PROIECT FINAL

- SUCIU MIHAI -

- 26.01.2023 -

Partea I

- **Variabila** – container in memorie in care se stocheaza valori, in functie de tipul de date utilizat. Variabilele isi pot schimba valoarea in timpul executiei programului

  Constantele – asemanatoare variabilelor, insa nu isi schimba valoarea pe parcursul executiei programului
- Un **tip de data** descrie caracteristica unei variabile

Tipuri de date: pastreaza o singura valoare intr-o variabila

- Numeric: tip integer x = 1, float y=2.5, double, long, complex
- Sir de caractere: tip string “examen”
- Boolean: True, False, 1, 0

Structuri de date: pastreaza mai multe valori intr-o singura variabila

- Lista: in functie de limbajul de programare, pot fi sau nu pastrate mai multe tipuri de date, valorile au index, accepta valori duplicate, sunt ordonate, mutabile

  my_list = [1, 5, -1, 56, “examen”, True]
- Dictionar: date de tip {cheie : valoare}, nu au index, nu accepta chei duplicate, sunt ordonate, mutabile

  my_dict = {“culoare”: “rosu”, “material”: ”panza”, “numar”: 5}
- Set: pastreaza mai multe valori unice, nu au index, nu sunt ordonate, imutabile

  my_set = {'cerc', 'patrat', 'dreptunghi'}
- Tuplu: pastreaza mai multe valori imutabile intr-o singura variabila, sunt ordonate au index, accepta valori duplicate

  my_tuple = (“masina”, “strada”, “semafor”)

Partea I

- **If...else**: - structura de tip conditional avand sintaxa

  daca <conditie> atunci:

  executa ramura 1cod

  altfel:

  executa ramura 2 cod

Structura decizionala care ramifica codul. Daca conditia este adevarata se va executa ramura de cod din blocul 'if', altfel se va executa ramura de cod din blocul 'else'. O structura decizionala poate fi mult mai complexa cu ajutorul 'else if', putand astfel adauga mai multe ramuri.

- **Functia**: - bloc de cod cu o logica proprie, care este executat doar cand este apelat. Ajuta la evitarea repetarii liniilor de cod in program. Functia poate fi apelata oride cate ori este nevoie. Se pot folosi, optional, date de intrare in functie, numite ***parametri***. Parametri sunt specificati intre paranteze, dupa numele functiei. O functie poate avea oricati parametri. Acestia sunt practic niste variabile declarate dar neinitializate. Initializarea se face la apelarea functiei.
- **Clasa** este un constructor pentru un obiect, adica un “blueprint”/o reteta, un concept. Existenta clasei nu inseamna automat si existenta obiectelor definite de clasa. **Obiectele** sunt instante ale clasei. Se pot instantia oricate obiecte din aceeasi clasa, acestea avand aceleasi atribute si aceleasi metode. Clasa contine elemente descriptive pentru obiect, numite atribute sau fields. Atributele pot fi modificate dupa instantiere. Clasa contine, de asemenea, actiuni posibile care se pot face asupra obiectului, numite metode. Metodele sunt practic functii. “Self”(in Python) sau “this”(in Java) este instanta a clasei care ajuta functia sa aiba acces la atributele clasei.

Partea I

- **Selectori**: - sunt folositi pentru a cauta(selecta) elemente HTML.

Tipuri de selectori:

- Id
- Link Text
- Partial Link Text
- Name
- Tag
- Class Name
- CSS
- Xpath

Partea I

TDD(Test Driven Development): - este o tehnica de dezvoltare Agile utilizata in ingineria software. Este un proces de dezvoltare inversa, in care prima data se scriu testele, iar apoi se scrie noul cod si se extinde continuu, astfel incat acesta sa treaca testele scrise. Procesul se face unitar pe functionalitati, iar ciclul se repeta continuu pana cand toate unitatile proiectate trec testele stabilite in functie de cerintele aplicatiei. Se foloseste **testarea unitara** , adica testarea individuala a unor unitati separate din programul dezvoltat. Unitatile pot fi o clasa, o metoda, o functie, un modul din programul dezvoltat.

BDD(Behaviour Driven Develoment): - dezvoltarea condusa de comportament este o extensie a TDD, primul pas este de a scrie cazuri de comportament, ulterior fiind scris codul necesar pentru dezvoltarea programului, astfel incat acesta sa indeplineasca comportamentul asteptat. Focusul este de asemenea pe testare si pe calitatea produsului. Avantajul este faptul ca toate persoanele interesate pot intelege usor rapoartele generate.

