**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Технологии автоматизации процесса разработки программного обеспечения»**

Тема: Разработка системы автоматизированного тестирования веб-приложений

Вариант 4

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Королёв С.Ю. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Королёв С.Ю. | | |
| Группа 9303 | | |
| Тема работы: Разработка системы автоматизированного тестирования веб-приложений | | |
| Исходные данные:  Необходимо реализовать docker-compose конфигурацию из двух узлов:   * app – контейнер с существующим демонстрационным веб-приложением * tester – контейнер для запуска всех тестов | | |
| Содержание пояснительной записки:  Содержание; Введение; Постановка задачи; Описание Dockerfile; Описание скриптов запуска тестов; Описание docker-compose конфигурации; Заключение; Список использованных источников. | | |
| Предполагаемый объем пояснительной записки:  Не менее 16 страниц. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Королёв С.Ю. |
| Преподаватель |  | Заславский М.М. |

**Аннотация**

Данная курсовая работа описывает конфигурацию системы, используемой для автоматизированного тестирования веб-приложений – демонстрационного веб-приложения и тестового экземпляра ИС ИОТ. Система состоит из двух docker-контейнеров, в первом запускается демонстрационное веб-приложение, а во втором выполняется запуск нескольких этапов тестирования, включая форматирование кода, статический анализ, интеграционное тестирование и тестирование с использованием веб-драйвера Selenium.

**Summary**

This course work describes the configuration of the system used for automated testing of web applications - a demo web application and a test instance of the IOT IS. The system consists of two Docker containers, the first of which runs a demo web application, and the second of which runs several testing stages, including code formatting, static analysis, integration testing and testing using the Selenium web driver

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 5](#_Toc165657596)

[**ПОСТАНОВКА ЗАДАЧИ** 6](#_Toc165657597)

[**1. ОПИСАНИЕ DOCKERFILE** 8](#_Toc165657598)

[1.1. Dockerfile для app-контейнера 8](#_Toc165657599)

[1.2. Dockerfile для tester-контейнера 8](#_Toc165657600)

[**2. ОПИСАНИЕ СКРИПТОВ ЗАПУСКА ТЕСТОВ** 11](#_Toc165657601)

[2.1. Скрипт run\_all.sh для запуска этапов тестирования 11](#_Toc165657602)

[2.2. Этап форматирования 11](#_Toc165657603)

[2.3. Этап статического анализа 11](#_Toc165657604)

[2.4. Этап интеграционного тестирования 12](#_Toc165657605)

[2.5. Этап selenium тестирования 12](#_Toc165657606)

[**3. ОПИСАНИЕ DOCKER-COMPOSE КОНФИГУРАЦИИ** 13](#_Toc165657607)

[**ЗАКЛЮЧЕНИЕ** 14](#_Toc165657608)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 15](#_Toc165657609)

[**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОЕКТА** 16](#_Toc165657610)

# **ВВЕДЕНИЕ**

Цель данной работы заключается в реализации docker-compose конфигурации, предназначенной для сборки и запуска контейнеров app и tester. Контейнеры по отдельности выполняют задачи, включающие в себя запуск демонстрационного веб-приложения, а также тестирование данного веб-приложения и ИС ИОТ на нескольких этапах (форматирование, статический анализ, интеграционные тесты, а также тесты с использованием веб-драйвера Selenium).

# **ПОСТАНОВКА ЗАДАЧИ**

Необходимо реализовать docker-compose конфигурацию из двух узлов:

* app - контейнер с существующим демонстрационным веб-приложением.
  + Устанавливать приложение необходимо скачивая репозиторий и копируя файлы из него при сборке вашего контейнера.
  + Чтобы все заработало, вам придется потратить время и поразбираться - из коробки может не работать.
  + Возможно, вам для выполнения заданий потребуются фиксы в исходник - делайте для них патчи
  + Корнем дерева процессов выступает запущенное веб-приложение
* tester - контейнер для запуска всех тестов (состав и особенности тестов задаются в таблице вариантов)
  + Корнем дерева процессов выступает стандартный python http сервер (python -m http.server 3000)
  + Этот сервер должен быть запущен в каталоге контейнера, где будет происходить работа тестовых скриптов
  + Тестовые скрипты запускаются через docker exec

