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"

Виконав: Ярмоленко Андрій Геннадійович
Група: КП-01

Перевірив: Погорєлов Володимир
Володимирович

_____

1 семестр 2022/2023

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

Узгоджено ЗАХИЩЕНА "__"_____2023р.

Керівник роботи з оцінкою_____

_____/Погорєлов.В.В./ _____/ Погорєлов.В.В./

Програмний додаток серверу що міститься у Docker-контейнері, та тести написані з використанням Robot framework

Виконавець роботи

Ярмоленко Андрій Геннадійович

_____2023р.

Код

app.py (server)

```
from flask import Flask, request, jsonify
from models.directory import directory
from models.binary_file import binary_file
from models.log_text_file import log_text_file
from models.buffer_file import buffer_file

app = Flask(__name__)

root = directory('root', 100)
deleted_list = []

#directory
@app.route('/directory', methods=['POST', 'GET', 'PATCH', 'DELETE'])
def directory():
   if request.method == 'POST':
      if any(x.name == request.args.get('name') for x in root.list) or
request.args.get('name') == 'root':
         return jsonify({
         "message": "Directory already exists.",
      }), 400
      dir = directory(request.args.get('name'),
request.args.get('max_elems'), root)
      return jsonify({
         "message": "Directory created successfully.",
         "directory": {
            "parent": str(dir.parent),
            "name": str(dir.name),
            "DIR_MAX_ELEMS": int(dir.DIR_MAX_ELEMS),
            "count_elems": int(dir.count_elems),
            "list": str(dir.list)
         }
      }), 201

   elif request.method == 'GET':
      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(x for x in root.list if x.name ==
request.args.get('name'))
         return jsonify({
         "message": "Directory was read successfully.",
         "directory": {
            "parent": str(dir.parent),
```

```
            "name": str(dir.name),
            "DIR_MAX_ELEMS": int(dir.DIR_MAX_ELEMS),
            "count_elems": int(dir.count_elems),
            "list": str(dir.list)
         }
      }), 200
      return jsonify({
         "message": "Directory doesn't exist.",
         }), 400

   elif request.method == 'PATCH':
      if 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.move(root)
         return jsonify({
         "message": "Directory moved successfully.",
         "directory": {
            "parent": str(dir.parent.name),
            "name": str(dir.name),
            "DIR_MAX_ELEMS": int(dir.DIR_MAX_ELEMS),
            "count_elems": int(dir.count_elems),
            "list": str(dir.list)
         }
      }), 200
      return jsonify({
         "message": "Directory doesn't exist.",
         }), 400

   else:
      if request.args.get('name') not in deleted_list 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'))
         del dir
         deleted_list.append(request.args.get('name'))
         return jsonify({
         "message": "Directory deleted successfully.",
         }), 200
      return jsonify({
         "message": "Directory was not deleted.",
         }), 400


@app.route('/binaryfile', methods=['POST', 'GET', 'PATCH', 'DELETE'])
def binaryfile():
   if request.method == 'POST':
      if any(x.name == request.args.get('name') for x in root.list):
         return jsonify({
```

```
            "message": "File already exists.",
        }), 400
        file = binary_file(root, request.args.get('name'),
request.args.get('info'))
        return jsonify({
            "message": "File created successfully.",
            "file": {
                "parent": str(file.parent.name),
                "name": str(file.name),
                "info": str(file.info)
            }
        }), 201

    elif request.method == 'GET':
        if 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'))
            return jsonify({
            "message": "File was read successfully.",
            "file": {
                "parent": str(file.parent.name),
                "name": str(file.name),
                "info": str(file.info)
            }
        }), 200
        return jsonify({
            "message": "File doesn't exist.",
            }), 400

    elif request.method == 'PATCH':
        if 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.move(root)
            return jsonify({
            "message": "File moved successfully.",
            "file": {
                "parent": str(file.parent.name),
                "name": str(file.name),
                "info": str(file.info)
            }
        }), 200
        return jsonify({
            "message": "File doesn't exist.",
            }), 400

    else:
```

```
      if request.args.get('name') not in deleted_list 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'))
         del file
         deleted_list.append(request.args.get('name'))
         return jsonify({
         "message": "File deleted successfully.",
         }), 200
      return jsonify({
         "message": "File was not deleted.",
         }), 400


@app.route('/logtextfile', methods=['POST', 'GET', 'PATCH', 'DELETE'])
def logtextfile():
   if request.method == 'POST':
      if any(x.name == request.args.get('name') for x in root.list):
         return jsonify({
         "message": "File already exists.",
      }), 400
      file = log_text_file(root, request.args.get('name'),
