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  Керівник роботи    \_\_\_\_\_\_\_\_\_\_/Погорєлов.В.В./ | ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_\_\_\_2023р.  з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_/ 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 |
| 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 |
| 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)

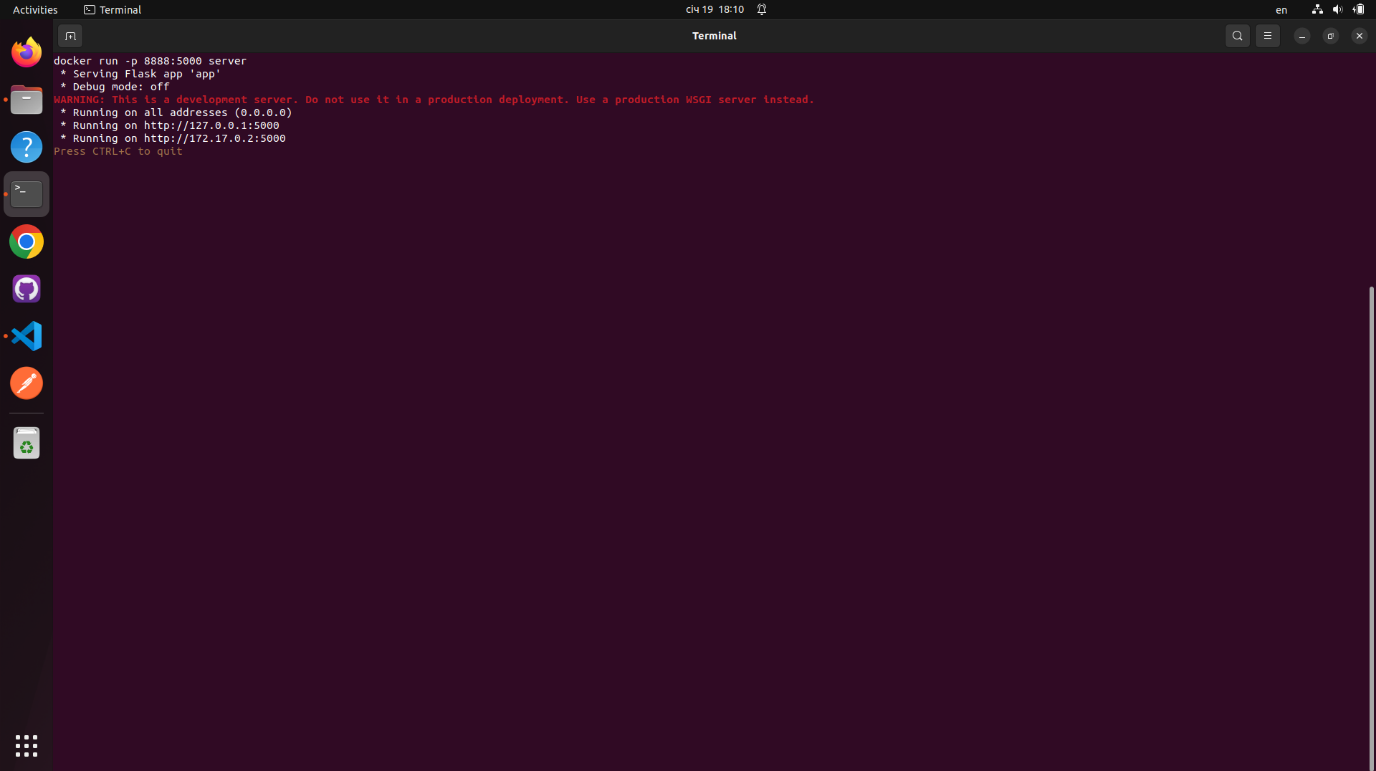
if sys.argv[1] == "delete":

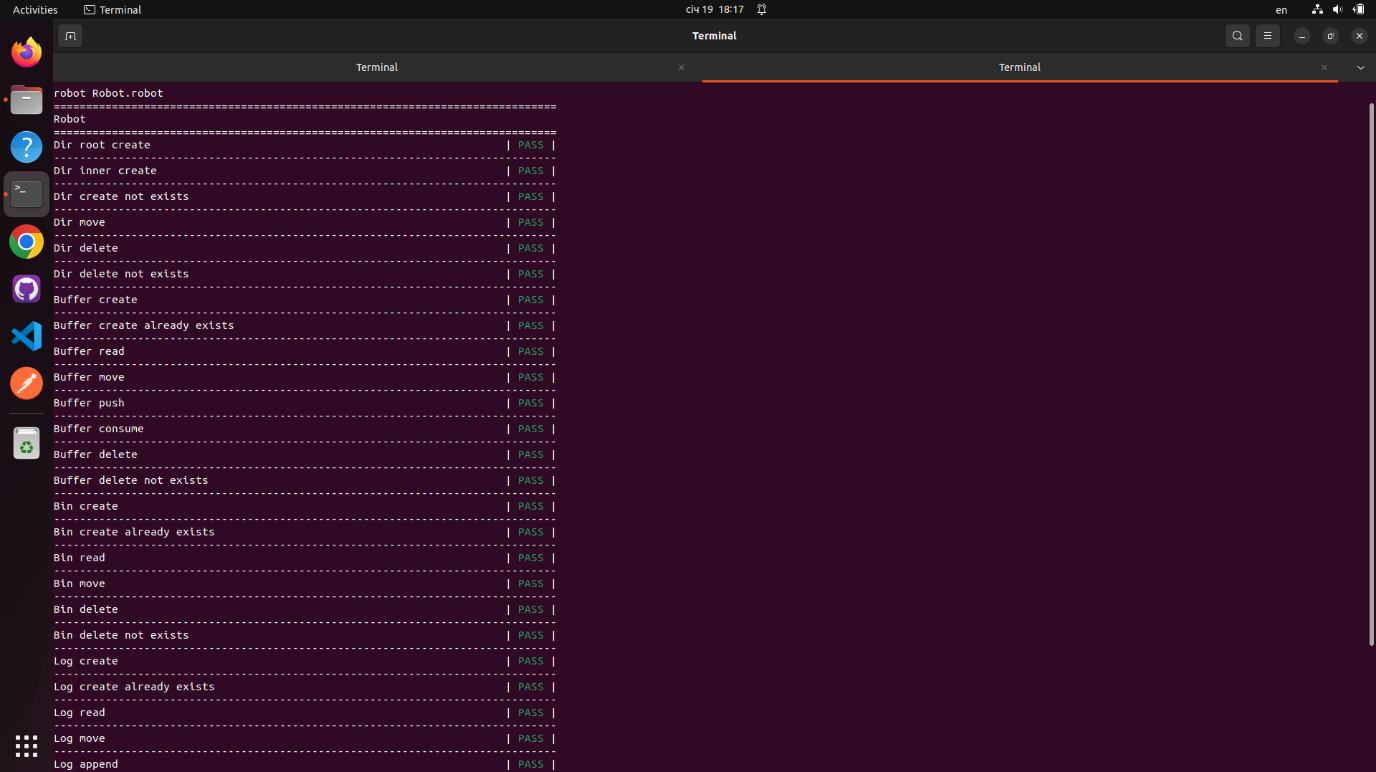
    response = requests.delete(Request)

