НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп'ютерних систем

КУРСОВА РОБОТА

з дисципліни "Компоненти програмної інженерії"

Виконала: Євтушенко Вікторія Павлівна
Група: КП-03

Допущено до захисту

_____

1 семестр 2022/2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

Узгоджено
Керівник роботи

_____/Погорелов В.В./

ЗАХИЩЕНА
"__"_____2023р.

з оцінкою_____

_____/Погорелов В.В./

Файлова система

Виконавиця роботи

Євтушенко Вікторія Павлівна

_____2023р.

Завдання

1. Розробити HTTP додаток, який дозволить виконувати операції над елементами файлової системи.
2. Запустити додаток у Docker контейнері.
3. Створити клієнт до розробленої програми, використовуючи Client Line Interface.
4. Написати тести до клієнту.

Інструменти розробки

Для виконання даного завдання використовувалися наступні технології розробки:

1. **Python** – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.
2. **Docker** - інструментарій для управління ізольованими Linux-контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.
3. **Flask** - мікрофреймворк для вебдодатків, створений з використанням Python. У ньому відсутній рівень абстракції для роботи з базою даних, перевірки форм або інші компоненти, які надають широковживані функції за допомогою сторонніх бібліотек. Однак, Flask має підтримку розширень, які надають додаткові властивості таким чином, наче вони були доступні у Flask із самого початку. Існують розширення для встановлення об'єктно-реляційних зв'язків, перевірки форм, контролю процесу завантаження, підтримки

різноманітних відкритих технологій аутентифікації та декількох поширених засобів для фреймворку.

4. **Click** - пакет Python для створення інтерфейсів командного рядка, які можна компонувати, з мінімальною кількістю коду.
5. **Requests** - бібліотека HTTP для мови програмування Python. Мета – зробити HTTP-запити простішими та зручнішими для людини.
6. **Pytest** – фреймворк, що дозволяє легко писати невеликі, читабельні тести та може масштабуватися для підтримки складного функціонального тестування для програм і бібліотек.

