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"***

на тему

**Файлова система. Клієнт-серверний додаток та тестування**

                                                         Виконав: Горобець Микола Романович

Група: КП-02

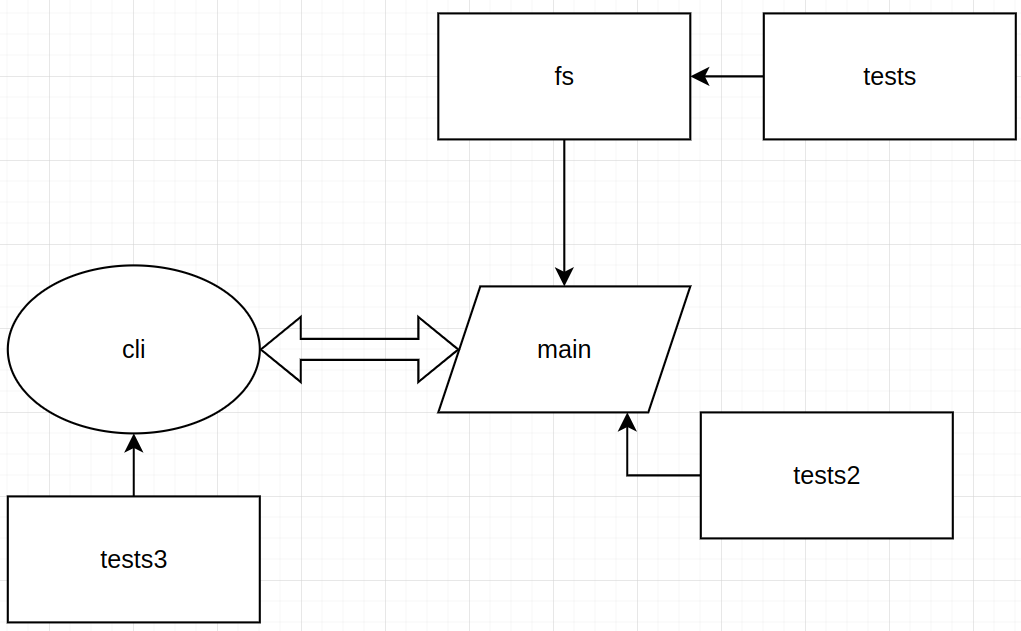
|  |  |
| --- | --- |
| Узгоджено  Керівник роботи  \_\_\_\_\_\_\_\_\_\_/Погорєлов В./ | ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_\_\_\_2023р.  з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_/Погорєлов В./ |

1 семестр 2022/2023

**ВСТУП**

Курсова робота присвячена розробці програмного забезпечення для симуляції файлової системи, створення REST-сервера та його перевірці. Завдання демонстрації створення та імітації файлової системи має велику актуальність сьогодні, оскільки всі користувачі комп'ютерів використовують файлові системи.

**Модульна організація та функціональні можливості**



Модуль **main** реалізує головні функції програми. Модуль **cli** напряму працює з користувачем за допомогою CLI інтерфейсу через командну строку. Модуль **fs** зберігає у собі усі класи і методи, необхідні для функціонування файлової системи. В модулях tests, tests2 та tests3 містяться усі тести інших модулів.

**Аналіз мов програмування та технологій розроблення**

- Мова програмування Python

* модуль **Queue** для зручності роботи з буферними файлами
* Модуль **Json** для роботи з python об’єктами JSON-формату
* Модуль **httpx** для обробки даних, отриманих через HTTP протокол

- Фреймворк Robot для тестування модулів

* *Бібліотека* ***JSONLibrary*** для роботи з JSON-форматом даних
* *Бібліотека* ***Process*** для виконання команд мовою python

- Фреймворк Flask для розробки веб-додатків

* Модуль **jsonify** для конвертації даних файлів у JSON-формат перед відправленнямклієнту
* Модуль **request** для отримання даних з HTTP-запитів, надісланих клієнтом