При этом при разработке необходимо учесть следующие требования:

* Dockerfile:
  + Минимальная версия докера Docker version 19.03.13, build 4484c46d9d
  + Базовый образ ubuntu:22.04
  + Не использовать Expose
  + При установке любых пакетов и программ (в том числе в requirements) ВСЕГДА указывать версии
  + Ограничить установку зависимостей apt одной строкой (один RUN)
  + Если настройка одной части приложения состоит из нескольких команд → необходимо разместить их в одном слое (в одном RUN)
* Docker-compose:
  + Минимальная версия docker compose version 1.27.4, build 40524192
  + Должно собираться по команде docker-compose build без sudo
  + Не использовать тип сети HOST
  + Не отрывать лишних (непредусмотренных заданием) портов
  + Не использовать порты хост-машины ⇐1024

Параметры конфигурации, заданные для 2 варианта:

|  |  |
| --- | --- |
| Параметр | Требование |
| Проверка на соответствие стилю кодирования / бьютификация | Форматирование Python (flake8) |
| Статический анализ | Анализ по 10 существующим критериям |
| Интеграционные тесты | Проверка на коды возврата |
| Selenium-тесты | Создание ОПОП, проверка истории изменений и выдачи прав. |
| Внешний SSH доступ в контейнеры | В tester – по публичному ключу (существующему) |
| Вывод логов работы tester | Каждый этап тестирования - в docker log (stdout + stderr) и в отдельный файл оба потока по каждому виду тестирования |
| Передача параметров в конфигурацию через .env | Список этапов тестирования для запуска |
| Органичения ресурсов | настройки | ОЗУ |

# **1. ОПИСАНИЕ DOCKERFILE**

## **Dockerfile для app-контейнера**

Последовательность инструкций создания образа app-контейнера:

1. Базовый образ - ubuntu:22.04.
2. Обновляются пакетные списки и устанавливаются необходимые apt-зависимости:
   1. git – система управления версиями для дальнейшего клонирования репозитория с демонстрационным приложением
   2. python3 – интерпретатор Python, необходимый для запуска веб-приложения
   3. python3-pip – пакетный менеджер Python, используемый для установки зависимостей
3. Клонируется репозиторий с демонстрационным веб-приложением и устанавливается рабочая директория внутри этого репозитория
4. Устанавливаются необходимые зависимости Python, используемые в веб-приложении (из файла requirements.txt):
   1. celery – асинхронная очередь задач
   2. Flask – фреймворк для создания веб-приложений
   3. Flask-Login – расширение Flask для аутентификации
   4. lti – библиотека для реализации LTI веб-приложений
5. Копируется реализованный patch-файл, изменяющий main.py для корректной работы веб-приложения.
6. Задается точка входа для контейнера, запускающая веб-приложение.

## **Dockerfile для tester-контейнера**

Последовательность инструкций создания образа tester-контейнера:

1. Базовый образ - ubuntu:22.04.
2. Обновляются пакетные списки и устанавливаются необходимые apt-зависимости:
   1. openssh-server – предоставляет SSH сервер для дальнейшего доступа по SSH
   2. git – система управления версиями для дальнейшего клонирования репозитория с демонстрационным приложением
   3. python3 – интерпретатор Python, необходимый для запуска веб-приложения
   4. python3-pip – пакетный менеджер Python, используемый для установки зависимостей
   5. wget – утилита для загрузки файлов, необходимая для загрузки Google Chrome для использования в Selenium-тестах
   6. xvfb – виртуальный фреймбуфер X, используемый для запуска Google Chrome виртуально в рамках Selenium-тестирования
3. Скачивается и устанавливается браузер Google Chrome для дальнейшего выполнения Selenium-тестов с его использованием.
4. Производится настройка конфигурации SSH сервера для разрешения входа под пользователем root, копируется публичный SSH ключ.
5. Клонируется демонстрационный проект из репозитория.
6. Копируются файлы из директории /tests внутрь контейнера
7. Устанавливаются необходимые зависимости Python, используемые в тестах (из файла requirements.txt):
   1. flake8 – инструмент для автоматического форматирования Python кода.
   2. pylint – инструмент статического анализа Python кода, проверяющий соответствие стандартам
   3. pytest – фреймворк написания и выполнения Python-тестов
   4. pytest-flake8 – плагин pytest для проверки требований flake8
   5. pytest-pylint – плагин pylint для подавления ложных срабатываний, связанных с pytest
   6. requests – библиотека для отправки HTTP-запросов, используется в интеграционных тестах
   7. selenium – библиотека для автоматизации тестирования веб-браузера
   8. webdriver-manager – инструмент для установки веб-драйверов (в частности для браузера Google Chrome)
8. Задается точка входа для контейнера, запускающая веб-сервер http на порту 3000 и SSH-сервер.

# **2. ОПИСАНИЕ СКРИПТОВ ЗАПУСКА ТЕСТОВ**

## **2.1. Скрипт run\_all.sh для запуска этапов тестирования**

Запуск этапов тестирования осуществляется с использованием bash-скрипта run\_all.sh, запускающего каждый из этапов по отдельности или совместно. В процессе выполнения тестов результаты записываются в монтированную в контейнер папку, создаваемую внутри скрипта.

При выполнении данного скрипта есть возможность передачи дополнительного аргумента, определяющего конкретный этап тестирования: flake8\_test (для форматирования), pylint\_test (для статического анализа), integration\_test (для интеграционных тестов), selenium\_test (для тестов с использованием веб-драйвера Selenium). Для каждого из этапов тестирования реализована отдельная функция, выполняющая их запуск.

В каждой из функций в первую очередь выводится информация о начале запуска конкретного этапа тестирования, после чего выполняются необходимые для запуска команды, а в конце выводится информация о завершении конкретного этапа тестирования. Результаты выполнения с помощью команды tee перенаправляются в файл .log контейнера (с названием самого этапа тестирования: flake8.log, pylint.log, integration.log, selenium.log).

## **2.2. Этап форматирования**

Для выполнения форматирования используется утилита flake8. Форматирование производится в соответствии с PEP 8.

## **2.3. Этап статического анализа**

При запуске данного этапа вначале создаётся файл \_\_init\_\_.py, необходимый для инициализации пакета и для корректной работы pylint.

Далее запускается статический анализ по 10 различным критериям. Для анализа используется конфигурационный файл pylintrc, в котором от линтера скрываются файлы интеграционного и селениум тестов и выполняется проверка для 10 критериев: typecheck, string, design, format, imports, main, refactoring, classes, variables, basic.

В конце данного этапа \_\_init\_\_.py удаляется.

## **2.4. Этап интеграционного тестирования**

Для запуска интеграционных тестов на проверку кодов возврата используется фреймворк pytest. Запускается скрипт integration\_test.py.

В ходе выполнения тестов с помощью библиотек unittest и requests выполняются различные HTTP-запросы (GET, POST) по различным эндпоинтам в веб-приложении, расположенном на 5000 порту. По данным эндпоинтам проверяется возвращаемые статус-коды.

## **2.5. Этап selenium тестирования**

В начале bash-скрипта запуска selenium\_test создаётся и настраивается виртуальный экран, с помощью виртуального фрейм-буфера xvfb. Это необходимо для запуска Google Chrome без создания графического окружения. Запуск самих тестов происходит напрямую с помощью запуска python3 main.py, сам файл main.py расположен в папке selenium\_test. В качестве начальной точки команде передается реализованный пакет selenium\_test, в результате которого выполняется следующий сценарий использования: «Создание ОПОП, проверка истории изменений и выдачи прав».