request.args.get('info'))
      return jsonify({
         "message": "File created successfully.",
         "file": {
            "parent": str(file.parent.name),
            "name": str(file.name),
            "info": str(file.info)
         }
      }), 201

   elif request.method == 'GET':
      if 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'))
         return jsonify({
         "message": "File was read successfully.",
         "file": {
            "parent": str(file.parent.name),
            "name": str(file.name),
            "info": str(file.info)
         }
      }), 200
      return jsonify({
         "message": "File doesn't exist.",
         }), 400

   elif request.method == 'PATCH':
```

```
        if 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'))
            if request.args.get('parent'):
                file.move(root)
                return jsonify({
                "message": "File moved successfully.",
                "file": {
                    "parent": str(file.parent.name),
                    "name": str(file.name),
                    "info": str(file.info)
                }
            }), 200
            if request.args.get('append'):
                file.append_line(request.args.get('append'))
                return jsonify({
                "message": "Line added successfully.",
                "file": {
                    "parent": str(file.parent.name),
                    "name": str(file.name),
                    "info": str(file.info)
                }
            }), 201
            return jsonify({
            "message": "Bad request.",
            }), 400
        return jsonify({
            "message": "File doesn't exist.",
            }), 400

    else:
        if request.args.get('name') not in deleted_list 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'))
            del file
            deleted_list.append(request.args.get('name'))
            return jsonify({
            "message": "File deleted successfully.",
            }), 200
        return jsonify({
            "message": "File was not deleted.",
            }), 400


@app.route('/bufferfile', methods=['POST', 'GET', 'PATCH', 'DELETE'])
def bufferfile():
    if request.method == 'POST':
        if any(x.name == request.args.get('name') for x in root.list):
```

```
            return jsonify({
            "message": "File already exists.",
        }), 400
        file = buffer_file(root, request.args.get('max_size'),
request.args.get('name'))
        return jsonify({
            "message": "File created successfully.",
            "file": {
                "parent": str(file.parent),
                "name": str(file.name),
                "MAX_BUF_FILE_SIZE": int(file.MAX_BUF_FILE_SIZE),
                "info": list(file.info )
            }
        }), 201

    elif request.method == 'GET':
        if 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'))
            return jsonify({
            "message": "File was read successfully.",
            "file": {
                "parent": str(file.parent),
                "name": str(file.name),
                "MAX_BUF_FILE_SIZE": int(file.MAX_BUF_FILE_SIZE),
                "info": list(file.info )
            }
        }), 200
        return jsonify({
            "message": "File doesn't exist.",
            }), 400

    elif request.method == 'PATCH':
        if 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'))
            if request.args.get('parent'):
                file.move(root)
                return jsonify({
                "message": "File moved successfully.",
                "file": {
                    "parent": str(file.parent.name),
                    "name": str(file.name),
                    "MAX_BUF_FILE_SIZE": int(file.MAX_BUF_FILE_SIZE),
                    "info": list(file.info)
                }
            }), 200
            if request.args.get('append'):
```

```
            file.push(request.args.get('append'))
            return jsonify({
            "message": "Line added successfully.",
            "file": {
               "parent": str(file.parent.name),
               "name": str(file.name),
               "MAX_BUF_FILE_SIZE": int(file.MAX_BUF_FILE_SIZE),
               "info": list(file.info)
            }
         }), 201
         if request.args.get('consume'):
            if len(file.info) > 0:
               file.consume()
            return jsonify({
            "message": "Line consumed successfully.",
            "file": {
               "parent": str(file.parent.name),
               "name": str(file.name),
               "MAX_BUF_FILE_SIZE": int(file.MAX_BUF_FILE_SIZE),
               "info": list(file.info)
            }
         }), 200
         return jsonify({
         "message": "Bad request.",
         }), 400
      return jsonify({
         "message": "File doesn't exist.",
         }), 400

   else:
      if request.args.get('name') not in deleted_list 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'))
         del file
         deleted_list.append(request.args.get('name'))
         return jsonify({
         "message": "File deleted successfully.",
         }), 200
      return jsonify({
         "message": "File was not deleted.",
         }), 400


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Binary_file.py

