CSCI E-33a (Web50) Section 7

Ref: Lecture 7 (Testing)

Vlad Popil

Nov 9, 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
- Testing
- Selenium (IDE)
- CI/CD (high-level)
- **Project 4**
- `jshint`, `pylint` (recap)
- Grading criteria (reminders)
- 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 4 - Due Sunday, Nov 13th at 11:59pm EDT **<< ONLY 4 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)* - CDT + IDE's Debugging, Project 3
- *Section 6 (User Interfaces)* - Animations, cURL/Postman, jshint
- *Section 7 (Testing, CI/CD)* - **Project 4**, DB modeling, Pagination, Test Driven Development, DevOps
- *Section 8 (Scalability and Security)* - **Final Project**, Cryptography, CAs, Attacks, App Deployment (Heroku)

Burning Questions?

Please ask questions, or topics to cover today!

Topics:

- Question on Ed, not completely answered, fetch() calls with @csrf_exempt
  - Easy way → add {% csrf_token %} in HTML → querySelect(...) → add to header in fetch()
-

Testing

Testing

- Frequently run and thoroughly written tests can save you a lot of time and effort when it comes to debugging an application.
- Tests become more and more useful the larger your application becomes.
- Test-Driven Development: Every time you find a bug, add a new test to your testing file checking for that bug.
- Along with print statements and such, there are many testing frameworks that we can utilize when writing code.

Testing in Python

Assert Function

- Takes in an argument that should evaluate to True or False
- If the argument is True:
  - Nothing Happens
- If the argument is False:
  - An AssertionError is raised

```
def square(x):
    return x + x

assert square(10) == 100

""" Output:
Traceback (most recent call last):
  File "assert.py", line 4, in <module>
    assert square(10) == 100
AssertionError
"""
```

Unittest Library

- Takes away some of the tedium of testing
- Must `import unittest` into your testing file
- Create a class `Tests` that extends `unittest.TestCase`
- For each test define a method:
  - ```
    def test_some_name(self):
        """Description of test"""

    self.assertSomething(argument)
    ```
- Execute all tests:
  - ```
    if __name__ == "__main__":
        unittest.main()
    ```

```
......
----------------------------------------
Ran 6 tests in 0.000s

OK
```

```
...F.F
========================================
FAIL: test_25 (__main__.Tests)
Check that 25 is not prime.
----------------------------------------
Traceback (most recent call last):
  File "tests1.py", line 26, in test_25
    self.assertFalse(is_prime(25))
AssertionError: True is not false

========================================
FAIL: test_8 (__main__.Tests)
Check that 8 is not prime.
----------------------------------------
Traceback (most recent call last):
  File "tests1.py", line 18, in test_8
    self.assertFalse(is_prime(8))
AssertionError: True is not false

----------------------------------------
Ran 6 tests in 0.001s

FAILED (failures=2)
```

Django TestCase

- A testing framework built specifically for Django applications
- Creates and destroys a separate database for test data
- Getting Started:
  - Always write this code in the `tests.py` file
  - Import your models and `from django.test import Client, TestCase`
  - Create a class that extends `TestCase`
  - Create a `setUp` method within that class where you add data to the new database
  - Add `test_…` methods to the class just like with `unittest`
  - Run the tests by running `python manage.py test`
- Client testing:
  - Create a new client with c_name = Client()
  - Make requests using c_name.get("url_path")

Client-Side Testing

Selenium

- Allows us to set up a virtual user of an HTML page in Python
- Getting Started:
  - Make sure to `pip install selenium` and `pip install chromedriver-py`
  - Import `os`, `pathlib`, and `from selenium import webdriver`
  - Find URI of file as described in notes
  - Set up a driver with `driver = webdriver.Chrome()`
  - Open your HTML file with the driver with `driver.get(uri)`
  - Use different functions to interact with the page
- Consider using Selenium within the unittest library

CI/CD

Continuous Integration and Continuous Delivery

CI/CD

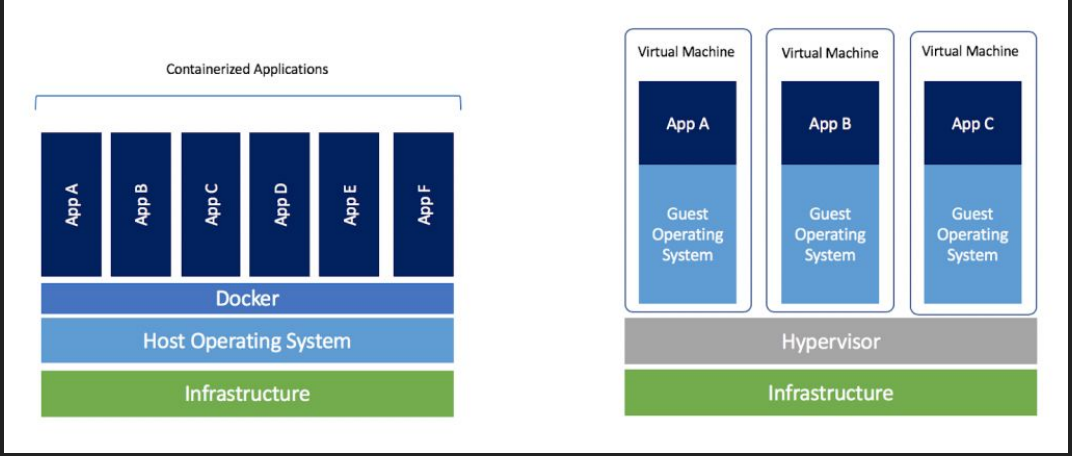
- Continuous Integration
  - Frequent Merges to the ~~main~~ `dev` branch
  - Automated Unit Testing with each merge
- Continuous Delivery
  - Short Release Schedules