После выполнения функции уничтожается процесс виртуального экрана.

Данный скрипт может не срабатывать в контейнере, однако работает без ошибок на хост-машине.

# **3. ОПИСАНИЕ DOCKER-COMPOSE КОНФИГУРАЦИИ**

Конфигурация docker-compose описывается в файле docker-compose.yml и включает в себя описание запуска двух контейнеров (сервисов) – app (azar-app) с помощью образа из Dockerfile\_app и tester (azar-tester) с помощью образа из Dockerfile\_tester.

В процессе запуска контейнера с веб-приложением устанавливается ограничение на максимальную доступную память (140Мб), а также пробрасываются порты из контейнера на хост-машину:

* "127.0.0.1:3022:5000" – порт веб-приложения внутри контейнера (5000) становится доступен на хост-машине.

В процессе запуска контейнера с тестирующими скриптами монтируется директория на хосте внутрь контейнера для сохранения результатов тестирования, а также устанавливается ограничение на максимальную доступную память (140Мб) и пробрасываются порты из контейнера на хост-машину:

* "127.0.0.1:3022:5000" - порт SSH-сервера внутри контейнера (22) преобразуется в 3023 порт на хост-машине для возможности дальнейшего получения доступа по SSH, используя приватный ключ.

Для выполнения тестирования необходимо наличие запущенного контейнера app, соответствующая инструкция указана для контейнера tester (depends\_on: - app).

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были изучены технологии docker и docker-compose, они были применены на практике при создании и настройке двух контейнеров – azar-app (для запуска приложения) и azar-tester (для запуска процесса тестирования и SSH-сервера). Была изучена технология Selenium WebDriver, знания применены на практике при реализации автоматизированных тестов.

Процесс тестирования включает в себя несколько этапов, по типу форматирования с использованием flake8, статический анализ кода с помощью pylint, а также интеграционное тестирование демонстрационного веб-приложения на корректность кодов возврата и тестирование ИС ИОТ с помощью веб-драйвера Selenium на сценарий «Создание ОПОП, проверка истории изменений и выдачи прав». Запуск интеграционных тестов выполнялся при помощи фреймворка pytest, а запуск selenium-тестов напрямую через запуск скрипта с помощью python3.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Docker Docs [Электронный ресурс]. URL: <https://docs.docker.com/> (дата обращения: 02.05.2024)
2. Pylint 3.1.0 documentation [Электронный ресурс]. URL: <https://pylint.readthedocs.io/en/stable/> (дата обращения: 02.05.2024)
3. pytest: helps you write better programs [Электронный ресурс]. URL: [https://docs.pytest.org/en/8.0.x/](https://docs.pytest.org/en/8.0.x/%20) (дата обращения: 02.05.2024)
4. The Selenium Browser Automation Project | Selenium [Электронный ресурс]. URL: <https://www.selenium.dev/documentation/> (дата обращения: 11.04.2024)
5. Flake8 – Your Tool For Style Guide Enforcement [Электронный ресурс]. URL: <https://flake8.pycqa.org/en/latest/> (дата обращения: 02.05.2024)
6. ИС «ИОТ» [Электронный ресурс]. URL: <https://digital.etu.ru/trajectories> (дата обращения 02.05.2024)
7. Linux man pages [Электронный ресурс] URL: <https://linux.die.net/man/> (дата обращения 11.04.2024)

# **ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОЕКТА**

**/app/Dockerfile\_app:**

FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \

git=1:2.34.1-1ubuntu1.10 \

python3=3.10.6-1~22.04 \

python3-pip=22.0.2+dfsg-1ubuntu0.4

RUN git clone https://github.com/moevm/devops-examples.git

WORKDIR /devops-examples/EXAMPLE\_APP/

ADD requirements.txt .

