EECS 4313 Project - YuRide Web Testing and Learning Challenge

1. Design Goals

The goal of this project is to practice **white box testing** on a real world web application. You will utilize **Python** and **Selenium** to practice testing on both **frontend** and **backend** on a website and the APIs developed by YuRide.

YuRide is a Ride-Sharing service founded by YorkU Alumni. They allow us to perform testing on their source code and web app in order to help you see a real world example and apply your knowldege that you have gained in this course on a real project.

This project consists of 3 parts, and you will write tests cases using the tools outlined below. You must follow good coding practice and design in order to achieve good grades for this project:

- Good test cases are essential for a robust software, and you must write these test cases to cover all the required aspects.
- Test cases should be well-defined, desireably with no duplication, and cover all the edge cases.
- Documentation principle: Try to write the tests so that it includes all the elements needed to understand the technique(s) that you are using.

1.1 Prerequisite

You will need to get yourself familiar with the following technologies, as well as installing them on your local machine because you are required to use them in order to complete this project.

RESTful APIs

In order to perform API testing, you need to know the definition of APIs. Since we are going to write tests on web service APIs, and these APIs adhere to the REST architectural constraints, so they are called RESTful APIs.
To learn more about the definition of RESTful APIs, you can visit the following links:

- What is REST - REST API Tutorial
- What is a REST API? | IBM - REST API (Introduction) - GeeksforGeeks

The APIs developed by YuRide are RESTful APIs, and we will write unit tests on them.

`requests` **Module**

We are going to use Python module requests for writing tests on APIs. Install the module by though `pip`, the Python package manager:

```
pip install requests
```

You can learn more about the module by reading it's documentation here.

Selenium

Selenium framework in Python is an extremely popular tool that is used by many developers and testers to automate their web tasks.

Using `pip`, you can install selenium like this:

```
pip install selenium
```

Selenium also requires a driver to interface with the chosen browser,

Browser	Driver download
Chrome	https://sites.google.com/chromium.org/driver/
Edge	https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox	https://github.com/mozilla/geckodriver/releases
Safari	https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Download the driver according to your browser, make sure it's in your PATH, e. g., place it in /usr/bin or /usr/local/bin.

For more information, you can refer to the official documentation.

2. Getting Started

Create a Python file, named it `test_yuride.py`, and in the `.py` file, put the following:

```
import requests

response = requests.get('https://dev.yuride.network/')

print(response)
```

Run it, what's the output?

It should be `200`, which is status code `OK`, if you need a refresher on HTTP requests, check out this.

This is how you send a `GET` request to an `URL` using Python module `requests`.

But how do we convert this to a test case? We simply create a `unittest` class:

```
import requests
import unittest

class TestYuride(unittest.TestCase):

    def test_ping(self):
        response = requests.get("https://dev.yuride.network/")
        self.assertEqual(response.status_code, 200)

if __name__ == "__main__":
    unittest.main()
```

If we run this file, it will show us `OK`. This is a simple example on how we write tests for APIs.

3. Requirements

3.1 Learning Challenge 1

For the first learning challenge, we will focus on white-box testing on 3 API endpoints.

Please following good coding standard, as this is also part of the grading. Feel free to create helper functions, write additional assertions or test cases as you wish.

Create a folder named `tests`, and put all the API test files in there.

Assuming the base URL is https://dev.yuride.network/, write unit tests for the following endpoints:

`/api/token/`

The endpoint `/api/token/` returns the current refresh and access tokens for a user, given the user's username and password.

Request:

- Route: `/api/token/`
- Method: **POST**
- Request payload (JSON):

```
{
    "username": "string",
    "password": "string"
}
```

Response:

- Response code: **200**
- Response body (JSON):

```
{
    "refresh": "string",
    "access": "string"
}
```

TODO

==Create test file: `test_token.py`, and write the following unit tests in the test file.==

`test_token`

- Test function name: `test_token`
- Request payload: arbitrary username and password
- Assertion:
    - Status code is 200
    - Verify both refresh and access tokens are returned

`test_token_random`

- Test function name: `test_token_random`
- Request payload: **randomly generated** username and password with 5 letters
- Assertion:
    - Status code is 200
    - Verify both refresh and access tokens are returned

`test_token_no_username`

- Test function name: `test_token_no_username`
- Request payload: arbitrary password, but empty username
- Assertion:

    - Status code is 400

test_token_no_password

- Test function name: test_token_no_password
- Request payload: arbitrary username, but empty password
- Assertion:
    - Status code is 400

test_token_no_body

- Test function name: test_token_no_body
- Request payload: nothing
- Assertion:
    - Status code is 400

Report

==You need to produce a report.pdf file.==

Choose your favorite editor to answer the following questions.

The two tokens are needed to communicate the authentication of a user to the server so that the user would not need to log in every single time they access the webpage.

- Q1: What happens when we send the requests with the same username and password? Hint: you can try to send the request and print the response in a scratch file to see.
- Q2: What if we enter a different password but the same username, what is the response this time?
- Q3: Is this secure behavior? How would you improve the functionality? (Just describe with words is fine)

==Please answer the above questions **in a formal and professional manner**, you should group your answers in their corresponding sections, and include code snippet with outputs if needed.==

/api/token/refresh/

Takes a refresh type JSON web token and returns an access type JSON web token if the refresh token is valid.

