Django pt. 2: MVC, Database, and Auth

Kianoosh Abbasi

CSC309 Winter 2022

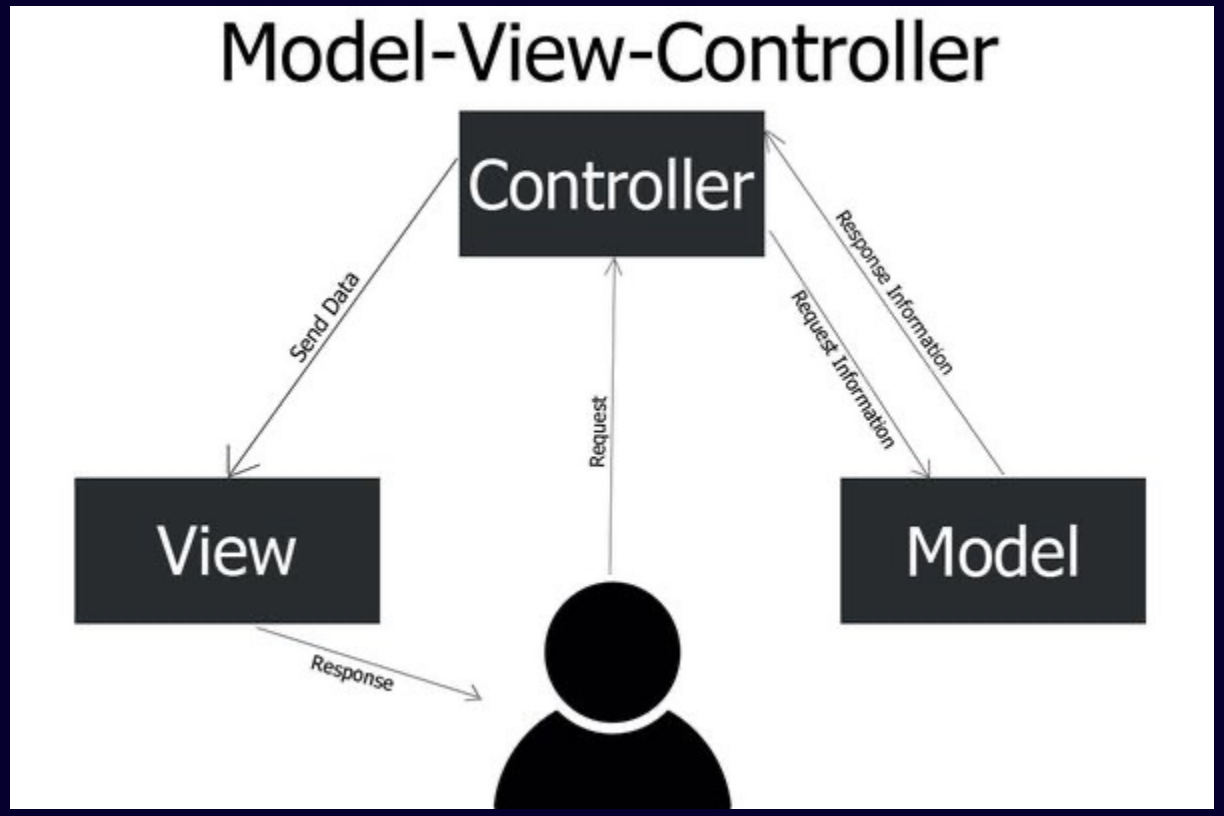
So far

- How web works, HTML, CSS
- Backend vs Frontend development
- Python projects
  Virtual environment & pip
- Django
  Setup, simple views, forms, templates