RUN pip install -r requirements.txt

ADD add\_host.patch .

RUN patch main.py add\_host.patch

ENTRYPOINT ["python3", "main.py"]

**/tester/Dockerfile\_tester:**

FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \

    openssh-server=1:8.9p1-3ubuntu0.7 \

    git=1:2.34.1-1ubuntu1.10 \

    python3=3.10.6-1~22.04 \

    python3-pip=22.0.2+dfsg-1ubuntu0.4 \

    wget=1.21.2-2ubuntu1 \

    xvfb=2:21.1.4-2ubuntu1.7~22.04.10

RUN wget https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb \

  && dpkg -i google-chrome-stable\_current\_amd64.deb; apt-get -fy install

RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd\_config

COPY ssh-keys/id\_rsa.pub /root/.ssh/authorized\_keys

RUN git clone https://github.com/moevm/devops-examples.git

WORKDIR /devops-examples/EXAMPLE\_APP/

COPY tests ./tests

COPY requirements.txt ./

RUN pip install -r requirements.txt

ENTRYPOINT ["bash", "-c", "service ssh start && python3 -m http.server 3000"]

**/docker-compose.yml:**

version: "3"

services:

  app:

    container\_name: azar-app

    build:

      context: ./app

      dockerfile: Dockerfile\_app

    ports:

      - "127.0.0.1:3022:5000"

    deploy:

      resources:

        limits:

          memory: 140M

  tester:

    container\_name: azar-tester

    depends\_on:

      - app

    build:

      context: ./tester

      dockerfile: Dockerfile\_tester

    ports:

      - "127.0.0.1:3023:22"

    volumes:

      - ./tester/tests/test\_res/:/devops-examples/EXAMPLE\_APP/tests/test\_res

    deploy:

      resources:

        limits:

          memory: 140M

**/app/add\_host.patch:**

78c78

<     app.run(debug = True)

---

>     app.run(host='0.0.0.0', debug = True)

**/tester/tests/.env:**

TEST\_STAGES="flake8 pylint integration selenium"

**/tester/tests/run\_all.sh:**

#!/bin/bash

TESTS\_FOLDER='/devops-examples/EXAMPLE\_APP/tests'

mkdir -p ${TESTS\_FOLDER}/test\_res;

function flake8\_test() {

    echo "STARTED FLAKE8 TEST";

    flake8 . \

    > >(tee -a "${TESTS\_FOLDER}/test\_res/flake8.log") \

    2>&1;

    echo "FINISHED FLAKE8 TEST";

}

function pylint\_test() {

    echo "STARTED PYLINT TEST";

    touch \_\_init\_\_.py;

    pylint $(pwd) -v --rcfile=${TESTS\_FOLDER}/pylintrc \

    > >(tee -a "${TESTS\_FOLDER}/test\_res/pylint.log") \

    2>&1;

    rm \_\_init\_\_.py;

    echo "FINISHED PYLINT TEST";

}

function integration\_test() {

    echo "STARTED INTEGRATION TEST";

    pytest -s -v ${TESTS\_FOLDER}/integration\_test.py \

    > >(tee -a "${TESTS\_FOLDER}/test\_res/integration.log") \

    2>&1;

    echo "FINISHED INTEGRATION TEST";

}

function selenium\_test() {

    echo "STARTED SELENIUM TEST";

    exec -a xvfb-run Xvfb :1 -screen 0 1920x1080x16 &> xvfb.log  &

    DISPLAY=:1.0

    export DISPLAY

    python3 ${TESTS\_FOLDER}/selenium\_test/main.py \

    > >(tee -a "${TESTS\_FOLDER}/test\_res/selenium.log") \

    2>&1;

    kill $(pgrep -f xvfb-run)

    echo "FINISHED SELENIUM TEST";