GitHub Actions

- Allows us to set up commands to be run every time a certain action is taken
- Can be used to run tests every time code is pushed to a repository
- Can even be used to block pull requests/pushes that fail certain tests.
- We use YAML (YAML Ain’t Markup Language) files to set up actions
- These actions are run on GitHub’s virtual machines
- Helpful Guide

Docker

- One type of **containerization** software
- Different than a **Virtual Machine**
- Set up using a Dockerfile
- [Helpful Guide]() to Docker + Django

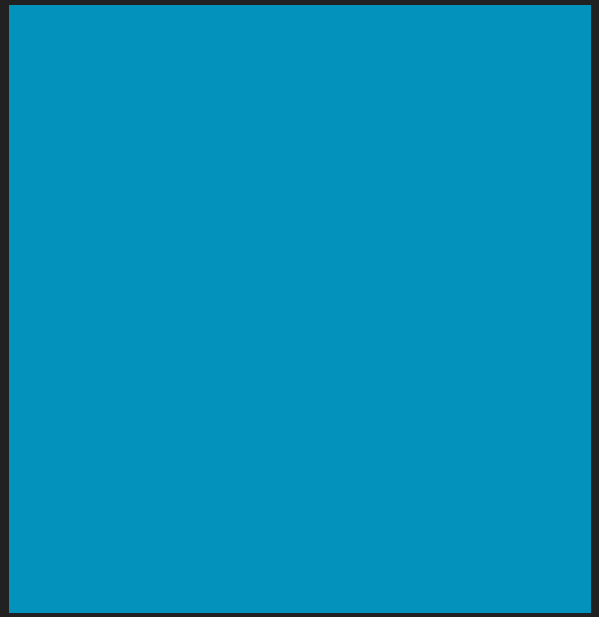


Questions?

Demo

Testing + TDD

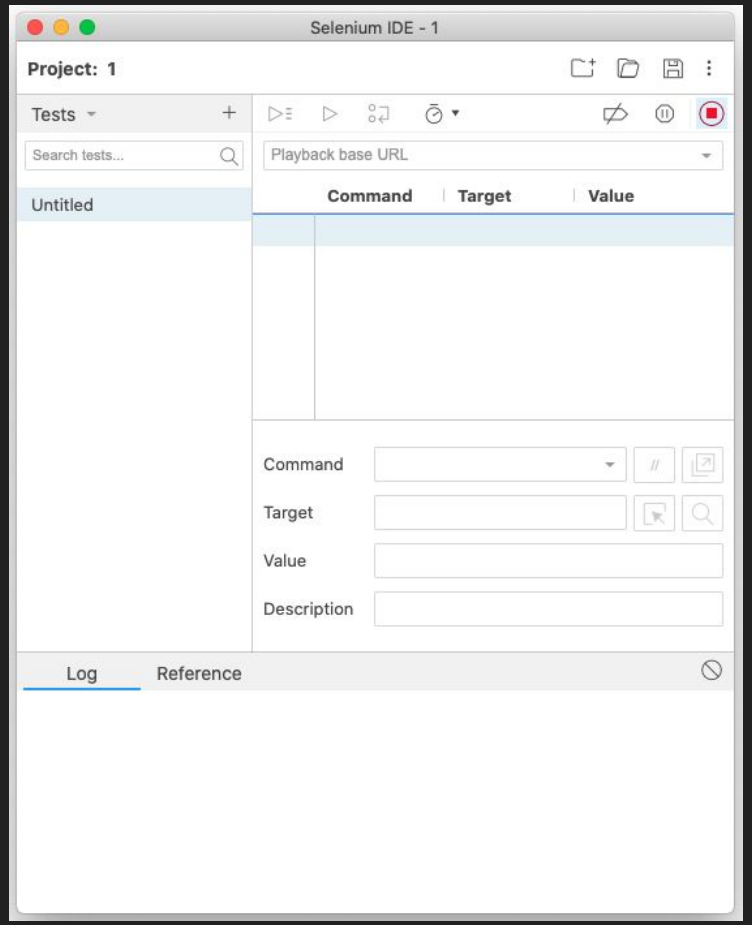
- Fixing bugs has chance of introducing new bugs
- Always write test for production software
- Test-driven development (TDD)



Selenium IDE

Chrome extension for Selenium

Demo...



