CSCI E-33a (Web50) Section 4

Ref: Lecture 4 (SQL, Models, and Migrations)

Vlad Popil

Sep 28, 2022

Welcome!

About me:

Volodymyr “Vlad” Popil

- 👨‍🏫 Teaching Fellow for E-33a (since 2019)
- 👨‍👶 Father of 4 y/o son
- 👨‍🎓 Master’s (ALM) in Software Engineering
- 👨‍💻 Software Engineer at Google
- 🇺🇦 Born in Ukraine

Email: vlad@cs50.harvard.edu

Sections: Wed 8:30-10:00 pm ET
Office Hours: Fri 7:00-8:30 pm ET

Agenda

- Logistics
- Lecture review
- Django w/ models Demo
- Project 2
- Grading criteria (not exhaustive)
- `pycodestyle` (repeat)
- `pylint` (stretch)
- Tips (2 tips)
- Q&A

Logistics

Intro

- Refer to website: https://cs50.harvard.edu/extension/web/2022/fall
- Sections and office hours schedule on website sections
- Get comfortable with command line (cd, ls, cat, python …, python -m pip …, rm, touch)
- Text editor is usually sufficient to write code, BUT IDEs (e.g. VSCode) is faster!
- Zoom:
  - Use zoom features like raise hand, chat and other
  - Video presence is STRONGLY encouraged
  - Mute your line when not speaking (enable temporary unmute)
- Six projects:
  - Start early (or even better RIGHT AWAY!!!)
  - Post **and answer** questions on Ed platform
  - Remember: bugs can take time to fix
  - Grade → 3 × Correctness (5/5) + 2 × Design [code] (5/5) + 1 × Style [code] (5/5) (Project 0 is an exception)
    - 15+10+5=30/30 | e.g. Correctness can be 15, 12, 9, 6, 3, 0
  - Overall weights: projects (90%), section attendance (10%)
  - Lateness policy - 72 late hours combined for first 5 proj., then 0.1per minute => **16hrs 40 min**
  - Project 2 - Due Sunday, Oct 9th at 11:59pm EDT **<< ONLY 11 FULL DAYS LEFT >>**

Reminders

- Sections/Office Hours:
  - Sections are recorded (published 72hrs), office hours are not
  - Real-time attendance is required of at least one section
  - Video and participation encouraged even more

- Section prep:
  - Watch lecture(s)
  - Review project requirements

- Office hours prep:
  - *~Write down your questions as you go, TODO, etc.~*
  - Come with particular questions

10,000 foot overview

- *Section 0 - SKIPPED*
- *Section 1+2 (Git + Python)* - Chrome Dev Tools (Inspector), CDT (Network), Project 0, Grading aspects
- *Section 3 (Django)* - Env Config, Markdown, RegEx, IDEs, pycodestyle, Debugging, Project 1
- *Section 4 (SQL, Models, Migrations)* - VSCode, linting, DB modeling, Project 2
- *Section 5 (JavaScript)* - cURL/Postman, jshint, CDT + IDE's Debugging, Project 3
- *Section 6 (User Interfaces)* - Animations, DB modeling, Pagination, Project 4
- *Section 7 (Testing, CI/CD)* - Test Driven Development, DevOps, Final Project
- *Section 8 (Scalability and Security)* - Cryptography, CAs, Attacks, App Deployment (Heroku)

Most sections: material review, logistics, project criteria review, reminders, hints, etc.

Burning Questions?

Please ask questions, or topics to cover today!

Topics:

- *Clarification on when to use return render() vs HttpResponseRedirect ()*
  - *render():*
    - *When: want to stay on same URL and show template (e.g. view entry)*
  - *HttpResponseRedirect():*
    - *When: functionality is already handled by another endpoint, and you want the url to change. (e.g. /random …. Grab "title" want show entry redirect to /wiki/<str:title>*
- *Lecture - models:* code | city; origin (FK -> code)
  - *__str__( )*

Import all models

In [1]: from flights.models import *
Create some new airports
In [2]: **jfk** = Airport(code="JFK", city="New York")
In [4]: **lhr** = Airport(code="LHR", city="London")
In [6]: **cdg** = Airport(code="CDG", city="Paris")
In [9]: **nrt** = Airport(code="NRT", city="Tokyo")

Save the airports to the database
In [3]: jfk.save()
In [5]: lhr.save()
In [8]: cdg.save()
In [10]: nrt.save()
Add a flight and save it to the database

f = Flight(origin=jfk, destination=lhr, duration=414)

f.save()
# Display some info about the flightIn [14]: f

Out[14]: <Flight: 1: New York (JFK) to London (LHR)>

Flight(models.Model):
origin = FK(
destination = FK(
def __str__(self):
return f"{self.id}: {self.origin} to {self.destination}"

Lecture Recap

5-10 min

SQL

Structured Query Language

SQL

- We often store data in **relational databases**, which include a series of tables that are related to one another.
- SQL is a language used to interact with databases (add tables, add rows, modify rows, extract data, etc.)
- Several different database management systems, but we'll use **SQLite**

