CSCI E-33a (Web50)
Section 4

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

Vlad Popil

Feb 24, 2021

Welcome!

About me:

Vlad Popil
Master's (ALM) in Software Engineering
Principal DA, Enterprise Data Management, Capital One

Email: vlad@cs50.harvard.edu

Sections: Wed 6:30-8:00pm EST
Office Hours: Thu 8:30-10:30 pm EST

Agenda

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

Logistics

Reminders

- Zoom:
  - Use zoom features like raise hand, chat and other
  - Video presence is STRONGLY encouraged
  - Mute your line when not speaking (enable temporary unmute)
- Projects:
  - Start early (or even better RIGHT AWAY!!!)
  - Post questions on Ed platform
  - Remember: bugs can take time to fix
  - Grade -> 3 × correctness + 2 × design + 1 × style
  - Lateness policy - 0.1 per minute x 60 x 24 x 7 days ~ 16.6 hrs
  - Set a reminder to submit the form for each project
  - Online search, Ed platform, etc.
  - Documentation
  - Project 2 - Due Sunday, Mar 7 at 11:59pm EDT (**11 DAYS LEFT**)

Reminders

- Sections/Office Hours:
  - Sections are recorded, office hours are not
  - Real-time attendance encouraged
  - Video and participation encouraged even more
- Section prep:
  - Watch lecture
  - 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)* - IDE's, SQLite backend, 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:

- *Model type for money - DecimalType(9,2)*
- *<img> not working, use URLField*
- *DB model (we'll cover)*

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

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
`ModelName.objects.filter(query)`	Gets multiple objects based on query
`object.field.add(other_object)`	Adds another object to a specific field (ManyToMany relationship)

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` ...

Project

Project 2 (Commerce)

- Start early!!!
- Google Form
- 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.owner != request.user:
- .order_by("-creation_time").all()
- Bid validate if larger than current bid; also first one > starting
- get_object_or_404()

Project 2 (Commerce)

- Watchlist add / delete (separate or same)
- Image feature models.URLField
- Spend time creating proper model (Let’s take a look…)
- Django Forms + extra param

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 variable, function, files, etc.
- Consistency is the key!

HTML beautifiers/prettify

- Automatically formats your HTML (except line breaks)
- Most IDEs supports integration of marketplace beautifiers
- VS Code doesn’t seem to have solid Django template plugin
- Pycharm does proper Django template formatting

Grading criteria generic suggestions (not limited to)

- Correctness:
  - All requirements + no bugs
- Design (not limited to):
  - Responsive
  - Simplest solution
  - Avoiding repetition (refactoring)
  - Structure (e.g separate files vs inline styling)
- Style (not limited to):
  - File structure
  - Line breaks
  - Spacing
  - Naming
  - **Comments**

Both Design and Style consider readability but from different perspective.

Random Tips

- Windows licence (https://harvard.onthehub.com/)
- GitHub Education Pack
- Spotify + Hulu + Showtime
- ~~The Great Suspender +~~ Chrome Tabs
- Video Speed Controller
- Leetcode / AlgoExpert

WATCHING THIS RECORDED? (**aka “Fruit of the Week”**)

<<< If you are watching this recorded >>

Please email the this name of the fruit in subject (no msg necessary): **CHERRY**

To: **vlad@cs50.harvard.edu**

Thank you.

Q&A

Please ask any questions. Ideas:

- Anything discussed today
- Anything from lecture material
- About the project
- Logistics
- *Random*

Resources

- https://github.com/vpopil/e33a-sections-spring-2021

CSCI E-33a (Web50)
Section 4

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

Vlad Popil

Feb 24, 2021