}

if [ $# -eq 0 ]

  then

    flake8\_test

    pylint\_test

    integration\_test

    selenium\_test

  else

    $1

fi

**/tester/tests/pylintrc:**

[MAIN]

ignore=integration\_test.py, selenium\_test

[MESSAGES CONTROL]

disable = all

enable = typecheck, string, design, format, imports, main, refactoring, classes, variables, basic

**/tester/tests/integration\_test.py:**

import unittest

import os

import requests

class IntegrationTest(unittest.TestCase):

    def \_\_init\_\_(self, method: str = "runTest") -> None:

        super().\_\_init\_\_(method)

        self.base\_url = os.environ.get("BASE\_URL", "http://app:5000")

    def test\_endpoint\_index\_get(self):

        res = requests.get(self.base\_url)

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_upload\_get(self):

        res = requests.get(f'{self.base\_url}/upload')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_upload\_post(self):

        with open('temp.txt', 'w') as f:

            f.write('Test file')

        files = {'file': open('temp.txt', 'rb')}

        res = requests.post(f'{self.base\_url}/upload', files=files)

        os.remove('temp.txt')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_download\_get(self):

        res = requests.get(f'{self.base\_url}/download/temp.txt')

        self.assertEqual(res.status\_code, 404)

    def test\_endpoint\_files\_get(self):

        res = requests.get(f'{self.base\_url}/files')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_to\_files\_get\_redirect(self):

        res = requests.get(f'{self.base\_url}/to\_files')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_success\_get(self):

        res = requests.get(f'{self.base\_url}/success/python')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_increment\_get\_exception(self):

        self.assertRaises(

            requests.exceptions.TooManyRedirects,

            requests.get, f'{self.base\_url}/increment/50'

        )

    def test\_endpoint\_check\_even\_get\_on\_even(self):

        res = requests.get(f'{self.base\_url}/check\_even/4')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_check\_even\_get\_on\_odd(self):

        res = requests.get(f'{self.base\_url}/check\_even/3')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_even\_get\_on\_even(self):

        res = requests.get(f'{self.base\_url}/even/4')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_odd\_get\_on\_odd(self):

        res = requests.get(f'{self.base\_url}/odd/3')

        self.assertEqual(res.status\_code, 200)

    def test\_endpoint\_login\_post\_good\_creds(self):

        data = {'name': 'user', 'password': 'pass'}

        res = requests.post(f'{self.base\_url}/login', data=data)

        self.assertEqual(res.status\_code, 401)

    def test\_endpoint\_login\_post\_bad\_creds(self):

        data = {'name': 'wrong\_user', 'password': 'wrong\_pass'}

        res = requests.post(f'{self.base\_url}/login', data=data)

        self.assertEqual(res.status\_code, 401)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**/tester/tests/selenium\_test/\_\_init\_\_.py:**

**/tester/tests/selenium\_test/main.py:**

import time

from selenium.webdriver.common.by import By

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver import Chrome, ActionChains

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

def wait\_and\_click(browser, selector):

    element = WebDriverWait(browser, 10).until(

        EC.element\_to\_be\_clickable((By.CSS\_SELECTOR, selector)))

    element.click()

opts = Options()

opts.add\_argument("--no-sandbox")

opts.add\_argument('--disable-dev-shm-usage')

browser = Chrome(service=Service(

    ChromeDriverManager().install()), options=opts)

browser.implicitly\_wait(10)

browser.get('https://dev.digital.etu.ru/trajectories-test/auth')

# Close modal

browser.find\_element(By.ID, "devServerModalId\_\_\_BV\_modal\_content\_")

time.sleep(1)

closeModal = browser.find\_element(

    By.CSS\_SELECTOR, "#devServerModalId\_\_\_BV\_modal\_footer\_ > button")

closeModal.click()

# Login through etu id

browser.find\_element(

    By.CSS\_SELECTOR, '.card-text > div > button:first-child').click()