Код

main.py

```
from flask import Flask, jsonify, request
from directory import Directory
from binaryFile import BinaryFile
from logTextFile import LogTextFile
from bufferFile import BufferFile

app = Flask(__name__)

root = Directory('root', None, 100)
trash = []


@app.route('/cleanup', methods=['POST'])
def cleanup():
    global root, trash
    root = Directory('root', None, 100)
    trash = []
    return None


def directory_post(request):
    if any(element.name == request.args.get('name') for element in
root.list) or request.args.get('name') == 'root':
        return jsonify({'message': 'Such directory already
exists.'}), 400
    dir = Directory(request.args.get('name'), root,
int(request.args.get('max_size')))
    return jsonify({
        'message': 'Created successfully.',
        'directory': {
            'parent_directory': dir.parent_directory.name,
            'name': dir.name,
            'max_size': int(dir.max_size)
        }
    }), 201


def directory_get(request):
    if any(dir.name == request.args.get('name') for dir in
root.list) or request.args.get('name') == 'root':
        if request.args.get('name') == 'root':
            dir = root
```

```
        else:
            dir = next(element for element in root.list if
element.name == request.args.get('name'))
        return jsonify({
            'message': 'Read successfully.',
            'directory': {
                'parent_directory': dir.parent_directory.name,
                'name': dir.name,
                'max_size': dir.max_size
            }
        }), 200
    return jsonify({'message': 'Directory does not exist.', }), 400


def directory_patch(request):
    if any(dir.name == request.args.get('name') for dir in
root.list):
        dir = next(element for element in root.list if element.name
== request.args.get('name'))
        dir.move_directory(root)
        return jsonify({
            'message': 'Moved successfully.',
            'directory': {
                'parent_directory': dir.parent_directory.name,
                'name': dir.name,
                'max_size': dir.max_size
            }
        }), 200
    return jsonify({'message': 'Directory does not exist.', }), 400


def directory_delete(request):
    if request.args.get('name') not in trash and any(dir.name ==
request.args.get('name') for dir in root.list):
        dir = next(x for x in root.list if x.name ==
request.args.get('name'))
        dir.delete()
        trash.append(request.args.get('name'))
        return jsonify({
            'message': 'Deleted successfully.',
        }), 200
    return jsonify({'message': 'Deleting failed.'}), 400


@app.route('/directory', methods=['POST', 'GET', 'PATCH',
'DELETE'])
def directory():
    if request.method == 'POST':
        return directory_post(request)
    elif request.method == 'GET':
        return directory_get(request)
    elif request.method == 'PATCH':
        return directory_patch(request)
    else:
        return directory_delete(request)


def binary_file_post(request):
    if any(element.name == request.args.get('name') for element in
root.list):
        return jsonify({
```

```
            'message': 'Such file already exists.',
        }), 400
    file = BinaryFile(request.args.get('name'), root,
request.args.get('contents'))
    return jsonify({
        'message': 'Created successfully.',
        'file': {
            'directory': file.directory.name,
            'name': file.name,
            'contents': file.contents
        }
    }), 201


def binary_file_get(request):
    if any(file.name == request.args.get('name') for file in
root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        return jsonify({
            'message': 'Read successfully.',
            'file': {
                'directory': file.directory.name,
                'name': file.name,
                'contents': file.contents
            }
        }), 200
    return jsonify({'message': "Such file does not exist."}), 400


def binary_file_delete(request):
    if request.args.get('name') not in trash and any(file.name ==
request.args.get('name') for file in root.list):
        file = next(x for x in root.list if x.name ==
request.args.get('name'))
        file.delete()
        trash.append(request.args.get('name'))
        return jsonify({'message': "Deleted successfully."}), 200
    return jsonify({"message": "Deleting failed."}), 400


def binary_file_patch(request):
    if any(file.name == request.args.get('name') for file in
root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        file.move_binary_file(root)
        return jsonify({
            'message': 'Moved successfully.',
            'file': {
                'directory': file.directory.name,
                'name': file.name,
                'content': file.contents
            }
        }), 200
    return jsonify({'message': "Such file does not exist."}), 400


@app.route('/binaryfile', methods=['POST', 'GET', 'PATCH',
'DELETE'])
def binaryfile():
```

```
    # file = BinaryFile(request.args.get('name'), root,
request.args.get('contents'))
    if request.method == 'POST':
        return binary_file_post(request)
    elif request.method == 'GET':
        return binary_file_get(request)
    elif request.method == 'PATCH':
        return binary_file_patch(request)
    else:
        return binary_file_delete(request)


def logTextFilePost(request):
    if any(element.name == request.args.get('name') for element in
root.list):
        return jsonify({
            'message': "Such file already exists.",
        }), 400
    file = LogTextFile(request.args.get('name'), root,
request.args.get('contents'))
    return jsonify({
        'message': "Created successfully.",
        'file': {
            'directory': file.directory.name,
            'name': file.name,
            'contents': file.contents
        }
