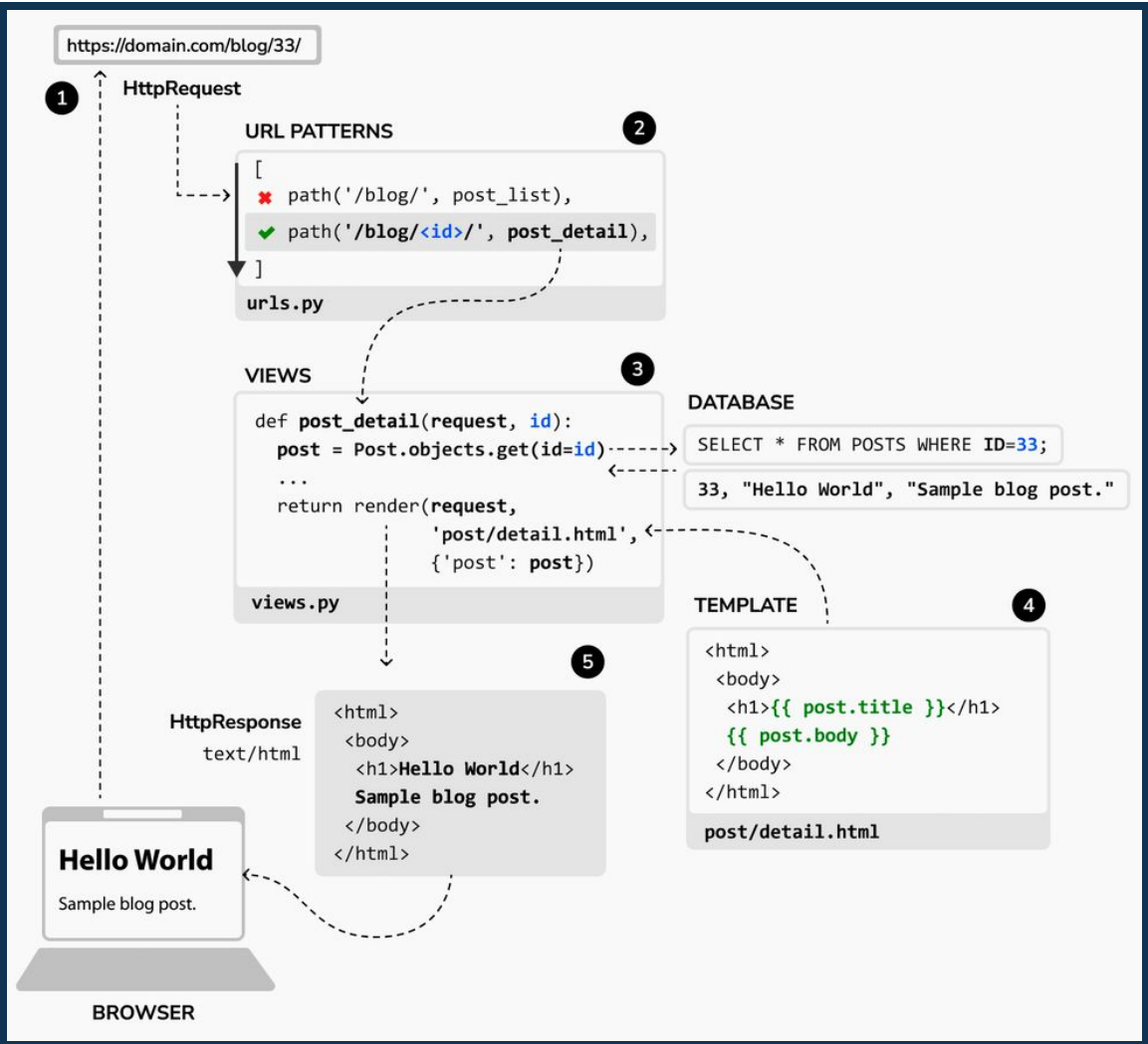
DELETE: Removing Data

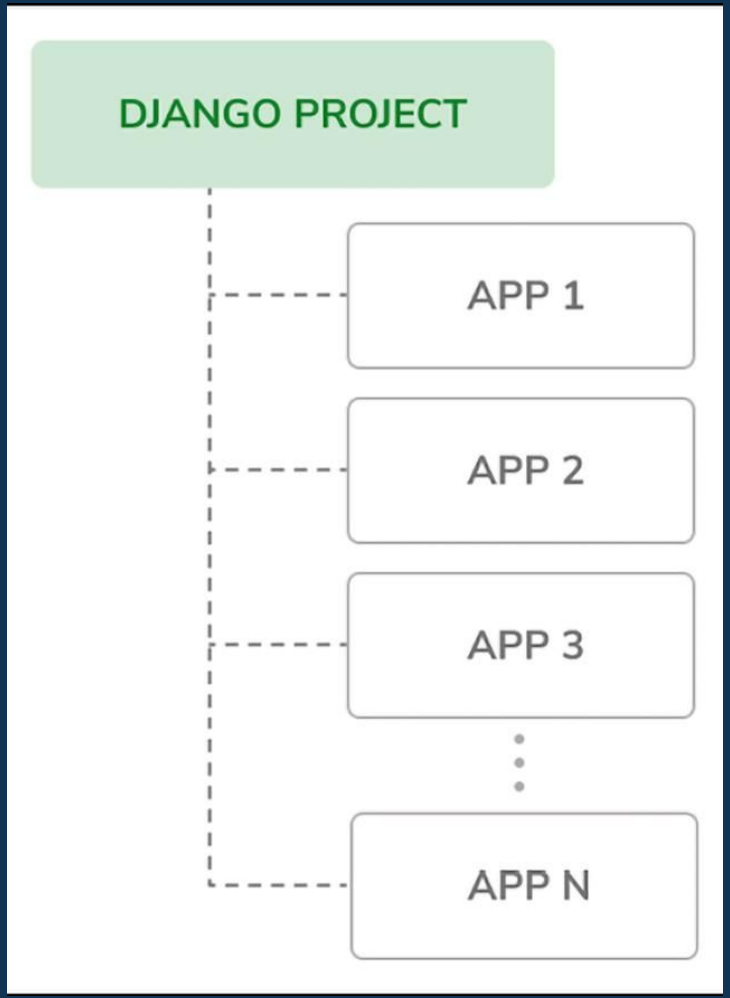
- **DELETE** requests remove data from the server (e.g., deleting a comment).
- It tells the server to delete the specified resource.
- **Example**: Like throwing away an old document that's no longer needed.