This session

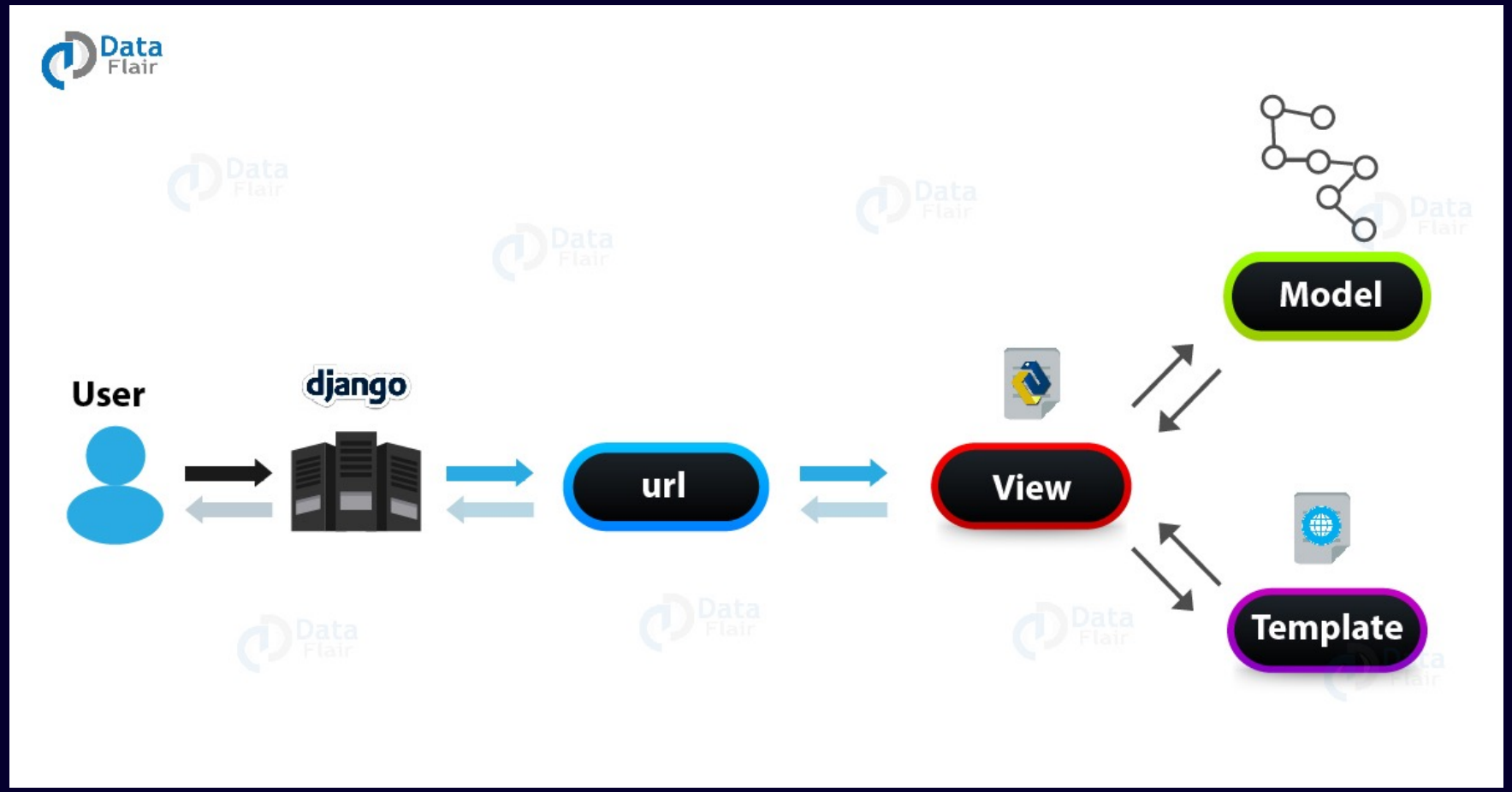
- MVC Design patterns
- Working with a database
  ORM and models
- Authentication
  User model, sessions, login
- Admin panel

MVC



Source: https://www.quora.com/Is-Django-an-MVC-or-an-MVT-framework

Django's architecture



Source: https://data-flair.training/blogs/django-architecture/

Django's architecture

- Almost an MVC framework
- Naming differences
  - Django's view: MVC's controller
  - Django's template: MVC's view
- Parts of controller already implemented by framework
  - URL dispatcher
- Django templates are more than just presentation
  - Capable of adding some logic

Django template language

- Data passed through context

```
context={'errors': ['err1', 'err2']}
```

- For loop

```
{% for error in errors %}
   <h3> {{ error }} </h3>
{% endfor %}
```