- Бібліотека Typer для створення інтерфейсу користувача

**ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ**

**Модуль сервера**

|  |
| --- |
| **main.py** |
| from flask import Flask, jsonify, requestfrom fs import FileSystem, Directory, BinaryFile, BufferFile, LogFile, get\_items, get\_file\_dataapp = Flask(\_\_name\_\_)fs = FileSystem()fs.addItem(Directory('sub1'), 'root')fs.addItem(Directory('sub2'), 'root')b = BufferFile('buff')b.push('1')b.push('2')b.push('3')fs.addItem(BinaryFile('binFile', "bibin"), 'root')fs.addItem(b, 'root')@app.get("/directory/<path:directory\_path>")def get\_directory(directory\_path): directory = fs.findDir(directory\_path) if directory is None: return {'error\_message': 'Directory not found'}, 400 items = directory.items() return get\_items(items)@app.get("/bufferfile/<path:file\_path>")def get\_buffer\_file(file\_path): buffer\_file = fs.findFile(file\_path) if buffer\_file is None or type(buffer\_file) is not BufferFile: return {'error\_message': 'File not found'}, 400 return jsonify(get\_file\_data(buffer\_file))@app.get("/logfile/<path:file\_path>")def get\_log\_file(file\_path): log\_file = fs.findFile(file\_path) if log\_file is None or type(log\_file) is not LogFile: return {'error\_message': 'File not found'}, 400 return jsonify(get\_file\_data(log\_file))@app.get("/binaryfile/<path:file\_path>")def get\_binary\_file(file\_path): binary\_file = fs.findFile(file\_path) if binary\_file is None or type(binary\_file) is not BinaryFile: return {'error\_message': 'File not found'}, 400 return jsonify(get\_file\_data(binary\_file))@app.delete("/directory/<path:directory\_path>")def delete\_directory(directory\_path): directory = fs.findDir(directory\_path) if directory is None or type(directory) is not Directory: return {'error\_message': 'Directory not found'}, 400 is\_removed = fs.removeItem(directory\_path) if is\_removed: return {'success': True} else: return {'error\_message': 'Directory could not be removed'}, 400@app.delete("/binaryfile/<path:file\_path>")def delete\_binaryfile(file\_path): bin\_file = fs.findFile(file\_path) if bin\_file is None or type(bin\_file) is not BinaryFile: return {'error\_message': 'BinaryFile not found'}, 400 fs.removeItem(file\_path) return {'success': True}@app.delete("/logfile/<path:file\_path>")def delete\_logfile(file\_path): log\_file = fs.findFile(file\_path) if log\_file is None or type(log\_file) is not LogFile: return {'error\_message': 'LogFile not found'}, 400 fs.removeItem(file\_path) return {'success': True}@app.delete("/bufferfile/<path:file\_path>")def delete\_bufferfile(file\_path): buffer\_file = fs.findFile(file\_path) if buffer\_file is None or type(buffer\_file) is not BufferFile: return {'error\_message': 'BufferFile not found'}, 400 fs.removeItem(file\_path) return {'success': True}@app.post("/directory/")def create\_directory(): json = request.get\_json() directory\_path = json.get('directory\_path') directory\_name = json.get('directory\_name') new\_directory = Directory(directory\_name) is\_created = fs.addItem(new\_directory, directory\_path) if is\_created: return {'success': True} else: return {'error\_message': 'Directory creation failed'}, 400@app.post("/bufferfile/")def create\_bufferfile(): json = request.get\_json() directory\_path = json.get('directory\_path') buffer\_file\_name = json.get('file\_name') new\_file = BufferFile(buffer\_file\_name) is\_created = fs.addItem(new\_file, directory\_path) if is\_created: return {'success': True} else: return {'error\_message': 'File creation failed'}, 400@app.post("/logfile/")def create\_logfile(): json = request.get\_json() directory\_path = json.get('directory\_path') binary\_file\_name = json.get('file\_name') new\_file = LogFile(binary\_file\_name, "") is\_created = fs.addItem(new\_file, directory\_path) if is\_created: return {'success': True} else: return {'error\_message': 'File creation failed'}, 400@app.post("/binaryfile/")def create\_binaryfile(): json = request.get\_json() directory\_path = json.get('directory\_path') binary\_file\_name = json.get('file\_name') file\_content = json.get('content') new\_file = BinaryFile(binary\_file\_name, file\_content) is\_created = fs.addItem(new\_file, directory\_path) if is\_created: return {'success': True} else: return {'error\_message': 'File creation failed'}, 400@app.post("/logfile/log")def log\_line\_to\_file(): json = request.get\_json() log\_file\_path = json.get('log\_file\_path') log\_line = json.get('log\_line') if log\_line is None or log\_file\_path is None: return {'error\_message': 'Invalid data'}, 400 log\_file = fs.findFile(log\_file\_path) if log\_file is None: return {'error\_message': 'File not found'}, 400 if type(log\_file) is not LogFile: return {'error\_message': 'File do not supports logging'}, 400 log\_file.append(log\_line) return {'success': True}@app.post("/bufferfile/push")def push\_element\_to\_buffer\_file(): json = request.get\_json() buffer\_file\_path = json.get('buffer\_file\_path') element = json.get('push\_element') if buffer\_file\_path is None or element is None: return {'error\_message': 'Invalid data'}, 400 buffer\_file = fs.findFile(buffer\_file\_path) if buffer\_file is None: return {'error\_message': 'File not found'}, 400 if type(buffer\_file) is not BufferFile: return {'error\_message': 'This operation can not be done with this file'}, 400 is\_pushed = buffer\_file.push(element) if is\_pushed: return {'success': True} else: return {'error\_message': 'Element can not be pushed'}, 400@app.get("/bufferfile/pop/<path:buffer\_file\_path>")def pop\_element\_from\_buffer\_file(buffer\_file\_path): buffer\_file = fs.findFile(buffer\_file\_path) if buffer\_file is None: return {'error\_message': 'File not found'}, 400 if type(buffer\_file) is not BufferFile: return {'error\_message': 'This operation can not be done with this file'}, 400 pop\_element = buffer\_file.pop() if pop\_element is not None: return {'pop\_element': pop\_element} else: return {'error\_message': 'Buffer file is empty'}, 400@app.post("/bufferfile/move")def move\_bufferfile(): json = request.get\_json() buffer\_file\_path = json.get('buffer\_file\_path') destination\_path = json.get('destination\_path') buffer\_file = fs.findFile(buffer\_file\_path) if buffer\_file is None or type(buffer\_file) is not BufferFile: return {'error\_message': 'File not found'}, 400 is\_moved = fs.moveItem(buffer\_file\_path, destination\_path) if is\_moved: return {'success': True} else: return {'error\_message': f'File can not be moved to {destination\_path}'}, 400@app.post("/directory/move")def move\_directory(): json = request.get\_json() directory\_path = json.get('directory\_path') destination\_path = json.get('destination\_path') directory = fs.findDir(directory\_path) if directory is None or type(directory) is not Directory: return {'error\_message': 'Directory not found'}, 400 is\_moved = fs.moveItem(directory\_path, destination\_path) if is\_moved: return {'success': True} else: return {'error\_message': f'Directory can not be moved to {destination\_path}'}, 400@app.post("/logfile/move")def move\_logfile(): json = request.get\_json() log\_file\_path = json.get('log\_file\_path') destination\_path = json.get('destination\_path') log\_file = fs.findFile(log\_file\_path) if log\_file is None or type(log\_file) is not LogFile: return {'error\_message': 'File not found'}, 400 is\_moved = fs.moveItem(log\_file\_path, destination\_path) if is\_moved: return {'success': True} else: return {'error\_message': f'File can not be moved to {destination\_path}'}, 400@app.post("/binaryfile/move")def move\_binaryfile(): json = request.get\_json() binary\_file\_path = json.get('binary\_file\_path') destination\_path = json.get('destination\_path') binary\_file = fs.findFile(binary\_file\_path) if binary\_file is None or type(binary\_file) is not BinaryFile: return {'error\_message': 'File not found'}, 400 is\_moved = fs.moveItem(binary\_file\_path, destination\_path) if is\_moved: return {'success': True} else: return {'error\_message': f'File can not be moved to {destination\_path}'}, 400if \_\_name\_\_ == '\_\_main\_\_': app.run() |