```python
class binary_file:
    def __init__(self, parent, name, info):
        if (parent.count_elems >= parent.DIR_MAX_ELEMS ):
            print('Parent directory is full.')
            return
        self.parent = parent
        self.parent.count_elems += 1
        self.name = name
        self.info = info
        self.parent.list.append(self)

    def __delete__(self, instance):
        print('Directory was deleted.')
        return

    def move(self, location):
        if (location.count_elems >= location.DIR_MAX_ELEMS):
            print('Directory is full. Can\'t move.')
            return
        self.parent.count_elems -=1
        index = self.parent.list.index(self)
        self.parent.list.pop(index)
        self.parent = location
        self.parent.list.append(self)
        self.parent.count_elems +=1

    def read(self):
        return self.info
```

Buffer_file.py

```python
from itertools import count


class buffer_file:
    def __init__(self, parent, max_size, name):
        if (parent.count_elems >= parent.DIR_MAX_ELEMS ):
            print('Parent directory is full')
            return
        self.parent = parent
        self.parent.count_elems += 1
        self.parent.list.append(self)
        self.MAX_BUF_FILE_SIZE = max_size
        self.name = name
        self.info = []

    def __delete__(self, instance):
        print('Directory was deleted.')
```

```
        return

    def move(self, location):
        if (location.count_elems >= location.DIR_MAX_ELEMS):
            return
        self.parent.count_elems -=1
        index = self.parent.list.index(self)
        self.parent.list.pop(index)
        self.parent = location
        self.parent.list.append(self)
        self.parent.count_elems +=1

    def push(self, elem):
        if len(self.info) >= int(self.MAX_BUF_FILE_SIZE):
            print('File is full. Can\'t push new line.')
            return
        self.info.append(elem)

    def consume(self):
        self.info.pop()
```

Directory.py

```
class directory:
    def __init__(self, name, max_elems, parent = None):
        if parent != None:
            if (parent.count_elems >= parent.DIR_MAX_ELEMS ):
                print('Parent directory is full.')
                return
            parent.count_elems += 1
            parent.list.append(self)
        self.parent = parent
        self.name = name
        self.DIR_MAX_ELEMS = max_elems
        self.count_elems = 0
        self.list = []

    def __delete__(self, instance):
        print('Directory was deleted.')
        return

    def list_elems(self):
        res = self.name + ': ( '
        for item in self.list:
            if type(item) is directory:
                res += item.list_elems()
            else:
                res += item.name
                res += ', '
```

```
        res += '), '
        return res

    def move(self, location):
        if (location.count_elems >= location.DIR_MAX_ELEMS +
self.count_elems + 1):
            print('Directory is full. Can\'t move.')
            return
        if self.parent != None:
            self.parent.count_elems -= self.count_elems + 1
            index = self.parent.list.index(self)
            self.parent.list.pop(index)
        self.parent = location
        self.parent.list.append(self)
        self.parent.count_elems += self.count_elems + 1
```

Log_text_file.py

```
class log_text_file:
    def __init__(self, parent, name, info):
        if (parent.count_elems >= parent.DIR_MAX_ELEMS ):
            print('Parent directory is full.')
            return
        self.parent = parent
        self.name = name
        self.info = info
        self.parent.count_elems += 1
        self.parent.list.append(self)

    def __delete__(self, instance):
        print('Directory was deleted.')
        return

    def move(self, location):
        if (location.count_elems >= location.DIR_MAX_ELEMS):
            print('Directory is full. Can\'t move.')
            return
        self.parent.count_elems -=1
        index = self.parent.list.index(self)
        self.parent.list.pop(index)
        self.parent = location
        self.parent.list.append(self)
        self.parent.count_elems +=1