- Attribute, method, dictionary lookup, or index access

```
{{ errors.0 }} {{ request.GET }} {{ mydict.items }}
```

Django template language

- If statement

```
{% if errors %}
   <h3> There has been errors </h3>
{% endif %}
```

- Filters

```
{{ errors|length }} {{ name|default="TBD" }}
```

- More tags (technically functions)

```
{% url 'accounts:hello' %} {% csrf_token %}
```

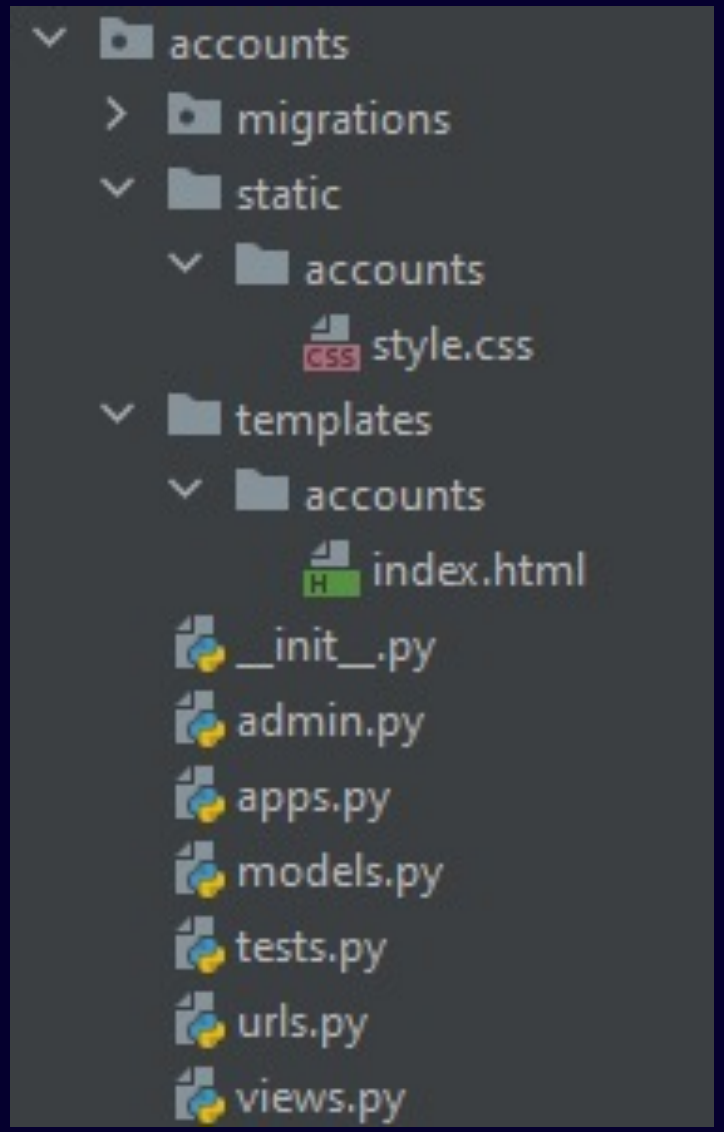
Static files

Visit https://docs.djangoproject.com/en/3.2/howto/static-files/

- The html response is dynamic
  - Created at each request, after template language is compiled

- Other files are static
  - CSS, JavaScript, images

- Put static files under static directory of each app
  Standard is to put them under a subdirectory with the app's name
- separate css, js, img
- Access them at template
  … `href="{% static 'accounts/style.css' %}"`
  Add `{% load static %}` to the top
- Translated to /static/accounts/style.css



Static files

- Some static files don’t really belong to an app
  Global style, bootstrap, font, etc.

- Custom directories for static files
  Usually a root static folder in the project

- At settings.py

```
STATICFILES_DIRS = [
    BASE_DIR / "static"
]
```

Root templates (optional)

- Can templates not belong to an app as well?
- Templates generally map to a single view
  - Views reside in apps
- But some parts of templates can be the same
  - Navigation bar, footer, metadata, etc.