Tables

- Tables are made up of a series of rows, where each row is a new data point

origin	destination	duration
New York	London	415
Shanghai	Paris	760
Istanbul	Tokyo	700
New York	Paris	435
Moscow	Paris	245
Lima	New York	455

Creating a Table

```
CREATE TABLE flights(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    origin TEXT NOT NULL,
    destination TEXT NOT NULL,
    duration INTEGER NOT NULL
);
```

Adding a row to a Table

```
INSERT INTO flights
    (origin, destination, duration)
    VALUES ("New York", "London", 415);
```

SELECT Queries

- Allows us to extract data from a SQL table
- Many different ways to narrow down which rows and columns we select
- `SELECT * FROM flights;`
- `SELECT origin, destination FROM flights;`
- `SELECT * FROM flights WHERE id = 3;`
- `SELECT * FROM flights WHERE origin = "New York";`
- `SELECT * FROM flights WHERE origin IN ("New York", "Lima");`
- `SELECT * FROM flights WHERE origin LIKE "%a%";`
- `SELECT AVG(duration) FROM flights;`

Updating/Deleting Rows

```
UPDATE flights
    SET duration = 430
    WHERE origin = "New York"
    AND destination = "London";
```

```
DELETE FROM flights WHERE destination = "Tokyo";
```

Joining Tables

- It is often more efficient to have multiple tables to avoid repeating information.

id	code	city
1	JFK	New York
2	PVG	Shanghai
3	IST	Istanbul
4	LHR	London
5	SVO	Moscow
6	LIM	Lima
7	CDG	Paris
8	NRT	Tokyo

id	origin_id	destination_id	duration
1	1	4	415
2	2	7	760
3	3	8	700
4	1	7	435
5	5	7	245
6	6	1	455

SQL Vulnerabilities

- SQL Injection attacks: When a user injects SQL code where your site is expecting plain text.
- Race Conditions: Multiple queries to a database occur simultaneously

Django Models

Django Models

- A layer of abstraction above direct SQL queries and databases
- Django Models are Python **Classes** that extend the `models.Model` class.
- Models can include values and functions.
- Many different field types for values.
- Models contained in `models.py`

A Dog Model

```
class Dog(models.Model):

    name = models.CharField(max_length=50)

    age = models.IntegerField()



    def __str__(self):

        return f"{self.name} is {self.age} years old"
```

Migrations

- We write our models in **models.py**, but that doesn't update our database.
- The command `python manage.py makemigrations` turns our models into Python scripts that can make changes to a database.
- The command `python manage.py migrate` applies our recently made migrations to our current database.

Django Shell

- Similar to the Python Interpreter
- Allows us to run Django commands one at a time
- **python manage.py shell (helpful for writing django queries)**

Model-Related Commands

Command	Purpose
`Object.save()`	Saves a newly created or updated object to your database
`ModelName.objects.all()`	Queries for every instance of that model as a QuerySet
`some_queryset.first()`	Extracts first element from QuerySet
`ModelName.objects.get(query)`	Gets **one** object based on query Alternatively (better) use: **get_object_or_404()**
`ModelName.objects.filter(query)`	Gets **multiple** objects based on query
`object.field.add(other_object)` ~~`object.save()`~~ not needed	Adds another object to a specific field (ManyToMany relationship) ***RUNS SAVE AS WELL***

Relating Models

- `models.ForeignKey`: allows us to store another instance of a model as a field in another model
- `models.ManyToManyField`: allows us to keep track of Many to Many relationships between models.
- **related_name** is an attribute we can give to a field that allows us to query for a specific object based on objects it is related to.

Django Admin

- Allows us to create an administrator that can manipulate models in a nice online interface
- How to use the admin interface:
  - In `admin.py`, register each of your models using `admin.site.register(ModelName)`
  - Create an admin user: `python manage.py createsuperuser`
  - Log into the admin app by visiting `base_url/admin`

Creating Forms from Models

- We can create a Django form class from the models we create!

```
>>> from django.forms import ModelForm
>>> from myapp.models import Article

# Create the form class.
>>> class ArticleForm(ModelForm):
...     class Meta:
...         model = Article
...         fields = ['pub_date', 'headline', 'content', 'reporter']
```

User Authentication

- We can use and extend the Django User model
  - Add `AUTH_USER_MODEL = "dogs.User"` to settings
  - Add `from django.contrib.auth.models import AbstractUser` to the beginning of models.py
  - Extend using `class User(AbstractUser):`
- Automatic User authentication from Django available

Questions?