MVT

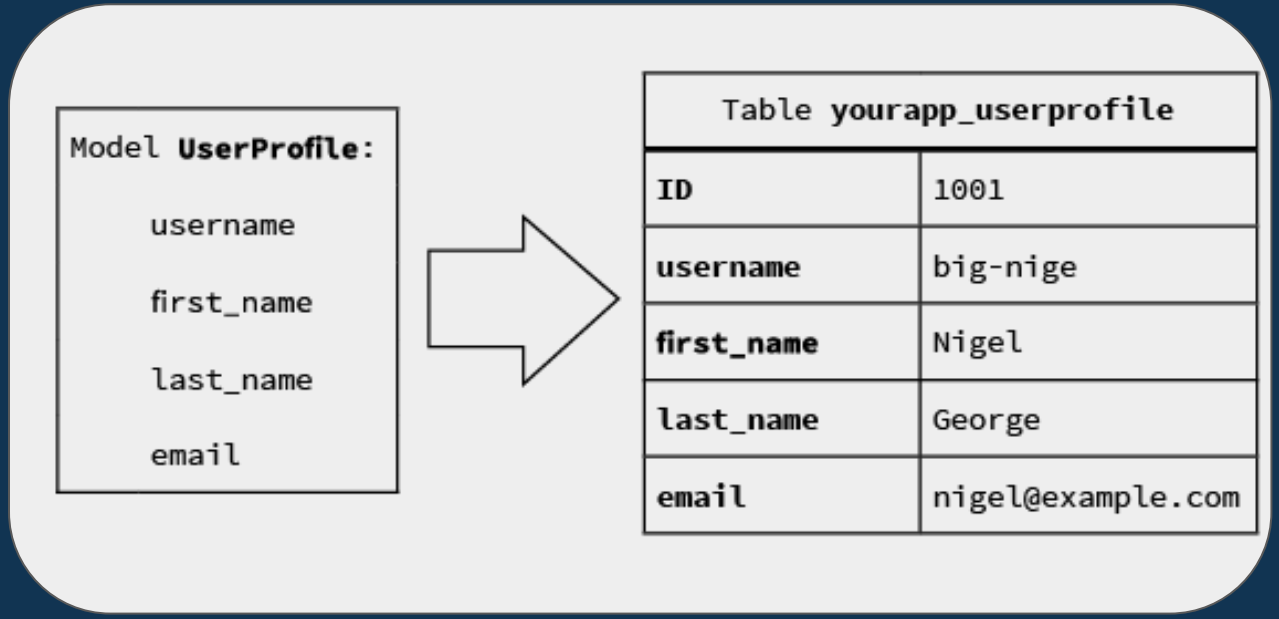

DJANGO
Django Framework
USER
Request
URL
Response
MODEL
VIEW
TEMPLATE
MVT


WEB BROWSER
HTTP Request
HTTP Response
DJANGO
URL DISPATCHER
TEMPLATE
VIEW
MODEL
DATABASE


https://domain.com/blog/33/
1
HttpRequest
URL PATTERNS
2
[
path('/blog/', post_list),
path('/blog/<id>/', post_detail),
]
urls.py
VIEWS
3
def post_detail(request, id):
post = Post.objects.get(id=id)
...
return render(request,
'post/detail.html',
{'post': post})
views.py
DATABASE
SELECT * FROM POSTS WHERE ID=33;
33, "Hello World", "Sample blog post."
TEMPLATE
4
<html>
<body>
<h1>{{ post.title }}</h1>
{{ post.body }}
</body>
</html>
post/detail.html
5
HttpResponse
text/html
<html>
<body>
<h1>Hello World</h1>
Sample blog post.
</body>
</html>
Hello World
Sample blog post.
BROWSER

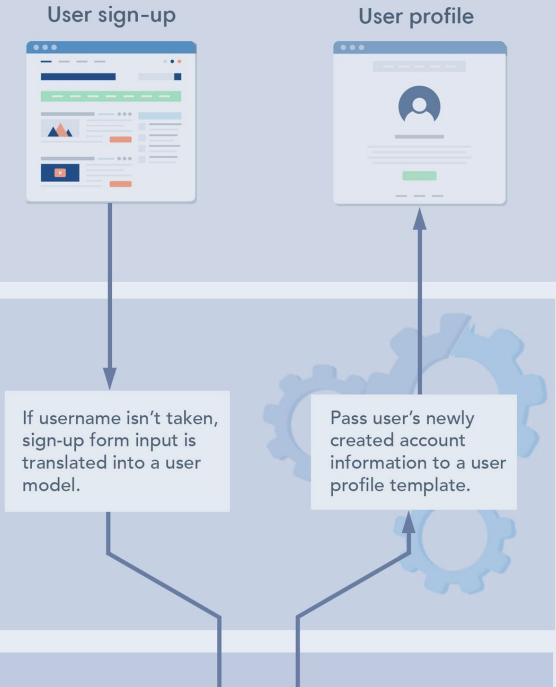

DJANGO PROJECT
APP 1
APP 2
APP 3
APP N

What is a Model?



TEMPLATE

Our app's presentation layer. This is what users actually see, and perceive to be the entirety of our app. Our presentation layer should only serve two purposes: accepting user input/actions, and displaying output generated by the "brains" of our application.