    }), 201


def log_text_file_get(request):
    if any(file.name == request.args.get('name') for file in
root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        return jsonify({
            'message': "Read successfully.",
            'file': {
                'directory': file.directory.name,
                'name': file.name,
                'contents': file.contents
            }
        }), 200
    return jsonify({'message': 'Such file does not exist.'}), 400


def log_text_file_patch(request):
    if any(file.name == request.args.get('name') for file in
root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        if request.args.get('directory'):
            file.move_log_file(root)
            return jsonify({
                'message': "Moved successfully.",
                'file': {
                    'directory': file.directory.name,
                    'name': file.name,
                    'contents': file.contents
                }
            }), 200
```

```
        elif request.args.get('append'):
            file.append_line(request.args.get('append'))
            return jsonify({
                'message': "Line appended successfully.",
                'file': {
                    'directory': file.directory.name,
                    'name': file.name,
                    'contents': file.contents
                }
            }), 201
        return jsonify({'message': 'Wrong request.'}), 400
    return jsonify({'message': 'File does not exist.'}), 400


def log_text_file_delete(request):
    if request.args.get('name') not in trash and any(file.name ==
request.args.get('name') for file in root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        file.delete()
        trash.append(request.args.get('name'))
        return jsonify({'message': "Deleted successfully."}), 200
    return jsonify({'message': 'Deleting failed.'}), 400


@app.route('/logtextfile', methods=['POST', 'GET', 'PATCH',
'DELETE'])
def logtextfile():
    if request.method == 'POST':
        return logTextFilePost(request)
    elif request.method == 'GET':
        return log_text_file_get(request)
    elif request.method == 'PATCH':
        return log_text_file_patch(request)
    else:
        return log_text_file_delete(request)


def buffer_file_post(request):
    if any(element.name == request.args.get('name') for element in
root.list):
        return jsonify({'message': 'Such file already exists.'}),
400
    file = BufferFile(request.args.get('name'), root,
int(request.args.get('max_size')))
    return jsonify({
        'message': 'Created successfully.',
        'file': {
            'directory': file.directory.name,
            'name': file.name,
            'max_size': file.max_size,
            'contents': file.contents
        }
    }), 201


def buffer_file_get(request):
    if any(file.name == request.args.get('name') for file in
root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
```

```
        return jsonify({
            'message': 'Read successfully.',
            'file': {
                'directory': file.directory.name,
                'name': file.name,
                'max_size': file.max_size,
                'contents': file.contents,
            }
        }), 200
    return jsonify({'message': 'Such file does not exist.'}), 400


def buffer_file_patch(request):
    if any(file.name == request.args.get('name') for file in
root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        if request.args.get('directory'):
            file.move_buffer_file(root)
            return jsonify({
                'message': 'Moved successfully.',
                'file': {
                    'directory': file.directory.name,
                    'name': file.name,
                    'max_size': file.max_size,
                    'contents': file.contents,
                }
            }), 200
        if request.args.get('push'):
            file.push_element(request.args.get('push'))
            return jsonify({
                'message': 'Element pushed successfully.',
                'file': {
                    'directory': file.directory.name,
                    'name': file.name,
                    'max_size': file.max_size,
                    'contents': file.contents,
                }
            }), 201
        if request.args.get('consume'):
            if len(file.contents) > 0:
                file.consume_element(request.args.get('consume'))
                return jsonify({
                    'message': 'Element consumed successfully.',
                    'file': {
                        'directory': file.directory.name,
                        'name': file.name,
                        'max_size': file.max_size,
                        'contents': file.contents,
                    }
                }), 200
        return jsonify({'message': 'Wrong request.'}), 400
    return jsonify({'message': 'Such file does not exist.'}), 400


def buffer_file_delete(request):
    if request.args.get('name') not in trash and any(file.name ==
request.args.get('name') for file in root.list):
        file = next(element for element in root.list if
element.name == request.args.get('name'))
        file.delete()
```

```
        trash.append(request.args.get('name'))
        return jsonify({'message': 'Deleted successfully.'}), 200
    return jsonify({'message': 'Deleting failed.'}), 400


@app.route('/bufferfile', methods=['POST', 'GET', 'PATCH',
'DELETE'])
def bufferfile():
    if request.method == 'POST':
        return buffer_file_post(request)
    elif request.method == 'GET':
        return buffer_file_get(request)
    elif request.method == 'PATCH':
        return buffer_file_patch(request)
    else:
        return buffer_file_delete(request)


if __name__ == '__main__':
    app.run(host="127.0.0.1", port=5000, debug=True)
```