Django

Demo `cookbook` …

- **[x] DB model**
- **[ ] sqlite3**
- **[x] models.py**
  - **[x] reverse_name - rule of thumb: use class name in plural form**
  - **[x] string representation**
- **[x] Django forms / submission (creating recipe)**
- **[ ] querying/traversing the relationships**

Project

Project 2 (Commerce)

- Start early!!!
- Make a checklist of requirement and check all before submission
- Make sure there’s no bugs
- Focus on functionality (**NOT PRETTINESS**)!!!
- ~~*Think about UI?*~~
- if listing.seller != request.user: <= (views.py) prevent non-owner edit
- .order_by("-creation_time").all() *QuerySet*
- Bid validate if larger than current bid; also first one > starting_price
- get_object_or_404()

Project 2 (Commerce)

- Watchlist add / delete (**separate** or same endpoint)
- Image feature **models.URLField**
- Spend time creating proper model (Let’s take a look…)
- **Functions as parameters! (get_price)**
- Django Forms + extra param (use commit=False)

Design

What can be considered (not exclusively):

- Proper refectoring (copy-paste is usually a no-no)
- Proper use of functions
- More reasonable solution
- Code/file structure
- Additional considerations: error preventions/handling
- Additional considerations for better application
- pylint

Style

What can be considered (not exclusively):

- pycodestyle (indentations, line breaks, long lines) views.py/models.py
- pylint
- **COMMENTS**!
- Naming for variables, function, files, etc.
- Consistency is the key!

Grading criteria generic suggestions (not limited to)

- Correctness:
  - All requirements + bugs
- Design (not limited to) - not UI:
  - Simplest solution
  - Avoiding repetition (refactoring)
  - Structure (e.g separate files vs inline styling)
- Style (not limited to) - not UI:
  - File naming/structure
  - Line breaks
  - Spacing / Indentation
  - Naming
  - Comments

Both Design and Style consider readability but from different perspective.

pycodestyle (formerly pep8)

- python -m pip install pycodestyle
- pycodestyle app.py --max-line-length=120

pylint (checks beyond style, but including)

- python -m pip install pylint
- pip install pylint-django
- pylint app.py --load-plugins pylint_django

HTML beautifiers/prettify

- Automatically formats your HTML (except line breaks)
- Most IDEs supports integration of marketplace beautifiers
- Demo...

IDEs and Debugging

Integrated Development Environments (Intro)

- Text Editor or Heavy IDE?
- Options:
  - VS Code
  - PyCharm (Pro)
  - Atom
  - Sublime
  - vim/Emacs
  - And dozens more, including Notepad :)
- My suggestion: ~~VS Code *or* PyCharm~~ “Codespace VS Code”
- Benefits: Debugging, Autocomplete, Navigation, Find Usages, Linting, Refactoring, Running App and much more.

VS Code

- Demo
- alias code="/Applications/Visual\ Studio\ Code.app/Contents/Resources/app/bin/code"

PyCharm

- No more Demo

Codespace (by GitHub)

- https://github.com/features/codespaces
- *No more demo*

Mac or Windows or Linux?

1. Mac
2. Linux
3. Windows with WSL
4. Windows
5. Remote IDEs e.g. codespaces (formerly on #1)
6. Chromebook? ¯_(ツ)_/¯

Running Python

1. Native installation:
    a. which python
    b. which python3
    c. python --version
    d. python3 --version
    e. python3 -m pip install requests
2. Virtual / Anaconda Environments

Anaconda Distribution

- Anaconda - World's Most Popular Python/R Data Science Platform
- Miniconda (lighter version):
  a. Download https://docs.conda.io/en/latest/miniconda.html
  b. Run in terminal in Downloads: `zsh Miniconda3-latest-MacOSX-x86_64.sh`
  c. Run `conda init` ONLY if not prompted during installation
  d. Create new environment: `
     - `conda create -n s33a python=3.7`
  e. See environments:
     - `conda env list`
  f. Deactivate/Activate environment:
     - `conda deactivate`
     - `conda activate s33a`
  g. Install more packages:
     - `conda install django` (preferred)
     - `pip install django` (if conda doesn't find), although
       It is better to `python -m pip install django` (to assure proper pip)

venv

- Lightweight Virtual Environment (https://docs.python.org/3/library/venv.html)
- Commands
  a. Create new environment: `
     - `python3 -m venv venv_e33a`
  b. Deactivate/Activate environment:
     - `deactivate`
     - `. ./venv_e33a/bin/activate`
  c. Install more packages:
     - `pip install django` BUT:
       It is better to `python -m pip install django` (to assure proper pip)

Random Tips

- Video Speed Controller (Chrome Extension)
  - Spotify + Hulu + Showtime => $5
  - GitHub Education Pack
  - Windows licence (https://harvard.onthehub.com)
  - Chrome Tabs
  - DSA:
    - Video by CS50: https://www.youtube.com/watch?v=OPC1lk-SsCI&ab_channel=CS50
    - Leetcode / AlgoExpert / Etc.
    - Stanford Algorithms Specialization (EdX link /Coursera) - more theory (time consuming)
    - e22 seems good!
    - e20 + e124 (combo) - HARD!
  - System Design:
    - Grokking System Design
    - Alex Xu - System Design

Resources

- https://github.com/vpopil/e33a-sections-fall-2022

CSCI E-33a (Web50) Section 4

Ref: Lecture 4 (SQL, Models, and Migrations)

Vlad Popil

Sep 28, 2022