    def read(self):
        return self.info

    def append_line(self, line):
```

```
        self.info += '\n'
        self.info += line
```

CLI.py

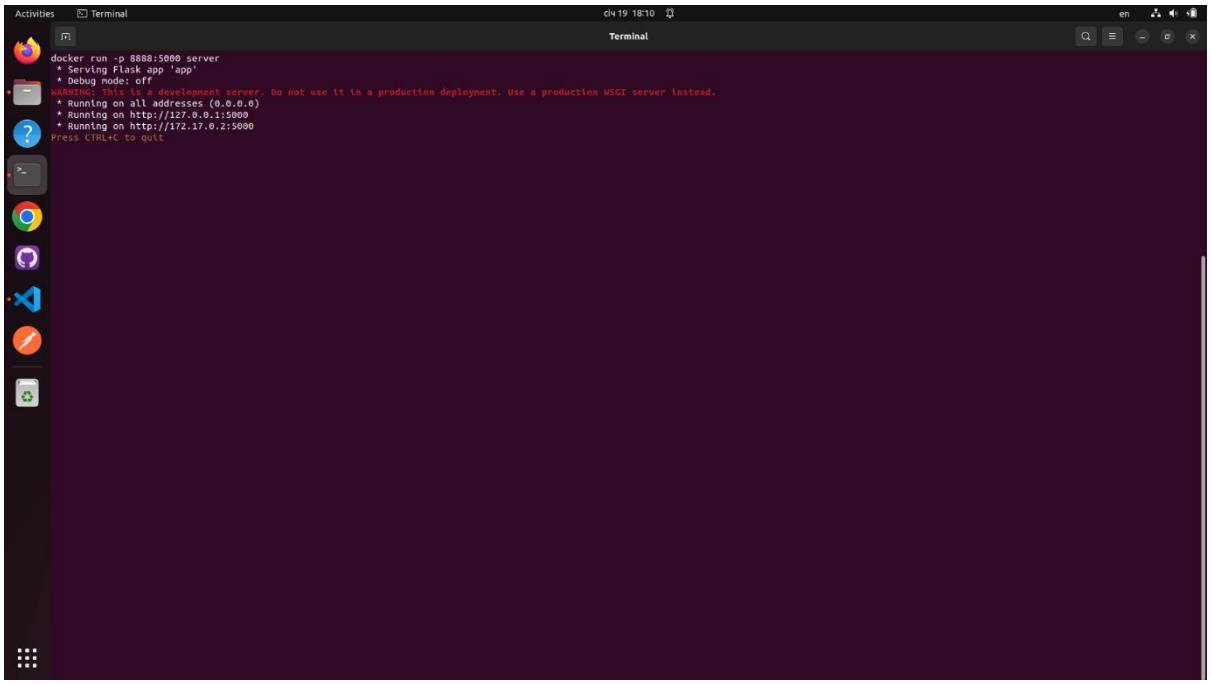
```
import sys
import pip._vendor.requests as requests

n = len(sys.argv)
Request = "http://localhost:8888/" + sys.argv[2]

response = ""
if sys.argv[1] == "get":
    response = requests.get(Request)
if sys.argv[1] == "post":
    response = requests.post(Request)
if sys.argv[1] == "patch":
    response = requests.patch(Request)
if sys.argv[1] == "delete":
    response = requests.delete(Request)