client.py

```
from flask import Flask
import requests
import click

app = Flask(__name__)


@click.group()
def cli():
    pass


@click.command(name="create_directory")
@click.argument('parent_directory')
@click.argument('name')
@click.argument('max_size')
def create_directory(parent_directory, name, max_size):
    for_post = {
        'parent_directory': parent_directory,
        'name' : name,
        'max_size' : max_size
    }
    response = requests.post("http://127.0.0.1:5000/directory",
params=for_post)
    print(response.json())
    print(response.status_code)


@click.command(name="delete_directory")
@click.argument('name')
def delete_directory(name):
    for_delete = {'name': name}
    response = requests.delete("http://127.0.0.1:5000/directory",
params=for_delete)
    print(response.json())
    print(response.status_code)
```

```
@click.command(name="list_directory")
@click.argument('parent_directory')
@click.argument('name')
@click.argument('max_size')
def list_directory(parent_directory, name, max_size):
    for_get = {
        'parent_directory': parent_directory,
        'name': name,
        'max_size': max_size
    }
    response = requests.get("http://127.0.0.1:5000/directory",
params=for_get)
    print(response.json())
    print(response.status_code)


@click.command(name="move_directory")
@click.argument('parent_directory')
@click.argument('name')
def move_directory(parent_directory, name):
    for_patch = {
        'parent_directory': parent_directory,
        'name': name
    }
    response = requests.patch("http://127.0.0.1:5000/directory",
params=for_patch)
    print(response.json())
    print(response.status_code)


@click.command(name="create_binary_file")
@click.argument('directory')
@click.argument('name')
@click.argument('contents')
def create_binary_file(directory, name, contents):
    for_post = {
        'directory': directory,
        'name': name,
        'contents': contents
    }
    response = requests.post("http://127.0.0.1:5000/binaryfile",
params=for_post)
    print(response.json())
    print(response.status_code)


@click.command(name="delete_binary_file")
@click.argument('name')
def delete_binary_file(name):
    for_delete = {'name': name}
    response = requests.delete("http://127.0.0.1:5000/binaryfile",
params=for_delete)
    print(response.json())
    print(response.status_code)


@click.command(name="move_binary_file")
@click.argument('directory')
@click.argument('name')
def move_binary_file(directory, name):
    for_patch = {
```

```
        'directory': directory,
        'name': name,
    }
    response = requests.patch("http://127.0.0.1:5000/binaryfile",
params=for_patch)
    print(response.json())
    print(response.status_code)


@click.command(name="read_binary_file")
@click.argument('directory')
@click.argument('name')
@click.argument('contents')
def read_binary_file(directory, name, contents):
    for_get = {
        'directory': directory,
        'name': name,
        'contents': contents
    }
    response = requests.get("http://127.0.0.1:5000/binaryfile",
params=for_get)
    print(response.json())
    print(response.status_code)


@click.command(name="create_buffer_file")
@click.argument('directory')
@click.argument('name')
@click.argument('max_size')
def create_buffer_file(directory, name, max_size):
    for_post = {
        'directory': directory,
        'name': name,
        'max_size': max_size,
    }
    response = requests.post("http://127.0.0.1:5000/bufferfile",
params=for_post)
    print(response.json())
    print(response.status_code)


@click.command(name="delete_buffer_file")
@click.argument('name')
def delete_buffer_file(name):
    for_delete = {'name': name}
    response = requests.delete("http://127.0.0.1:5000/bufferfile",
params=for_delete)
    print(response.json())
    print(response.status_code)


@click.command(name="move_buffer_file")
@click.argument('directory')
@click.argument('name')
def move_buffer_file(directory, name):
    for_patch = {
        'directory': directory,
        'name': name
    }
    response = requests.patch("http://127.0.0.1:5000/bufferfile",
params=for_patch)
```

```
    print(response.json())
    print(response.status_code)


@click.command(name="read_buffer_file")
@click.argument('directory')
@click.argument('name')
@click.argument('max_size')
def read_buffer_file(directory, name, max_size):
    for_get = {
        'directory': directory,
        'name': name,
        'max_size' : max_size,
    }
    response = requests.get("http://127.0.0.1:5000/bufferfile",
params=for_get)
    print(response.json())
    print(response.status_code)


@click.command(name="push_buffer_file")
@click.argument('name')
@click.argument('element')
def push_buffer_file(name, element):
    for_patch = {
        'name' : name,
        'push' : element
    }
    response = requests.patch("http://127.0.0.1:5000/bufferfile",
params=for_patch)
    print(response.json())
    print(response.status_code)


@click.command(name="consume_buffer_file")
@click.argument('name')
@click.argument('element')
def consume_buffer_file(name, element):
    for_patch = {
        'name': name,
        'consume': element,
    }
    response = requests.patch("http://127.0.0.1:5000/bufferfile",
params=for_patch)
    print(response.json())
    print(response.status_code)


@click.command(name="create_log_text_file")
@click.argument('directory')
@click.argument('name')
@click.argument('contents')
def create_log_text_file(directory, name, contents):
    for_post = {
        'directory': directory,
        'name': name,
        'contents': contents
    }
    response = requests.post("http://127.0.0.1:5000/logtextfile",
params=for_post)
    print(response.json())
```

```
    print(response.status_code)


@click.command(name="delete_log_text_file")
@click.argument('name')
def delete_log_text_file(name):
    for_delete = {'name': name}
    response = requests.delete("http://127.0.0.1:5000/logtextfile",
params=for_delete)
    print(response.json())
    print(response.status_code)


@click.command(name="move_log_text_file")
@click.argument('directory')
@click.argument('name')
def move_log_text_file(directory, name):
    for_patch = {
        'directory': directory,
        'name': name,
    }
    response = requests.patch("http://127.0.0.1:5000/bufferfile",
params=for_patch)
    print(response.json())
    print(response.status_code)


@click.command(name="read_log_text_file")
@click.argument('directory')
@click.argument('name')
def read_log_text_file(directory, name):
    for_get = {
        'directory': directory,
        'name': name,
    }
    response = requests.get("http://127.0.0.1:5000/logtextfile",
params=for_get)
    print(response.json())
    print(response.status_code)


@click.command(name="append_log_text_file")
@click.argument('name')
@click.argument('append')
def append_log_text_file(name, append):
    for_patch = {
        'name' : name,
        'append' : append
    }
    response = requests.patch("http://127.0.0.1:5000/logtextfile",
params=for_patch)
    print(response.json())
    print(response.status_code)


cli.add_command(create_directory)
cli.add_command(delete_directory)
cli.add_command(move_directory)
cli.add_command(list_directory)

cli.add_command(create_binary_file)
```

```
cli.add_command(delete_binary_file)
cli.add_command(move_binary_file)
cli.add_command(read_binary_file)

cli.add_command(create_buffer_file)
cli.add_command(delete_buffer_file)
cli.add_command(move_buffer_file)
cli.add_command(push_buffer_file)
cli.add_command(consume_buffer_file)

cli.add_command(create_log_text_file)
cli.add_command(delete_log_text_file)
cli.add_command(move_log_text_file)
cli.add_command(read_log_text_file)
cli.add_command(append_log_text_file)


if __name__ == '__main__':
    cli()
```