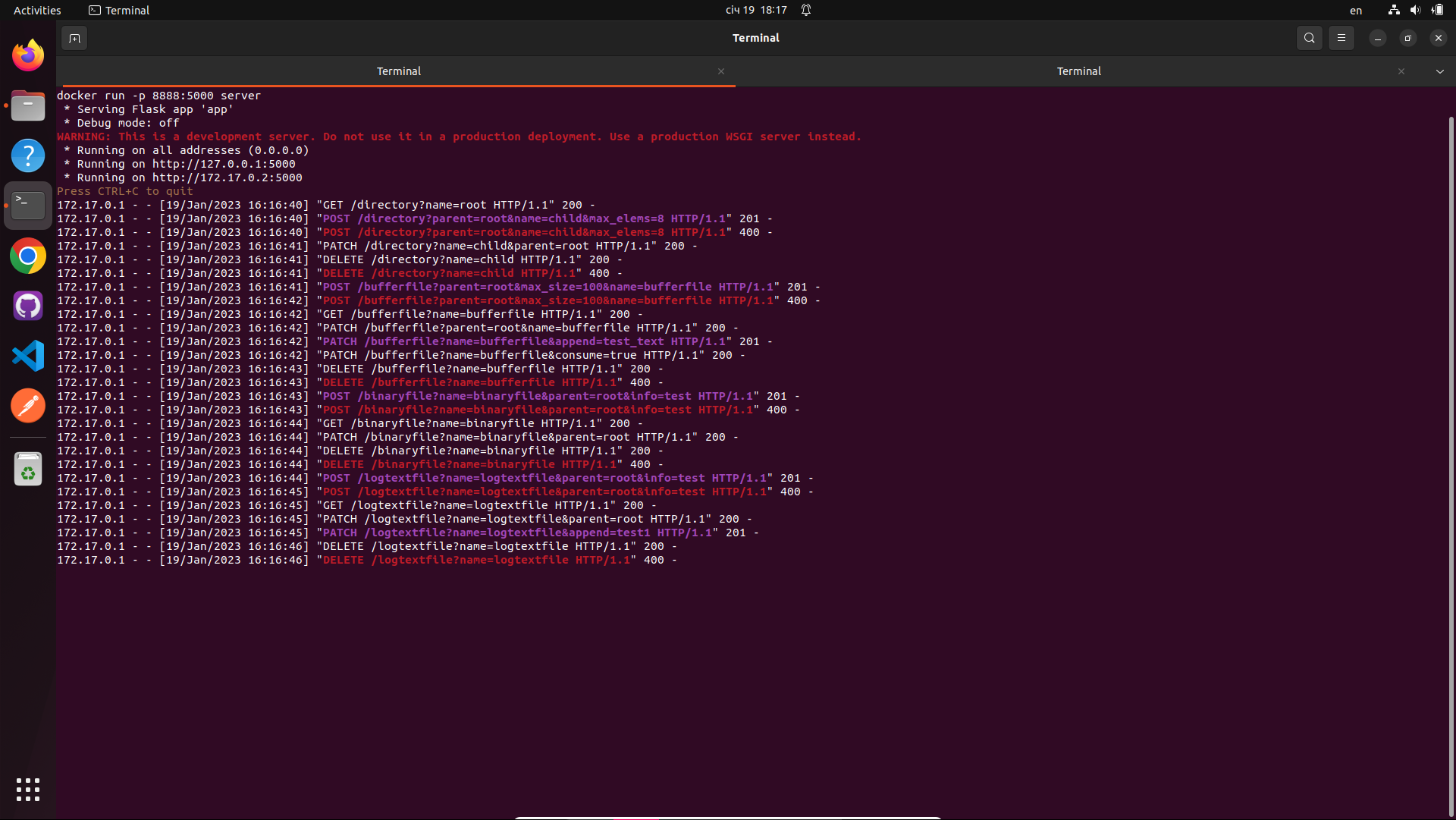
print("Response: ", response.status\_code + response.json())

|  |
| --- |
| 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 :







# 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.