Cu ajutorul sintaxei Gherkin, product owner-ul poate genera, relativ usor, “user stories” pentru cazurile de comportament asteptate, care contin de asemenea “acceptance criteria”. Sintaxa Gherkin este o metoda usoara de a scrie test cases-urile. Foloseste limba engleza la un nivel simplu, fiecare linie incepe cu un cuvant cheie: Given, When, Then, And, But, cu ajutorul carora sunt descrise cazurile de comportament ale utilizatorilor in aplicatia dezvoltata.

Partea I

- **API (Application Programming Interface)**: - este o interfata software care asigura legatura intre doua aplicatii fara interventia utilizatorului, permitant comunicarea si schimbul de date. De exemplu, interfata intre client, care poate fi o pagina web, si logica din spate, care poate fi un server, o baza de date, etc.

- **HTTP(Hypertext Transfer Protocol)**: - este o metoda folosita pentru accesarea informatiilor pastrate pe serverele WWW(World Wide Web). Protocol de tip text de nivel aplicatie. Ofera o tehnica de comunicare prin care paginile web se pot transmite de la server, aflat la distanta, catre client.

- **Principalele metode HTTP**: - exista 5 metode principale pe care un client le poate folosi intr-o cerere pentru a manipula o resursa API
  - GET – metoda folosita pentru a extrage date de pe un server
  - POST – metoda utilizata pentru a creea o noua resursa pe server
  - PUT – metoda utilizata pentru actualizarea datelor pe server
  - PATCH – metoda utilizata pentru actualizarea partiala a datelor pe server
  - DELETE – metoda utilizata pentru stergerea unor resurse de pe server

  Pentru orice request de tip HTTP, API-ul returneaza un HTTP response status code, care indica daca request-ul a fost valid sau a aparut o eroare:

  (100 - 199) – raspunsuri de informare
  (200 - 299) – raspunsuri ‘successful’
  (300 – 399) – mesaje de redirectare
  (400 - 499) – raspunsuri de tip ‘client error’, request invalid
  (500 - 599) – raspunsuri de tip ‘server error’

Partea II

In prezentul proiect, am realizat testarea automata a unui API, specific unei pagini web de e-commerce. API-ul testat se numeste Simple Grocery Store API, are URL-ul de baza https://simple-grocery-store-api.glitch.me, iar documentatia completa a acestuia poate fi gasita pe Github https://github.com/vdespa/Postman-Complete-Guide-API-Testing/blob/main/simple-grocery-store-api.md.

Proiectul de testare automata poate fi accesat pe Github: https://github.com/MihaiS-git/SimpleGroceryStoreAPI

Proiectul continue 44 de teste, din care 4 teste cad din cauza unor bug-uri descoperite in API. Bug-urile constau in dublarea id-ului unor produse si in returnarea de produse diferite in cazul request-ului pe categorii de produse fata de request-ul de returnare a intregii liste de produse.

Simple Grocery Store API are un numar de 5 endpoint-uri, ‘Status’, ‘Products’, ‘Cart’, ‘Orders’ si ‘API Authentication’ care au fost testate fiecare in parte. Pentru scrierea testelor, am folosit limbajul de programare ‘Python 3.10.5’ si IDE-ul ‘PyCharm’. Am utilizat, de asemenea, framework-ul ‘Pytest’ care faciliteaza scrierea de teste automate si rularea acestora atat in pachet, cat si in mod independent, libraria ‘pytest-html’ pentru generarea raportului in format html, libraria ‘requests’ pentru trimiterea request-urilor catre API, ‘json’ si ‘jsonschema‘ pentru raspunsurile de tip JSON primite de la API, respectiv librarira ‘assertpy’ pentru validarea testelor.