client_test.py

```
import requests
import pytest
from client import *
import click
from click.testing import CliRunner

@pytest.fixture(autouse=True)
def run_before_and_after_tests(tmpdir):
    try:
        yield  # this is where the testing happens
    finally:
        requests.post("http://127.0.0.1:5000/cleanup")


def test_if_directory_creatable():
    runner = CliRunner()
    result = runner.invoke(create_directory, ['root', 'child',
'5'])
    assert result.exit_code == 0

def test_if_directory_movable():
    requests.post("http://127.0.0.1:5000/directory?
name=testing&parent_directory=root&max_size=20")
    runner = CliRunner()
    result = runner.invoke(move_directory, ['root', 'testing'])
    assert result.exit_code == 0


def test_if_directory_deletable():
    requests.post("http://127.0.0.1:5000/directory?
name=child&parent_directory=root&max_size=20")
    runner = CliRunner()
    result = runner.invoke(delete_directory, ['child'])
    assert result.exit_code == 0


def test_if_directory_reads():
    requests.post("http://127.0.0.1:5000/directory?
name=child&parent_directory=root&max_size=20")
```

```
    runner = CliRunner()
    result = runner.invoke(list_directory, ['root', 'child', '20'])
    assert result.exit_code == 0


def test_if_binary_file_creatable():
    runner = CliRunner()
    result = runner.invoke(create_binary_file, ['root',
'binaryfile', 'test'])
    assert result.exit_code == 0


def test_if_binary_file_movable():
    requests.post("http://127.0.0.1:5000/binaryfile?
name=binaryfile1&directory=root&contents=test")
    runner = CliRunner()
    result = runner.invoke(move_binary_file, ['root',
'binaryfile'])
    assert result.exit_code == 0


def test_if_binary_file_readable():
    requests.post("http://127.0.0.1:5000/binaryfile?
name=binaryfile1&directory=root&contents=test")
    runner = CliRunner()
    result = runner.invoke(read_binary_file, ['root',
'binaryfile1', 'test'])
    assert result.exit_code == 0


def test_if_binary_file_deletable():
    requests.post("http://127.0.0.1:5000/binaryfile?
name=binaryfile1&directory=root&contents=test")
    runner = CliRunner()
    result = runner.invoke(delete_binary_file, ['binaryfile1'])
    assert result.exit_code == 0


def test_if_buffer_file_creatable():
    runner = CliRunner()
    result = runner.invoke(create_buffer_file, ['root',
'bufferfile', '100'])
    assert result.exit_code == 0


def test_if_buffer_file_movable():
    requests.post("http://127.0.0.1:5000/bufferfile?
directory=root&max_size=100&name=bufferfile1")
    runner = CliRunner()
    result = runner.invoke(move_buffer_file, ['root',
'bufferfile1'])
    assert result.exit_code == 0


def test_if_element_pushes():
    requests.post("http://127.0.0.1:5000/bufferfile?
directory=root&max_size=100&name=bufferfile")
    runner = CliRunner()
    result = runner.invoke(push_buffer_file, ['bufferfile',
'test'])
    assert result.exit_code == 0
```

```
def test_if_element_consumes():
    requests.post("http://127.0.0.1:5000/bufferfile?
directory=root&max_size=100&name=bufferfile")
    requests.patch("http://127.0.0.1:5000/bufferfile?
name=bufferfile&push=test")
    runner = CliRunner()
    result = runner.invoke(consume_buffer_file, ['bufferfile',
'test'])
    assert result.exit_code == 0


def test_if_buffer_file_deletable():
    requests.post("http://127.0.0.1:5000/bufferfile?
max_size=100&name=bufferfile")
    runner = CliRunner()
    result = runner.invoke(delete_buffer_file, ['bufferfile'])
    assert result.exit_code == 0


def test_if_buffer_file_readable():
    requests.post("http://127.0.0.1:5000/bufferfile?
directory=root&max_size=100&name=bufferfile")
    requests.patch("http://127.0.0.1:5000/bufferfile?
name=bufferfile&push=test")
    runner = CliRunner()
    result = runner.invoke(read_buffer_file, ['root', 'bufferfile',
'100'])
    assert result.exit_code == 0


def test_create_log_text_file():
    runner = CliRunner()
    result = runner.invoke(create_log_text_file, ['root',
'logtextfile', 'test'])
    assert result.exit_code == 0


def test_if_log_file_movable():
    requests.post("http://127.0.0.1:5000/logtextfile?
name=logtextfile1&directory=root&contents=test")
    runner = CliRunner()
    result = runner.invoke(move_log_text_file, ['root',
'logtextfile'])
    assert result.exit_code == 0
    requests.post("http://127.0.0.1:5000/cleanup")


def test_if_log_file_readable():
    requests.post("http://127.0.0.1:5000/logtextfile?
name=logtextfile1&directory=root&contents=test")
    runner = CliRunner()
    result = runner.invoke(read_log_text_file, ['root',
'logtextfile1'])
    assert result.exit_code == 0


def test_if_line_appends():
    requests.post("http://127.0.0.1:5000/logtextfile?
name=logtextfile&directory=root&contents=test")
```

```
    runner = CliRunner()
    result = runner.invoke(append_log_text_file, ['logtextfile',
'test'])
    assert result.exit_code == 0


def test_if_log_file_deletable():
    requests.post("http://127.0.0.1:5000/logtextfile?
name=logtextfile&directory=root&contents=test")
    runner = CliRunner()
    result = runner.invoke(delete_log_text_file, ['logtextfile'])
    assert result.exit_code == 0
```