Template inheritance

- Templates can inherit from a parent template!
- Use `{% extends 'base.html' %}`
- Parent is served

  Everything else in child template is ignored
- Except for overriden blocks

Blocks

- Wraps a piece of code in template

  `{% block myblock %} … {% endblock %}`

- The same block at child template overrides parent's

- Can extend parent's block with `{{ block.super }}`

base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    {% block staticfiles %}
        <link rel="stylesheet"
              href="{% static 'css/bootstrap.min.css' %}" />

        {# Add more static files at child tempaltes #}
    {% endblock %}
    <title>
        {% block title %}
            Home page
        {% endblock %}
    </title>
</head>
<body>
    {% block content %}
        <h1>This is the base template</h1>
    {% endblock %}
</body>
</html>
```

login.html

```
{%  extends 'base.html' %}
{% load static %}

{% block title %}
    Login
{% endblock %}

{% block staticfiles %}
    {{ block.super }}
    <link rel="stylesheet"
          href="{% static 'accounts/css/login.css' %}" />
{% endblock %}

{% block content %}
    <h1>Login</h1>
{% endblock %}
```

Root template

- A base.html will probably help
  - load base js, css files
  - Navigation bar
  - Footer
  - General layout
- This file might not belong to any app
- Like static files, create a root template folder
  - Add `BASE_DIR / "templates"` to `DIRS` under `TEMPLATES` in settings.py

Database, ORM, and Models

- We have not stored/read data so far!
- Every web application needs a persistent storage
- Many different databases are around
  - Relational: Postgres, mySQL
  - Non-relational: Cassandra, MongoDB
- Django supports various database backends

Do we need Django's support?

- Technically, we can make a connection to any database and run queries

- But this is a terrible idea!

  WHY?

- How can the framework/language help us out?

Object Relational Mapper

- Provides an abstraction over the underlying database queries
- Method/attribute accesses are translated to queries
- Results are wrapped by objects/attributes

Object Relational Mapper

- Simplicity: No need to use SQL syntax
- Consistency: Everything is in the same language (Python)
- Can switch database backend easily
- Enables Object Oriented Programming
- Runs a secure efficient query
  SQL injection, atomicity, etc.
- But, for super-efficient queries, you might still need to run raw queries

SQLite

- Django's default database backend
- Light-weight database that stores everything in one single file
- Follows standard SQL syntax
- Great option for development: no setup/installation required
- For production, switch to a more sophisticated database

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Models

- Represents, stores, and manages application's data
- Typically, a table in the database
- Thanks to ORM, models can be defined as classes
- Django has some pre-defined models
- User: Django's default model for authentication and authorization

Authentication vs Authorization

- Authentication:
  + Who's calling?
  - This is Daniel Zingaro
  + Is it really Daniel Zingaro?
- Obtains user information from user/pass, session, API key, etc.

- Authorization:
  Does Daniel Zingaro have enough access and permissions (aka authorized) to make this request?
- Checks user's properties and permissions

User

- Pre-defined fields:
  username, password, first_name, last_name, email, etc
- Raw passwords must never be saved to database
- Considerations:
  Username is case-sensitive!
  Emails don’t have to be unique!

Creating a user

- Initially, database is empty and has no table
  - Even Django's pre-defined tables
- To add/updates tables, you must migrate the database
  - Run `./manage.py migrate`
- More on migrations next session

Creating a user

- Create a user via ORM

```
User.objects.create_user(username='dan1995', password='123',
first_name='Daniel', last_name='Zingaro')
```

- Access user(s)

```
users = User.objects.all()
dan = User.objects.get(username='dan1995’)
active_users = User.objects.filter(is_active=True)
```

- Delete user(s)

```
User.objects.all().delete()
dan.delete()
```

Working with the ORM

- Every models has an objects attribute
  Handles database queries
- filter and get both run select statements
- get returns exactly one object
  Throws an exception if zero, two, or more objects are returned
- filter returns a list of objects (more precisely, a queryset)

Querysets

- Evaluated lazily

  Queries not run until really needed

- This example only runs one query:

```
users = User.objects.all()
users2 = users.filter(is_active=True)
users3 = users2.filter(username__contains='test')
user = users3.get()
user.get_full_name()
```

Update queries

- Update a queryset