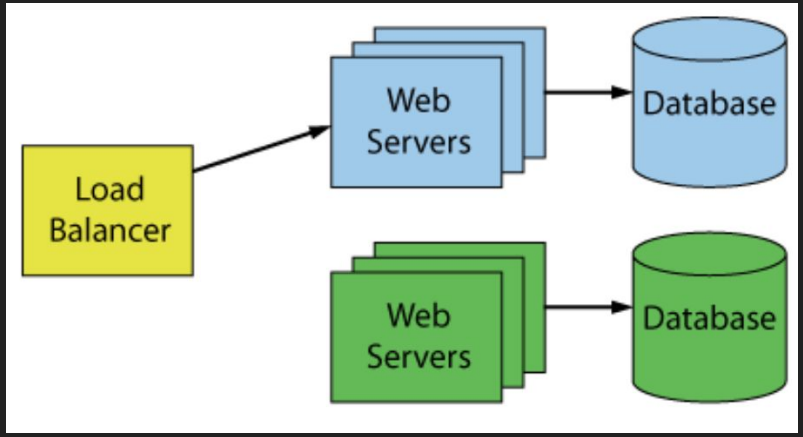
CI/CD (tools)

CI - continuous integration

Examples:

- Github Action
- Travis
- TeamCity
- CircleCI
- Jenkins
- Bamboo
- etc

CD (Blue-Green Deployment)



Questions?

Project 4

Project 4

Common suggestions:

- Start early!!!
- Make a checklist of requirement and check all before submission
- Make sure there’s no bugs
- Focus on functionality (NOT PRETTINESS)!!!

Project 4 (modeling)

- Models:
  - User model can point at self
  - Likes can be a standalone model (2 Foreign keys), or a direct ManyToMany*
  - Modeling exercise?

Project 4 (refactoring)

- All Posts vs Profile vs Following
  - Can be done as three HTML files or can reuse index for all three
  - If reusing index.html:
    - Can show the profile conditionally {% if … %}
    - Can show follow/unfollow buttons conditionally (when? If not your page)
      {% if request.user in profile_user.followers %}
      - Show ‘Unfollow’ button
    - Can show New Post conditionally (when? Not under following, and not on else profile)
      - {% if request.user != profile_user %}
- New Post:
  - Conditionally if reusing index.html, otherwise do we need condition? Probably no (for separate page)

Project 4 (pagination)

- Pagination

Project 4 (javascript vs template)

- JS for:
  - like/unlike
  - edit/save
- Follow/Unfollow
  - Navigation (button is Follow:
    - /follow/<int:user_id>
      - def follow(request, user_id>):
        - followee = User.object.get_object_or_404(pk = user_id)
        - if request.user not in followee.followers:
          - followee.followers.add(request.user)
    - /unfollow/<int:user_id>
      - …
- Posts / dataset (data-...)
- Edit render

Project 4 (csrf_token)

- Token:
  - {{csrf_token}}
  - const token = document.querySelector('input[name="csrfmiddlewaretoken"]').value;
  - fetch(`posts/`, {json},
    headers: {
    "X-CSRFToken": token
    }...)

- Alternative (good enough) No token:
  - @csrf_exempt

Design

What can be considered (not exclusively):

- Proper refactoring (copy-paste is usually a no-no)
- Use of constants/vars:
  - 1. const
  - 2. let
  - 99999. var
- Proper use of functions
- More reasonable solution
- Code/file structure
- fetch(‘/email/’ + id) -> fetch(`/email/${id}`)

Design (continued)

What can be considered (not exclusively):

- Repetitive use of querySelector?
- Proper data structures
- == vs === ?
  - const x = 5
  - const y = ‘5’
  - x == y -> T
  - x === y -> F
- Code repetition

Style

What can be considered (not exclusively):

- *jshint (indentations, line breaks, long lines)*
- COMMENTS!
- Naming for variable, function, files, etc.:
  - getemailbyid -> get_email_by_id (Python convention)
  - getEmailById (JS convention)
- Consistency is the key!

Style (continued)

What can be considered (not exclusively):

- ‘ vs “ consistency
- camelCase(c*, Javascript, Java) vs snake_case (Python)
- == vs ===

jshint

- UI:
  - https://jshint.com/

- CLI:
  - brew update
  - brew doctor
  - brew install node
  - npm install -g jshint
  - In ~/.jshintrc add:
    - {
    - "esversion": 6
    - }

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

Chrome Developer Tools (Network)

In Chrome:

1. Right click
2. Inspect
3. → Demo

Extremely powerful! Let’s try...

cURL / Postman

Allows to call API endpoints directly.

Demo...

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.

Demo

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=ODC1lk-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 7

Ref: Lecture 7 (User Interfaces)

Vlad Popil

Nov 9, 2022