directory.py

```
class Directory:

    def __init__(self, name: str, parent_directory, max_size: int):
        if (parent_directory is not None) and
(parent_directory.count < parent_directory.max_size):
            parent_directory.count += 1
            parent_directory.list.append(self)
            self.name = name
            self.parent_directory = parent_directory
            self.max_size = max_size
            self.count = 0
            self.list = []
        elif parent_directory is None:
            self.name = name
            self.parent_directory = parent_directory
            self.max_size = max_size
            self.count = 0
            self.list = []
        elif parent_directory.count >= parent_directory.max_size:
            print("This directory is already full")
            return

    def delete(self):
        print("Deleting " + self.name)
        index = self.parent_directory.list.index(self)
        self.parent_directory.list.pop(index)
        del self
        print("Deleted successfully")

    def list_elements(self):
        resulting_list = 'In ' + self.name + ': '
        for element in self.list:
            resulting_list += element.name + '; '
        print(resulting_list)

    def move_directory(self, location):
        if self.parent_directory is None:
            print("Impossible to move root directory")
        if location.count < location.max_size + self.count + 1:
            self.parent_directory.count -= self.count + 1
            index = self.parent_directory.list.index(self)
            self.parent_directory.list.pop(index)
            self.parent_directory = location
            self.parent_directory.list.append(self)
            self.parent_directory.count += self.count + 1
        else:
```

```
            print("Directory is already full")
            return
```