VIEW

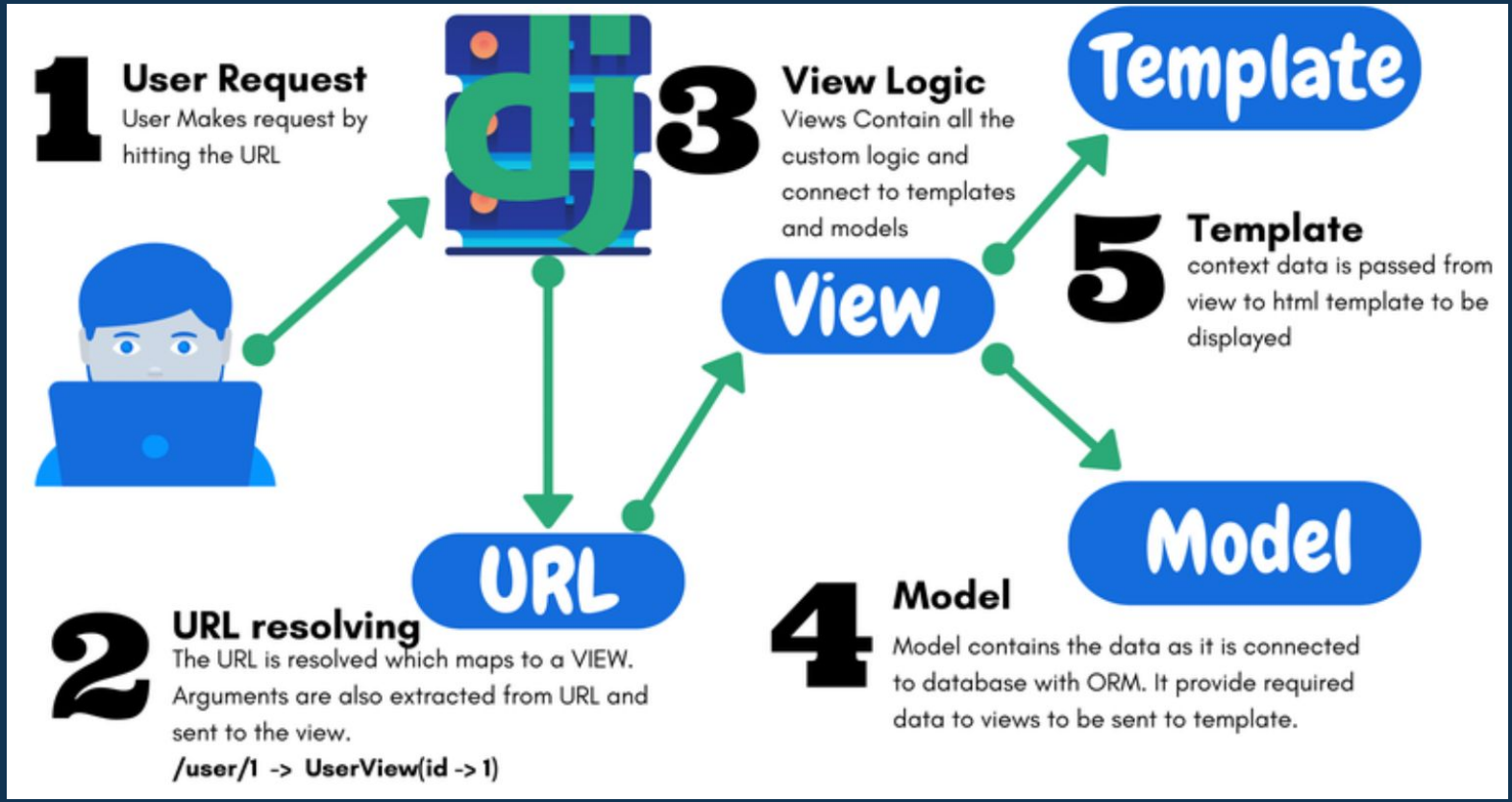
Core business logic used to interpret user inputs or construct the the proper output to be served by a user's request. Views are responsible for intepreting user requests, transforming/fetching data associated with their requests, and passing along the proper response to a template which communicates the outcome.