print("Response: ", response.status_code + response.json())
```

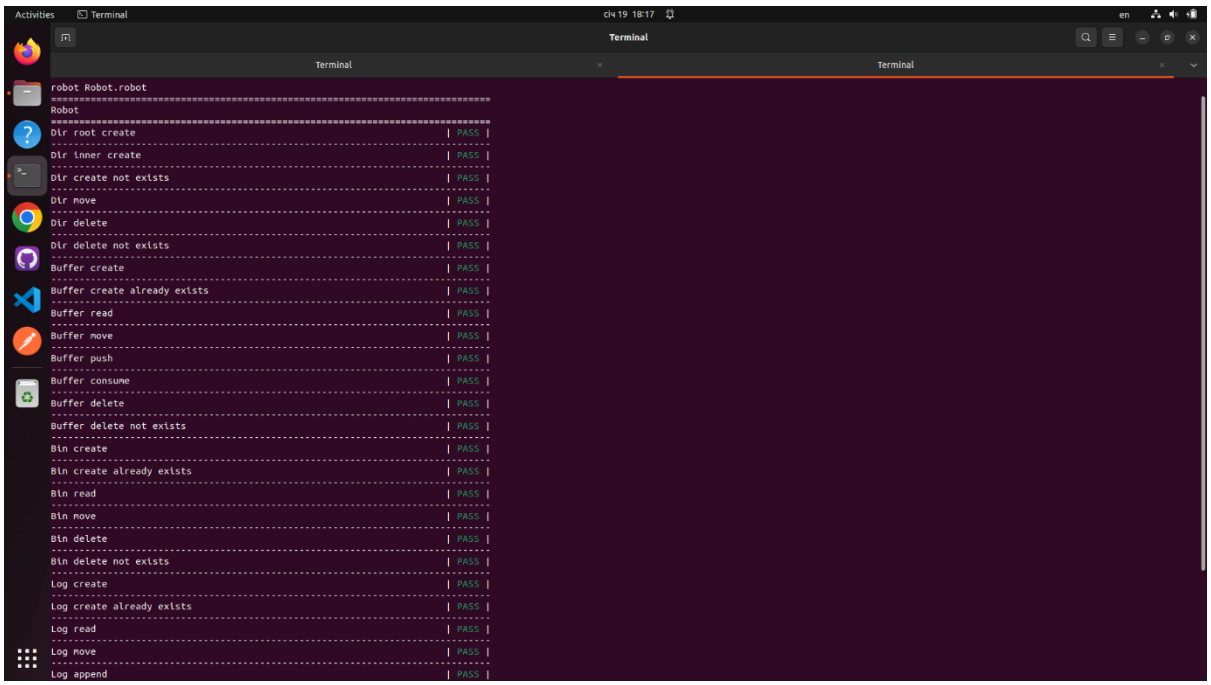
Robot.robot

```
*** Settings ***
Library    Process
Library    OperatingSystem



*** Variables ***
${cli}    /home/andrii/Documents/GitHub/qa-kp01-
Yarmolenko/coursework/CLI.py



*** Test Cases ***
Dir root create
    ${result} =    Run Process    python3    ${cli}    get
directory?name\=root
    Should Contain    ${result.stdout}    Status code: 200

Dir inner create
    ${result} =    Run Process    python3    ${cli}    post
directory?parent\=root&name\=child&max_elems\=8
    Should Contain    ${result.stdout}    Status code: 201

Dir move
    ${result} =    Run Process    python3    ${cli}    patch
directory?name\=child&parent\=root
    Should Contain    ${result.stdout}    Status code: 200

Dir delete
```

```
    ${result} =    Run Process    python3   ${cli}    delete
directory?name\=child
    Should Contain    ${result.stdout}   Status code: 200



Buffer create
    ${result} =    Run Process    python3   ${cli}    post
bufferfile?parent\=root&max_size\=100&name\=bufferfile
    Should Contain    ${result.stdout}   Status code: 201

Buffer read
    ${result} =    Run Process    python3   ${cli}    get
bufferfile?name\=bufferfile
    Should Contain    ${result.stdout}   Status code: 200

Buffer move
    ${result} =    Run Process    python3   ${cli}    patch
bufferfile?parent\=root&name\=bufferfile
    Should Contain    ${result.stdout}   Status code: 200

Buffer delete
    ${result} =    Run Process    python3   ${cli}    delete
bufferfile?name\=bufferfile
    Should Contain    ${result.stdout}   Status code: 200



Bin create
    ${result} =    Run Process    python3   ${cli}    post
binaryfile?name\=binaryfile&parent\=root&info\=test
    Should Contain    ${result.stdout}   Status code: 201

Bin read
    ${result} =    Run Process    python3   ${cli}    get
binaryfile?name\=binaryfile
    Should Contain    ${result.stdout}   Status code: 200