binaryFile.py

```
class BinaryFile:

    def __init__(self, name: str, directory, contents: str):
        if directory.count < directory.max_size:
            self.directory = directory
            self.directory.count = 1
            self.name = name
            self.contents = contents
            self.directory.list.append(self)
        else:
            print("This directory is already full")
            return

    def delete(self):
        print("Deleting " + self.name)
        index = self.directory.list.index(self)
        self.directory.list.pop(index)
        del self
        print("Deleted successfully")

    def move_binary_file(self, location):
        if location.count < location.max_size:
            self.directory.count -= 1
            index = self.directory.list.index(self)
            self.directory.list.pop(index)
            self.directory = location
            self.directory.list.append(self)
            self.directory.count += 1
        else:
            print("Directory is already full")
            return

    def read_file(self):
        print(self.contents)
```

logTextFile.py

```
class LogTextFile:

    def __init__(self, name: str, directory, contents: str):
        if directory.count < directory.max_size:
            self.directory = directory
            self.directory.count = 1
            self.name = name
            self.contents = contents
            self.directory.list.append(self)
        else:
            print("This directory is already full")
            return

    def delete(self):
        print("Deleting " + self.name)
        del self
        print("Deleted successfully")

    def move_log_file(self, location):
        if location.count < location.max_size:
```