Partea II

Testarea automata realizata este o testare de tip black-box, functionalitatea si securitatea API-ul au fost testate fara a avea access nici la cod, nici la structura interna a acestuia. Testarea de tip black-box a fost realizata cu focusare pe inputs si outputs:

- Examinarea documentatiei si testarea manuala a functionalitatii API-ului cu Postman, stabilind input-uri valide pentru requests si obtinand raspunsuri de tip JSON, respectiv HTTP code response-urile aferente

- Positiv test scenario & negativ test scenario – scrierea in Python de teste cu input-uri valide si scrierea de teste cu input-uri invalide pentru verificarea functionarii corecte

- Validarea raspunsurilor primite in functie de input-uri si raspunsurile asteptate, respectiv ‘schema’ JSON a raspunsurilor

- Construirea de scenarii de testare pe baza input-urilor testate anterior

- Executarea test case-urilor

- Compararea raspunsurilor primite cu raspunsurile asteptate

- Verificarea securitatii prin testarea validitatii structurii raspunsurilor

- Verificarea securitatii testand folosirea de input-uri de autentificare invalide

Partea II

Proiectul se gaseste integral pe Github si poate fi clonat si rulat astfel:

Prerequisites:

- instalare Python https://www.python.org/downloads/
- instalare IDE https://www.jetbrains.com/pycharm/download
- instalare Git https://gitforwindows.org

Rulare teste:

- se creeaza un folder local pentru clonarea proiectului
- se deschide Git Bash din folderul creat
- se utilizeaza in Git Bash comanda: git clone https://github.com/MihaiS-git/SimpleGroceryStoreAPI
- se deschide IDE-ul instalat, iar din terminal se acceseaza adresa folderului in care a fost clonat proiectul
- tot din terminalul IDE-ului, se vor instala toate pachetele necesare specificate in fisierul 'requirements.txt', folosind comanda: pip install -r requirements.txt
- din terminalul IDE-ului, din folderul 'tests', pentru rularea testelor si generarea raportului html, se utlizeaza comanda: pytest --html=report.html --self-contained-html
- raportul html generat poate fi gasit in folderul 'tests/report.html'

Partea II

Proiectul a fost structurat in mai multe foldere, fiecare avand un rol distinct in asamblul si functionalitatea proiectului, astfel:

- api_requests – contine fisierul ‘api_requests.py’ unde sunt toate functiile pentru timiterea de request-uri catre API, folosind principalele metode HTTP: GET, POST, PUT, PATCH, DELETE

- api_utils – contine fisierul ‘api_utils.py’ unde este stocat URL-ul de baza pentru apelarea API-ului, parametri de apelare, functii de utilitate generala in cadrul proiectului, care genereaza, citesc si returneaza date de autentificare ale clientului, stocate in fisiere txt din acelasi folder

- json_schema – contine JSON ‘schema’ de validare pentru raspunsurile returnate de catre API

- tests – contine fisierele cu scenariile de test si raportul html generat in urma rularii testelor

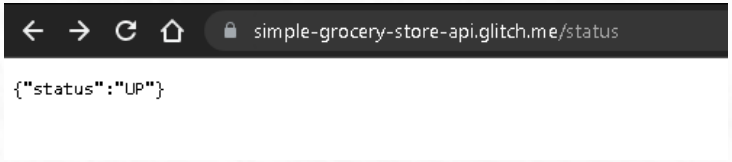
- validators – contine fisierul ‘json_validator.py’ cu functii de validare a JSON ‘schema’, utilizate in teste

Legaturile dintre toate aceste fisiere au fost realizate prin importuri intr-o stuctura procedurala logica si functionala, care evita repetarea codului.

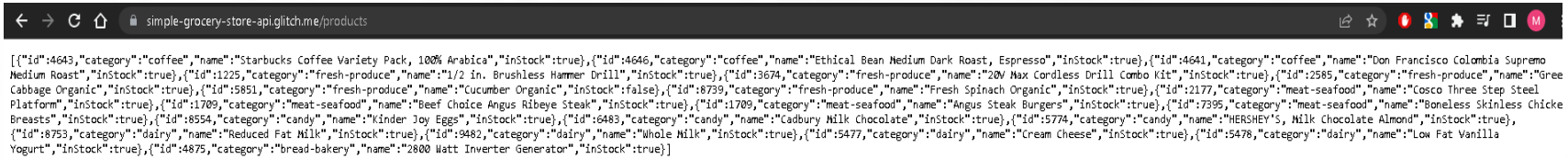
Partea II

Modul de accesare a endpoint-urilor API-ului – Exemple:

- Accesarea endpoint-ului ‘Status’ se poate face direct din browser cu URL-ul de baza la care se adauga numele endpointului:



- Accesarea endpoint-ului ‘Products’:



In imagini se vad si raspunsurile pe care le trimite API-ul la request-urile de accesare a endpoint-urilor. Acestea sunt raspunsuri in format JSON(JavaScript Object Notation). JSON este un format realtiv simplu de gazduire si transmitere de date constituit din perechi ‘cheie’: ‘valoare’, usor lizibil. Tipurile de date de baza ale JSON sunt: numeric, string, Boolean, array, object si null.