Bin move
    ${result} =    Run Process    python3   ${cli}    patch
binaryfile?name\=binaryfile&parent\=root
    Should Contain    ${result.stdout}   Status code: 200

Bin delete
    ${result} =    Run Process    python3   ${cli}    delete
binaryfile?name\=binaryfile
    Should Contain    ${result.stdout}   Status code: 200
```

Фото роботи :


Activities
Terminal
Терминал 19 18:10
en
Terminal
docker run -p 8888:5000 server
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit


Activities
Terminal
Терминал 19 18:17
en
Terminal
Terminal
Terminal
robot Robot.robot
==========
Robot
==========
Dir root create | PASS |
Dir inner create | PASS |
Dir create not exists | PASS |
Dir move | PASS |
Dir delete | PASS |
Dir delete not exists | PASS |
Buffer create | PASS |
Buffer create already exists | PASS |
Buffer read | PASS |
Buffer move | PASS |
Buffer push | PASS |
Buffer consume | PASS |
Buffer delete | PASS |
Buffer delete not exists | PASS |
Bin create | PASS |
Bin create already exists | PASS |
Bin read | PASS |
Bin move | PASS |
Bin delete | PASS |
Bin delete not exists | PASS |
Log create | PASS |
Log create already exists | PASS |
Log read | PASS |
Log move | PASS |
Log append | PASS |

```
docker run -p 8888:5000 server
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [19/Jan/2023 16:16:40] "GET /directory?name=root HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:40] "POST /directory?parent=root&name=child&max_elems=8 HTTP/1.1" 201 -
172.17.0.1 - - [19/Jan/2023 16:16:40] "POST /directory?parent=root&name=child&max_elems=8 HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:41] "PATCH /directory?name=child&parent=root HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:41] "DELETE /directory?name=child HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:41] "DELETE /directory?name=child HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:41] "POST /bufferfile?parent=root&max_size=100&name=bufferfile HTTP/1.1" 201 -
172.17.0.1 - - [19/Jan/2023 16:16:42] "POST /bufferfile?parent=root&max_size=100&name=bufferfile HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:42] "GET /bufferfile?name=bufferfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:42] "PATCH /bufferfile?parent=root&name=bufferfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:42] "PATCH /bufferfile?name=bufferfile&append=test_text HTTP/1.1" 201 -
172.17.0.1 - - [19/Jan/2023 16:16:42] "PATCH /bufferfile?name=bufferfile&consume=true HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:43] "DELETE /bufferfile?name=bufferfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:43] "DELETE /bufferfile?name=bufferfile HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:43] "POST /binaryfile?name=blnaryfile&parent=root&info=test HTTP/1.1" 201 -
172.17.0.1 - - [19/Jan/2023 16:16:43] "POST /binaryfile?name=binaryfile&parent=root&info=test HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:44] "GET /binaryfile?name=binaryfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:44] "PATCH /binaryfile?name=binaryfile&parent=root HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:44] "DELETE /binaryfile?name=binaryfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:44] "DELETE /binaryfile?name=binaryfile HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:44] "POST /logtextfile?name=logtextfile&parent=root&info=test HTTP/1.1" 201 -
172.17.0.1 - - [19/Jan/2023 16:16:45] "POST /logtextfile?name=logtextfile&parent=root&info=test HTTP/1.1" 400 -
172.17.0.1 - - [19/Jan/2023 16:16:45] "GET /logtextfile?name=logtextfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:45] "PATCH /logtextfile?name=logtextfile&parent=root HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:45] "PATCH /logtextfile?name=logtextfile&append=test1 HTTP/1.1" 201 -
172.17.0.1 - - [19/Jan/2023 16:16:46] "DELETE /logtextfile?name=logtextfile HTTP/1.1" 200 -
172.17.0.1 - - [19/Jan/2023 16:16:46] "DELETE /logtextfile?name=logtextfile HTTP/1.1" 400 -
```

Висновки

Метою даного курсового проекту було розроблення серверу на мові pyhton, розташування його у Docker контейнері, розроблення інтерфейсу командного рядка для створення запитів до серверу та тестування цього інтерфейсу за допомогою Robot framework.