Request:

- Route: /api/token/refresh/
- Method: **POST**
- Request payload (JSON):

```
{
      "refresh": "string"
}
```

Response:

- Response code: **200**

- Response body (JSON):

```
{
      "access": "string"
}
```

TODO

==Create test file: test_refresh.py, and write the following unit tests in the test file.==

test_refresh

- Test function name: test_refresh
- Request payload: a valid refresh token
- Assertion:
    - Status code is 200
    - Verify the access token is returned

test_refresh_no_body

- Test function name: test_refresh_no_body
- Request payload: nothing
- Assertion:
    - Status code is 400

test_refresh_invalid_token

- Test function name: test_refresh_invalid_token
- Request payload: an invalid refresh token
- Assertion:
    - Status code is 401
    - The value of field code in response is token_not_valid

Report

- Q1: Why do we need a refresh token, isn't the access token alone enough for the authentication process?
- Q2: Given a valid fresh token and the server returns a new access token, does this work with the other way around? Provide your thoughts and some explanation.

/api/users/me

Returns the information on the given user.

Request:

- Route: /api/users/me/
- Method: **GET**

Response:

- Response code: **200**
- Response body (JSON):

```
{
     "username": "string",
     "email": "string",
     "photo_url": "string",
     "first_name": "string",
     "last_name": "string",
    ...
}
```

TODO

==Create test file: `test_users.py`, and write the following unit tests in the test file.==

This API will return the information on the required user, and you will need to put the access token in your request header.

Generate the access token using the endpoint mentioned in the previous part, with the username and password of your choice.

`test_me`

- Test function name: `test_me`
- Request header: access token
- Assertion:
  - Status code is 200
  - Verify the returned username is the same as the one you created

`test_me_invalid_token`

- Test function name: `test_me_invalid_token`
- Request header: an invalid access token
- Assertion:
  - Status code is 403
  - The value of field `code` in response is `token_not_valid`

`test_me_no_token`

- Test function name: `test_me_no_token`
- Request header: nothing
- Assertion:
  - Status code is 403

Report

- Q1: How is the server identifying which user is sending the requests? Do you think this a secure way?
- Q2: Since we only have an endpoint to retrieve the user information, could you think of any additional endpoints regarding the state of the current user?
    - Write at least 3 endpoints in the same format as above (specify the request, response, route etc.)
    - Justify why do you think these endpoints are needed, and your reasons of choosing the method, request body etc.

This conclude the first learning challenge, you should have improved your understanding on RESTful APIs, white-box testing, HTTP requests, and web authentication.

3.2 Learning Challenge 2

For the second learning challenge, we are going to use Selenium to conduct UI testing.
Before we proceed, make sure you complete the Selenium installation in the previous chapter.
You can refer to the official documentation for how to use Selenium.

TODO

==Start a new script named `test_ui.py`, use the same `unittest` class, test format in the previous challenge, and Import selenium libraries including `webdriver`.==
In the Unit Test class, leverage Selenium to achieve the following functionality:

1. Use Selenium to navigate to **https://dev.yuride.network**
2. Verify the website title is `Your Website
3. Make Selenium click the `Sign In` button on the left
4. Wait until `Passport York Login` page is loaded
5. Use Selenium to enter your own Passport York credentials and login
6. Click the `Send Me a Push` button
7. Wait until you manually confirm the login request on your device
8. Wait until the login redirect OR click the redirect link
9. Verify in the body, there's the `H1` element with content `DASHBOARD` (successfully logged in)

After the correct implementation, the expected behavior is:

1. A browser window opened automatically
2. Navigated to the YuRide website and sign in
3. Passport York credentials are entered and 2 factor authentication is passed through your own intervention
4. Login successful and you are redirected to the Dashboard page

It's your own responsibility to figure out how to install/use new tools, utilize all the necessary information you could find online (documentation, tutorials etc.) to help you learn the new tools.

Report

Walk us though your testing script, design overview, reasons for the method calls, accompany with screenshots of your testing scripts running with your browser opened by Selenium. (if things are happening too fast, you can always make the script wait implicitly)

==!!!BEFORE YOU SUBMIT, remove your Passport York credentials in your test file!!!==

This conclude the second learning challenge, you should have leaned how to do frontend UI testing using Selenium as well as improved HTML knowledge.

3.3 Learning Challenge 3

In this last challenge, we are focusing on investigating some privacy topics.

1. Navigate to YuRide dev website: https://dev.yuride.network/
2. Log yourself in using your own Passport York credentials
3. After logging in, go to https://dev.yuride.network/api/users/me/

Report

Answer the following questions in a small essay fashion, in 100-200 words:

1. What information do you find about yourself on the YuRide platform?
2. Can you see your Passport York username? What has it became and Why?
3. What would YuRide need to obtain to collect your information and store it?
4. What do you suggest YuRide should do to securely store your information?

This conclude the third and final learning challenge, you should feel safe on logging into YuRide with your Passport York now.

4. To Submit

You should submit a `zip` file with the following structure:

```
yuride.zip
|
|
├───tests/
|    |
|    ├───test_refresh.py
|    |
|    ├───test_token.py
|    |
|    ├───test_ui.py
|    |
|    └───test_users.py
|
|
└───report.pdf
```

==!!!BEFORE YOU SUBMIT, remove your Passport York credentials in the `test_ui` file!!!==