```
User.objects.filter(is_active=True).update(is_active=False)
```

- Update a single instance

```
dan = User.objects.get(first_name='Daniel')
dan.first_name = 'Dan'
dan.save()
```

- Attributes are locally cached values

  To refresh: `dan.refresh_from_db()`

Exercise: Extend the signup form to actually create a user

Authentication

- Client should tell us who they are
- Via Authorization header in HTTP
- Several authentication methods
  - Password auth
  - Session auth
  - Token auth

Basic (password) auth

- Simply sends username and password at every request
- No concept of login and logout
- Unencrypted base64 strings
- So insecure: transfers raw password this many times

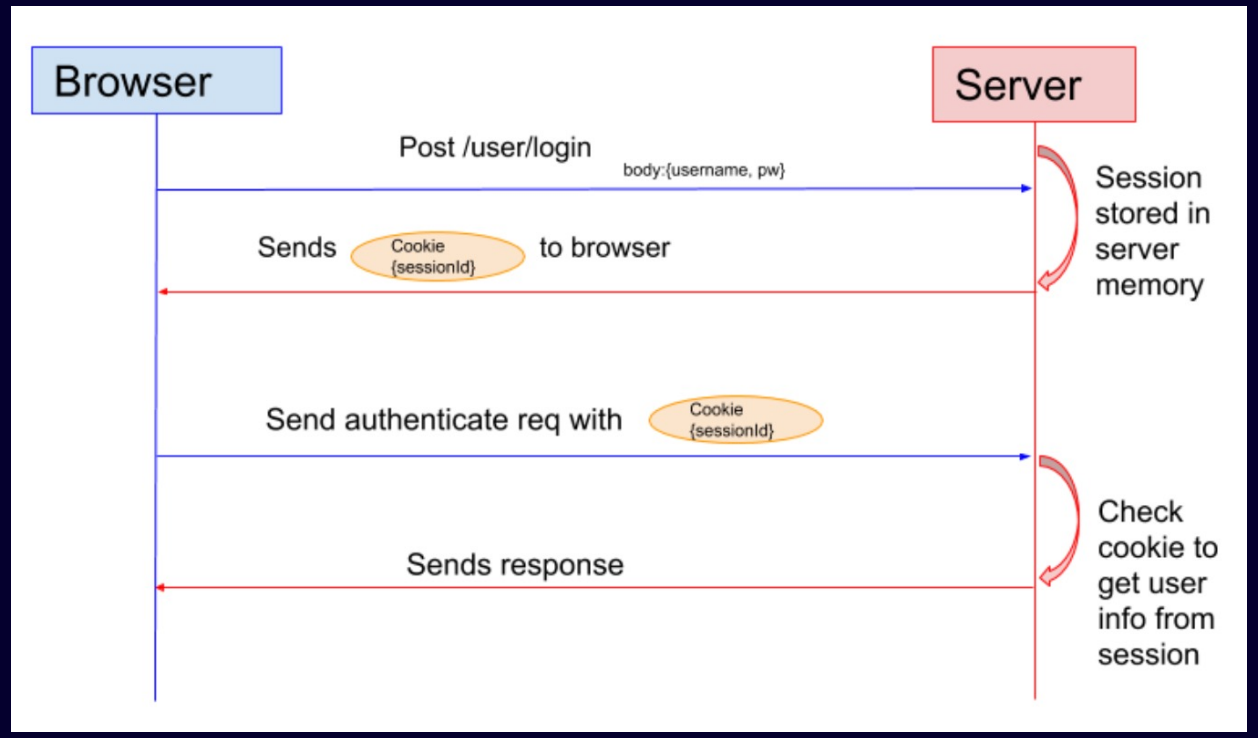
Session auth

- Client sends user/pass at login
- If successful, server creates and stores a session id
  Mapped to user
- Session id returned in the response
  Browser saves it in cookies
- Browsers sends the same session id at next requests
  Incognito tab: browser does not send the same session id

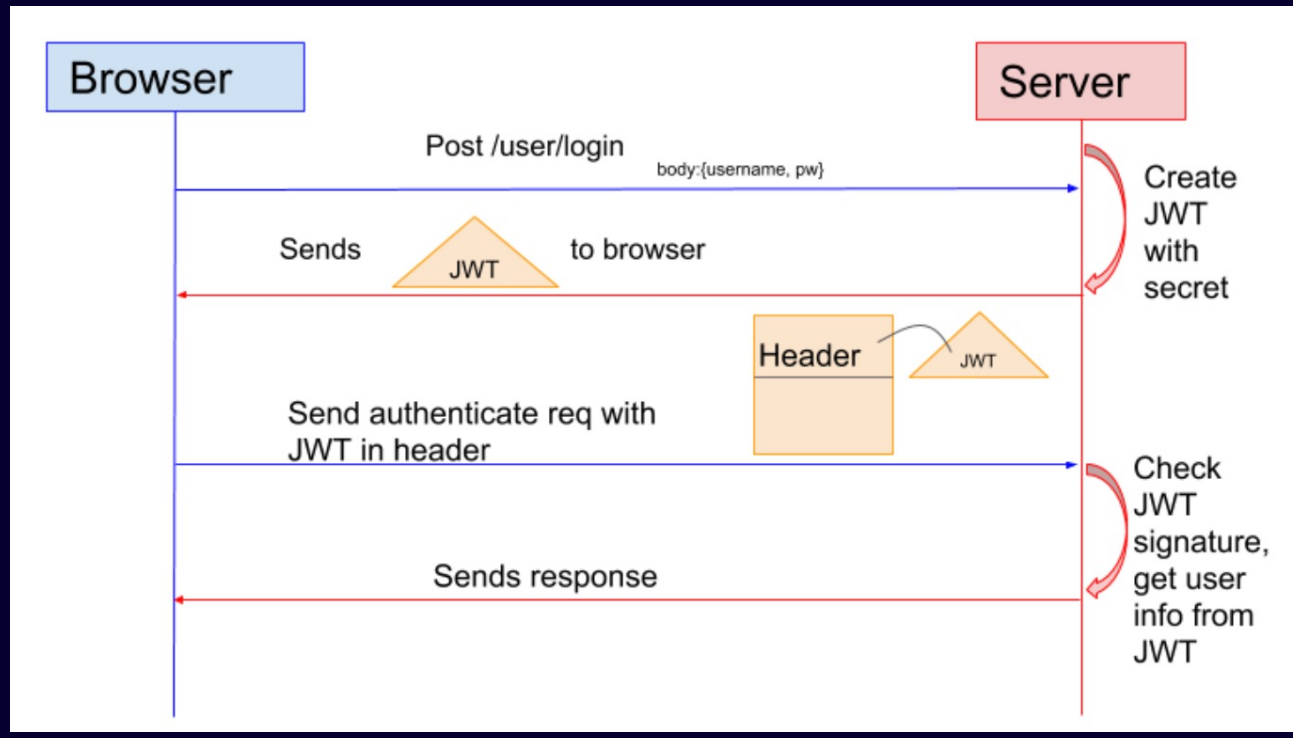
Token auth

- Storing every single session could be an overhead
  Limits the scalability of the application
- Instead of a random session id, the token can contain information about the user
- Must be signed by the server to avoid attacks

Session auth



Token auth



Source: https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4

Django's session auth

- Check user/pass combination is right

```
user = authenticate(username='john', password='secret')
```

- Django's login function: attaches user to the current session

```
login(request, user)
```

- Django does the session id lookup itself
  - User object accessible at `request.user`
  - `AnonymousUser` if unauthenticated

- logout function: removes session data

Admin panel

- A very convenient medium to see/change database records
  - Instead of running raw queries or python code at `./manage.py shell`
- The admin url at urls.py
- Needs an active user with `is_superuser` and `is_staff True`
  - Can be created manually through the shell
  - Or via command: `./manage.py createsuperuser`

This session

- MVC Design patterns
- Working with a database
  ORM and models
- Authentication
  User model, sessions, login
- Admin panel

Next session

- Create custom models
- Migrations
- Advanced viewed: Class-based views and CRUD

Final notes

- Project phase 1 deadline in one week
- Sign up for mentor meetings!
- Practice Django at home
  https://docs.djangoproject.com/en/4.0/contents/