Pentru fiecare raspuns API in format JSON, se asteapta o anumita ‘schema’ pentru formatul datelor returnate, care trebuie de asemenea validata, procesul facand parte din testarea de securitate. Structurile ‘schema’ validate in prezentul proiect, pot fi accesate in folder-ul ‘json_schema’.

Partea II

Exemplu de JSON ‘schema’ pentru raspunsul asteptat de la endpoint-ul ‘Status’:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "status": {
            "type": "string"
        }
    },
    "additionalProperties": false
}
```

Functia si metoda folosite la validarea JSON ‘schema’ pentru ‘Status’:

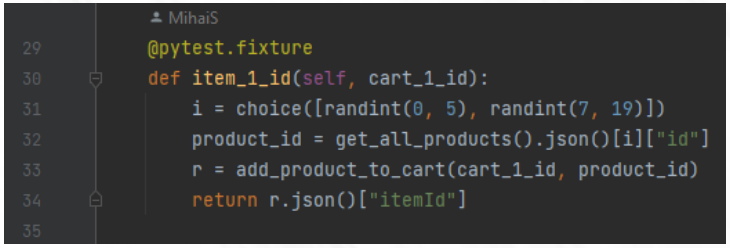
```
MihaiS
def validate_json_schema(json_data, json_schema):
    try:
        validate(instance=json_data, schema=json_schema)
    except jsonschema.exceptions.ValidationError as err:
        return False
    return True


MihaiS
def validate_status_json():
    status_schema = json.loads(open('../json_schema/status_schema.txt').read())
    response = get_status()
    json_data = response.json()
    is_valid = validate_json_schema(json_data, status_schema)
    return is_valid
```

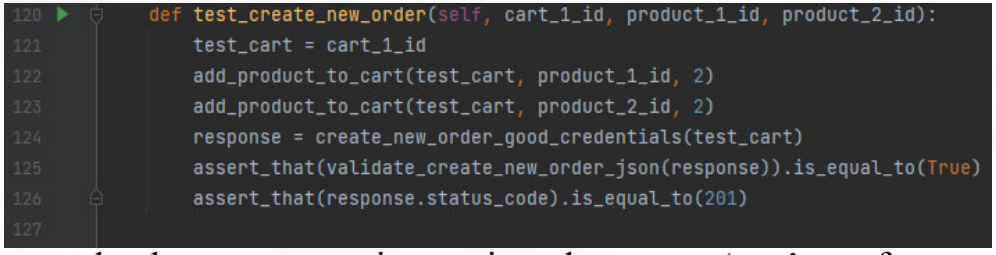
In fisierul ‘json_validator.py’ din folder-ul ‘validators’, am creat functia ‘validate_json_schema’ care realizeaza efectiv validarea, iar functia este apelata in toate metodele de validare, evitand astfel rescrierea codului. Parametrii functiei sunt raspunsul API-ului in format JSON si ‘schema’ ca format asteptat pentru structura raspunsului primit, citita dintr-un fisier text. Functia returneaza True sau False in urma validarii.

Partea II

Testarea functionalitatii API-ului:

test_api_functionality.py – contine toate teste de functionalitate a API-ului, scrise pe baza documentatiei acestuia. Am creat o clasa de teste denumita 'TestAPIFunctionality'. In primele linii, se regasesc cu decoratorul '@pytest.fixture', functii ce stabilesc baseline-ul pentru anumite teste din fisier. Practic, sunt folosite ca si parametrii de intrare, stabilind un anumit mediu de lucru(environment) pentru teste sau alte fixtures. De exemplu, creaza un cos de cumparaturi 'cart_1_id' si returneaza id-ul acestuia, folosit ulterior ca si parametru atat intr-un alt fixture, cat si in testul 'test_create_new_order'..





Urmeaza efectiv testele in sine, sub forma de metode ale caror nume incepe intotdeauna cu 'test', conform instructiunilor PyTest. Astfel, aceste metode sunt recunoscute in mod automat ca si teste ce pot fi rulate si in mod independent.

Pentru a asigura validitatea generala a testelor, la alegerea produselor returnate de API, am folosit modulul in-built Python denumit 'random', care are utilitatea de a genera numere aleatorii dintr-un interval specificat sau de a face alegeri aleatorii dintr-o lista.

Partea II

Raportul HTML pentru ‘test_api_functionality.py’ ne arata ca testele au trecut cu o exceptie:

Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_status	1.19
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_head_api	0.55
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_all_products	1.18
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_coffee_category_products	1.12
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_fresh_produce_category_products	1.14
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_meat_seafood_category_products	1.08
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_candy_category_products	1.23
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_dairy_category_products	1.09
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_a_product_by_id	1.22
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_create_a_cart	1.23
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_add_item_to_cart	1.79
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_cart_items	2.32
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_replace_item_in_cart	3.27
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_delete_item_in_cart	2.58
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_register_api_client_new_credentials	0.62
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_create_new_order	4.37
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_all_orders	5.42
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_update_order	3.08
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_get_a_single_order	3.12
Passed *(show details)*	test_api_functionality.py::TestAPIFunctionality::test_delete_an_order	3.03

Testul care solicita returnarea produselor de tip “bread-bakery, care a picat din cauza faptul ca solicitand lista tuturor produselor din API, am obtinut 1 produs din acest tip si ne asteptam ca raspunsul sa fie acelasi si la request-ul pe categorie, insa sunt returnate 5 produse diferite:

Result	Test
Failed *(hide details)*	test_api_functionality.py::TestAPIFunctionality::test_get_bread_bakery_category_products