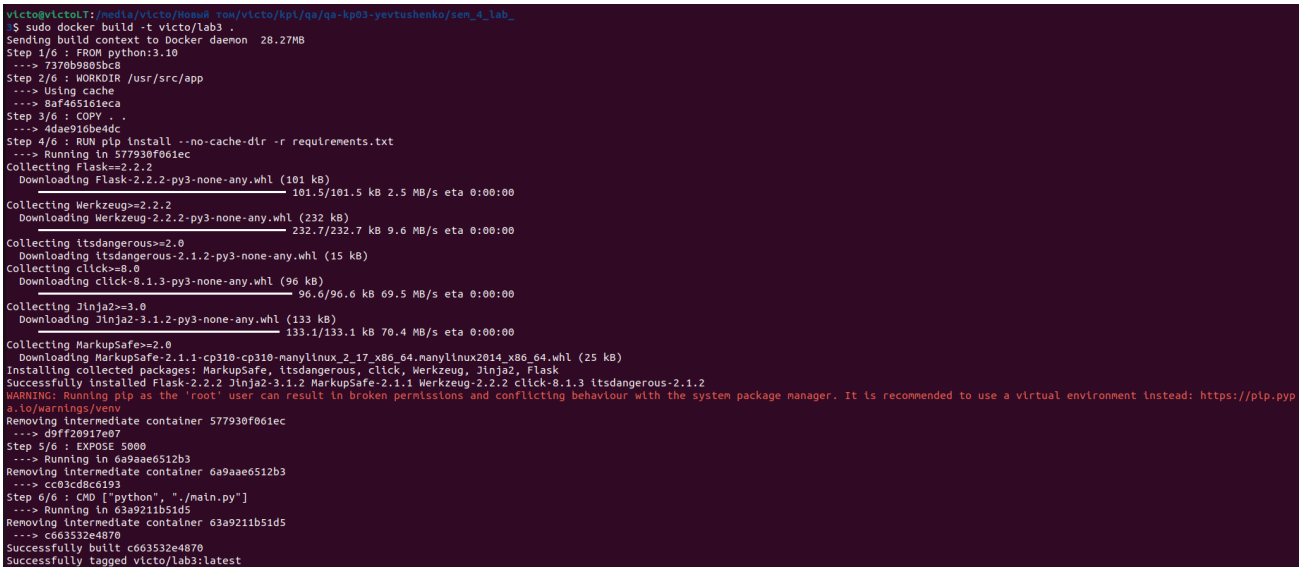
```
            self.directory.count -= 1
            index = self.directory.list.index(self)
            self.directory.list.pop(index)
            self.directory = location
            self.directory.list.append(self)
            self.directory.count += 1
        else:
            print("Directory is already full")
            return

    def read_file(self):
        print(self.contents)

    def append_line(self, line: str):
        self.contents += '\n' + line
```

bufferFile.py

```
class BufferFile:

    def __init__(self, name: str, directory, max_size: int):
        if directory.count < directory.max_size:
            self.directory = directory
            self.directory.count = 1
            self.directory.list.append(self)
            self.max_size = max_size
            self.name = name
            self.contents = []
        else:
            print("This directory is already full")
            return

    def delete(self):
        print("Deleting " + self.name)
        index = self.directory.list.index(self)
        self.directory.list.pop(index)
        del self
        print("Deleted successfully")

    def move_buffer_file(self, location):
        if location.count < location.max_size:
            self.directory.count -= 1
            index = self.directory.list.index(self)
            self.directory.list.pop(index)
            self.directory = location
            self.directory.list.append(self)
            self.directory.count += 1
        else:
            print("Directory is already full")
            return

    def push_element(self, element):
        if len(self.contents) < self.max_size:
            self.contents.append(element)
        else:
            print("File already full")

    def consume_element(self, element):
        index = self.contents.index(element)
        self.contents.pop(index)
```

Результати

Створення нового образу контейнера.



Запуск програми у контейнері.

Команди, які можна виконати над елементами файлової системи.

Висновки

У ході виконання даної курсової роботи був розроблений додаток, що дозволяє виконувати операції над елементами файлової системи, а також створено клієнт до нього. Була проведена робота із Docker, Python, Flask, Click та Pytest.