# Send login form creds and wait

browser.find\_element(By.CSS\_SELECTOR, "input[type=email]").send\_keys(

    "lolovishka@mail.ru")

browser.find\_element(

    By.CSS\_SELECTOR, "input[type=password]").send\_keys("Ugngbnm19")

browser.find\_element(By.CSS\_SELECTOR, "button[type=submit]").click()

time.sleep(1)

# Authorize btn click

browser.find\_element(By.CSS\_SELECTOR, "button[type=submit]").click()

# Remove cookies notification

browser.find\_element(By.CSS\_SELECTOR, ".btn.mb-1.btn-outline-primary").click()

# Click on open sidebar menu

wait\_and\_click(browser, "button.mr-2")

# Click btn "authorize as other user"

wait\_and\_click(browser, "a[href=\"/trajectories-test/admin/fake\"]")

# Open popup filter by id

browser.find\_element(

    By.CSS\_SELECTOR, "span.ag-header-icon.ag-header-cell-menu-button:first-of-type").click()

# Enter id 1305 in input

browser.find\_element(

    By.CSS\_SELECTOR, "div.ag-popup-child input").send\_keys("1305")

# Double-click on user row

userRow = browser.find\_element(

    By.CSS\_SELECTOR, "div[ref=\"eCenterContainer\"] > div")

action = ActionChains(browser)

action.double\_click(on\_element=userRow)

action.perform()

# Wait for main page to load

WebDriverWait(browser, 10).until(

    EC.presence\_of\_element\_located((By.CSS\_SELECTOR, 'img[src="/trajectories-test/logo-leti.png"]')))

# === CREATING OPOP ===

# Open sidebar menu by btn click

wait\_and\_click(browser, "button.btn.mr-2.btn-primary.rounded-pill.collapsed")

# Click opop btn

wait\_and\_click(browser, "a[href=\"/trajectories-test/documents/opop-list\"]")

# Click add opop btn

wait\_and\_click(browser, ".btn.mx-1:first-of-type")

# Select speciality name

browser.find\_element(

    By.CSS\_SELECTOR, ".form-group.valid.required.field-multiselect:nth-of-type(1) > div > div").click()

browser.find\_element(

    By.CSS\_SELECTOR, ".multiselect\_\_element:first-of-type").click()

time.sleep(1)

# Select speciality plan

browser.find\_element(

    By.CSS\_SELECTOR, ".form-group.valid.required.field-multiselect:nth-of-type(2) > div > div").click()

browser.find\_element(

    By.CSS\_SELECTOR, ".multiselect\_\_element:nth-of-type(6)").click()

time.sleep(1)

# Click on add btn

browser.find\_element(

    By.CSS\_SELECTOR, "#creationModalId\_\_\_BV\_modal\_content\_ .btn.btn-primary").click()

# Wait for blank to load

WebDriverWait(browser, 10).until(EC.presence\_of\_element\_located(

    (By.CSS\_SELECTOR, ".mx-1.status-text")))

# === / CREATING OPOP ===

# === GIVING RIGHTS ===

# go to rights tab

browser.find\_element(By.CSS\_SELECTOR, 'a[aria-posinset="13"]').click()

time.sleep(1)

# click on add btn

browser.find\_element(By.CSS\_SELECTOR, '.btn.float-right.btn-primary').click()

time.sleep(1)

# select first user

browser.find\_element(

    By.CSS\_SELECTOR, '.ag-center-cols-container > div:first-of-type').click()

time.sleep(1)

# click checkbox first option

browser.find\_element(

    By.CSS\_SELECTOR, '.custom-control.custom-control-inline.custom-checkbox:first-of-type').click()

time.sleep(1)

# click on give rights btn

browser.find\_element(

    By.CSS\_SELECTOR, '#addRightsModal\_\_\_BV\_modal\_footer\_ > button:first-of-type').click()

time.sleep(1)

# === / GIVING RIGHTS ===

# === CHECKING HISTORY ===

browser.find\_element(By.CSS\_SELECTOR, 'a[aria-postinset="15"]').click()

time.sleep(1)

# === / CHECKING HISTORY ===