```
self = <tests.test_api_functionality.TestAPIFunctionality object at 0x000001DDD2051DB0>

    def test_get_bread_bakery_category_products(self):
        response = get_products_list_by_category('bread-bakery')
        assert_that(validate_bread_bakery_products_json() is True)
        assert_that(response.status_code).is_equal_to(200)
>       assert_that(len(response.json())).is_equal_to(1)
E       AssertionError: Expected <5> to be equal to <1>, but was not.

test_api_functionality.py:86: AssertionError
```

Se poate observa in cod validarea in cascada a JSON ‘schema’, a status codului HTTP si a numarului de produse returnate in raspunsul API-ului. Pentru orice validare falsa, testul cade, indicand eroarea de ‘assert’

Partea II

Testarea securitatii API-ului:

test_api_security.py – contine teste de autentificare de tip negativ. Se testeaza inregistrarea unui nou client avand un nume deja utilizat, ceea ce este posibil, status code-ul asteptat fiind ‘201’ created, inregistrarea unui nou client cu un email deja utilizat, ceea ce nu este acceptat, status code-ul asteptat fiind ‘409’ request invalid. De asemenea se testeaza crearea unei noi comenzi(‘order’) cu un token de access fals, obtinerea tuturor sau a unei singure comenzi, repectiv stergerea unei comenzi cu un token fals, status code-ul asteptat fiind ‘401’ request invalid. Toate aceste teste au trecut cu success.

Passed *(show details)*	test_api_security.py::TestAPISecurity::test_register_api_client_new_credentials	0.00
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_register_api_client_registered_name	0.64
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_register_api_client_registered_email	0.56
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_create_new_order_fake_token	2.28
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_get_all_orders_fake_token	2.87
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_update_order_fake_token	2.94
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_get_a_single_order_fake_token	2.84
Passed *(show details)*	test_api_security.py::TestAPISecurity::test_delete_an_order_fake_token	3.03

Alte scenarii:

test_scenario_1.py – testarea accesului clientilor la toate produsele fie accesand ‘all products’, fie ‘by category’(cazut – numarul produselor returnate ca lista complete de produse este diferit de suma numarului celor returnate pe categorie)

test_scenario_2.py –testarea unicitatii id-ului produselor returnate (cazut – exista id folosit pentru 2 produse diferite)

test_scenario_3.py – testarea adaugarii in cos a tuturor produselor cu id unic care sunt pe stoc, adaugarea de produse care nu sunt pe stoc , adaugarea tutror produselor din stoc indifferent de id(cade – exista id dublat, iar produsul cu id dublat nu mai poate fi adaugat in cos)

test_scenario_4_e2e.py – este un test de tip ‘end to end’, se testeaza parcurgerea inlantuita a mai multor pasi: crearea uni cos, adaugarea de produse in cos, consultarea produselor aflate in cos, modificarea acestora, inlocuirea si stergerea de produse din cos, crearea unui cont, crearea unei comenzi, consultarea comenzilor, updatarea comenzii, stergerea comenzii

PROIECT FINAL

Va multumesc!

- SUCIU MIHAI -
- 26.01.2023 -