MODEL

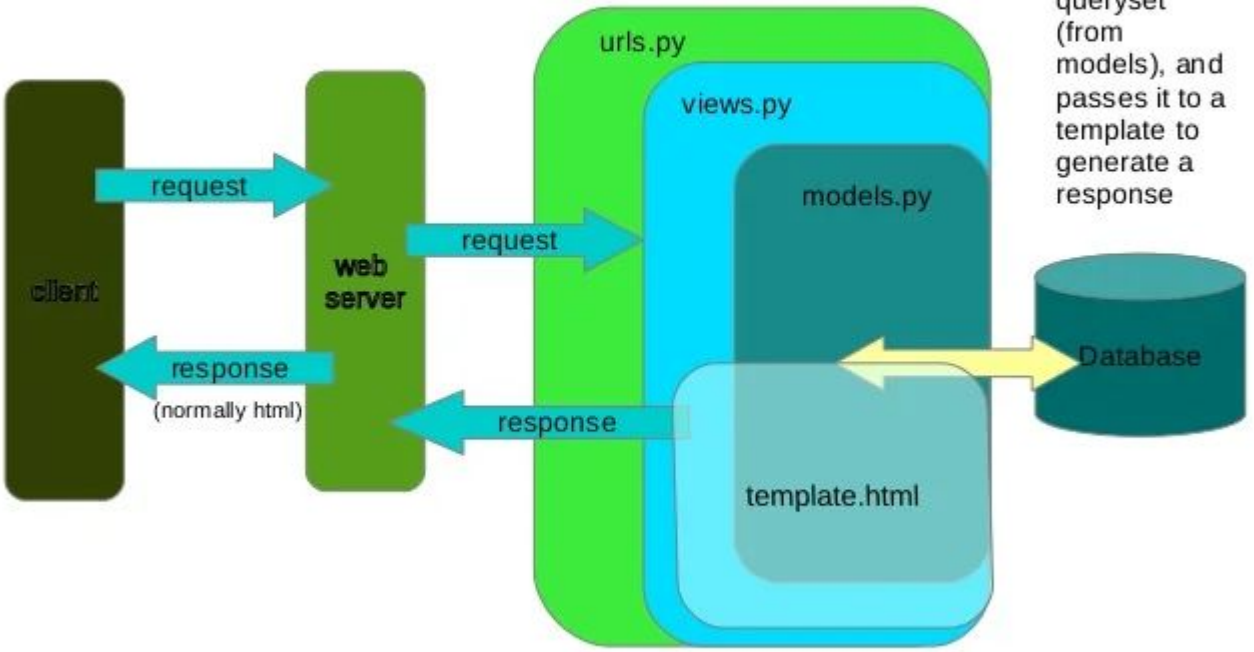
The data layer where activity in our application ultimately creates, updates, deletes. or retrieves records from our app's database. All data in an MVC framework is typically abstracted into objects known as "models." For example, users might be represented by a class named User which contains attributes like name, email, and so forth.

Hyperiondev

Views in MVT


1
User Request
User Makes request by hitting the URL
3
View Logic
Views Contain all the custom logic and connect to templates and models
Template
5
Template
context data is passed from view to html template to be displayed
View
URL
Model
2
URL resolving
The URL is resolved which maps to a VIEW. Arguments are also extracted from URL and sent to the view.
/user/1 -> UserView(id -> 1)
4
Model
Model contains the data as it is connected to database with ORM. It provide required data to views to be sent to template.

