Section 7

Django

materials at jrsacher.github.io/web50

Outline

- Project 1 post-mortem
- Project 2 notes/questions
- Project 3 details
- Django

Project 1

- Grades/comments will be released before Project 2 is due
- Some common issues
  - Lack of comments
  - Lack of error handling
  - Focusing on looks before specifications

Project 2

- Due Monday 7/22 at noon
- Advice
  - Make liberal use of `console.log()` and `print()` in debugging
    - remove before submitting
  - Spend time thinking about how to best store channel/message info
  - Don't try to make everything work at once
    - Break problem down into small pieces, then break those down as well
    - If stuck, try writing out pseudocode as comments, then filling in actual code
  - Use the `readme.md` file to your advantage
  - Plan for a user that's trying to break things

Project 3

- Last structured project!
- All critical material was covered in Lecture 7
- Important to stay organized, work in small steps
- Spend time planning, diagramming, etc.
  - Especially your DB!
- Due Wednesday 7/31 at noon (but don't wait!)

A note on style

Some places to start:

- PEP 8, Python's official style guide
- AirBnB and Google JS style guides
- W3Schools and Google for HTML

Style is not just looks!

- appropriate, meaningful variable names
- appropriate comments in the code
- etc.

Django

Flask

- "Microframework"
- Lightweight
- SQLAlchemy needed for DBs
- Single application per project
- Jinja2 for templating
- Users
  - Netflix
  - Reddit
  - Lyft AND Uber

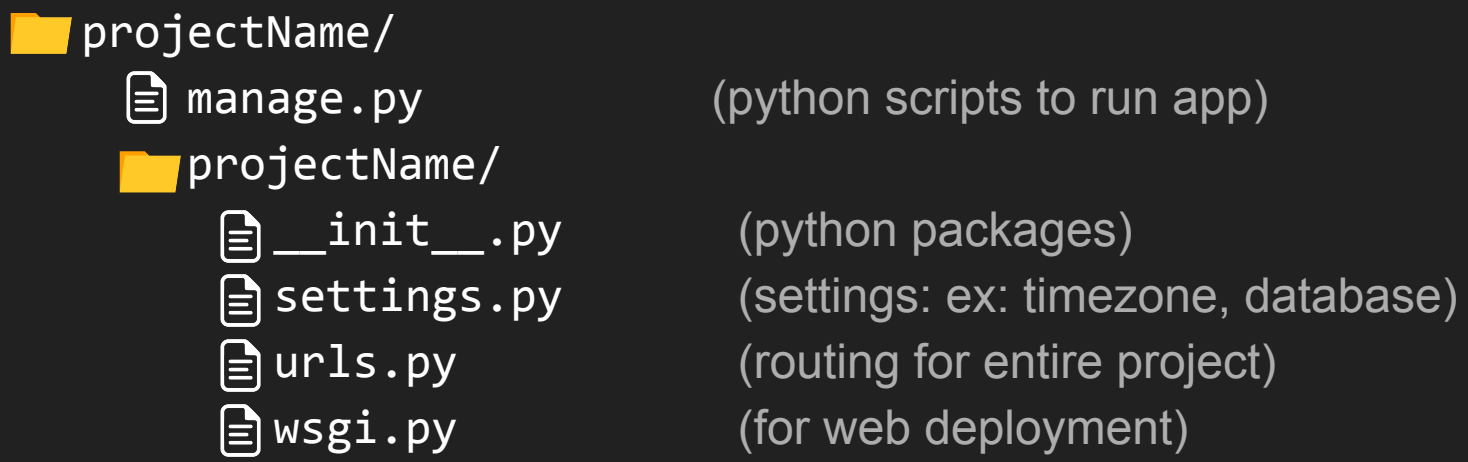
Django

- "Full-stack"
- Tons of built-in tools
- DB access central to framework
- Multiple apps in one project possible
- Django templating language
- Users
  - YouTube
  - Instagram
  - Spotify

Installation: `pip install Django`
Django documentation

Creating a project

```
django-admin startproject <projectname>
```



Slide: Elle Buellesbach

Applications

```
django-admin startapp <appname>
```

```
projectName/
    manage.py
    projectName/        (the high level project controls)
        __init__.py
        settings.py
        urls.py         (URLs will be routed through here first)
        wsgi.py
    appName/            (an individual app within the project)
        __init__.py
        urls.py         (Associating URLs with view functions)
        view.py         (functions that run to create responses)
        models.py
          AND OTHERS!
```

Slide: Elle Buellesbach

Convert a Flask app to Django

Protein identity matrix calculator from UniProt IDs

https://cadd-cdot.appspot.com/identity

Some test data:

```
P00533 P04626 P21860 Q15303
```

Can take a bit to run -- not optimized yet!
Not guaranteed to be up forever. On a public Google Cloud instance for development.

Django databases

Official Django documentation
Mozilla walkthrough of models

- Database models go in `models.py`
  - Similar to `Flask-SQLalchemy` ORM (lecture 4)

```
python manage.py makemigrations
python manage.py migrate
```

```python
class Flight(models.Model):
    origin = models.ForeignKey(Airport, on_delete=models.CASCADE, relatec
    destination = models.ForeignKey(Airport, on_delete=models.CASCADE, re
    duration = models.IntegerField()

    def __str__(self):
        return f"{self.id} - {self.origin} to {self.destination}"

class Passenger(models.Model):
    first = models.CharField(max_length=64)
    last = models.CharField(max_length=64)
    flights = models.ManyToManyField(Flight, blank=True, )
```

Interacting with the DB

- Python shell
  - `python manage.py shell`
  - Interactive Python session
- Django admin app
  - Make a superuser (`python manage.py createsuperuser`)
  - Register models in `admin.py` (`admin.site.register(<model>)`)
  - Run server and go to `http://127.0.0.1:8000/admin`

Django user forms

- Easy to do! A built in feature
  - `from django.contrib.auth import authenticate, login, logout`
- Brian demoed manual user creation
- Lots of resources out there that show how to make a login form
  - Like this one