Django - overview

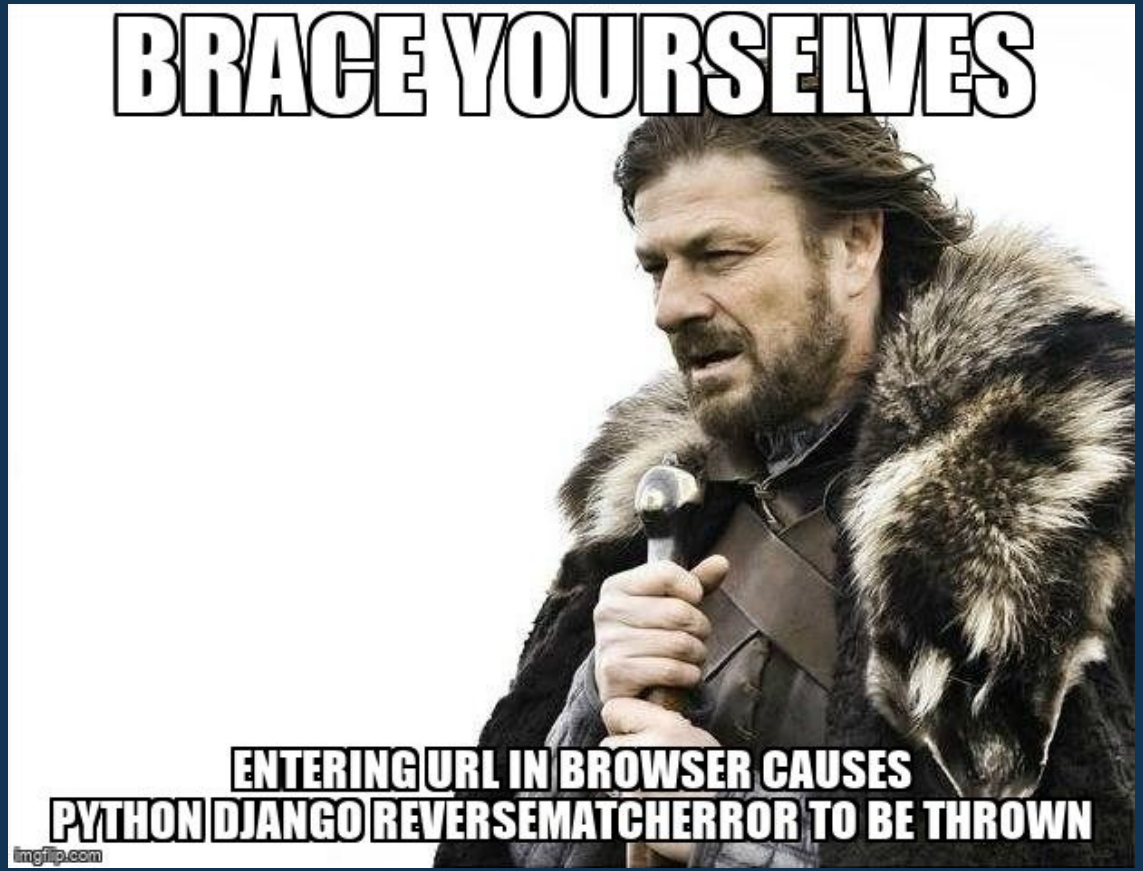


A view usually generates a queryset (from models), and passes it to a template to generate a response

Best Practices

- Adhere to naming conventions for files, classes, and variables. Use descriptive names that convey the purpose of the components.
- For app names, use lowercase with underscores (e.g., `myapp_name`) and follow the Python naming convention (e.g., `snake_case`).
- Implement a clear separation of concerns in your application. Models should handle data and database interactions, Views should manage HTTP requests and responses, and Templates should focus on presentation.
- Break your project into smaller apps, each with its own specific functionality. This modularity makes code more manageable and facilitates code reuse.
- Create reusable components within your project. For example, develop custom middleware or utility functions that can be shared across multiple apps.
- Maintain a consistent code style throughout the project by adhering to PEP 8 (Python Enhancement Proposal 8) and adhering to Django's coding style recommendations.

Wrapping up


BRACE YOURSELVES
ENTERING URL IN BROWSER CAUSES
PYTHON DJANGO REVERSEMATCHERROR TO BE THROWN
imgflip.com

Questions and Answers

Questions around Defensive Programming