Модуль призначений для роботи сервера

Ось функціонал, який надає цей модуль:

1. Отримання списку елементів у каталозі.
2. Отримання даних файлів BufferFile, LogFile та BinaryFile.
3. Видалення каталогів, BufferFile, LogFile та BinaryFile.
4. Створення каталогів, BufferFile, LogFile та BinaryFile.
5. Запис рядка у LogFile.
6. Додавання (push) та вилучення (pop) елементів у BufferFile.
7. Переміщення каталогів, BufferFile, LogFile та BinaryFile.

**Модуль клієнта**

|  |
| --- |
| **Cli.py** |
| import jsonimport typerimport httpximport tracebackapp = typer.Typer()headers = {'Content-Type': 'application/json'}base\_url = 'http://0.0.0.0:5000/'directory\_catalog = 'directory/'logfile\_catalog = 'logfile/'binaryfile\_catalog = 'binaryfile/'bufferfile\_catalog = 'bufferfile/'version\_catalog = 'version'move\_subcatalog = 'move'@app.command()def dir\_get(path: str): try: res = httpx.get(base\_url + directory\_catalog + path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def dir\_create(path: str, name: str): try: data = {'directory\_path': path, 'directory\_name': name} res = httpx.post(base\_url + directory\_catalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception as e: print(json.dumps({"error\_message": str(e), "traceback": traceback.format\_exc()}))@app.command()def dir\_delete(directory\_path: str): try: res = httpx.delete(base\_url + directory\_catalog + directory\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def dir\_move(directory\_path: str, destination\_path: str): try: data = {'directory\_path': directory\_path, 'destination\_path': destination\_path} res = httpx.post(base\_url + directory\_catalog + move\_subcatalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def log\_create(path: str, file\_name: str): try: data = {'directory\_path': path, 'file\_name': file\_name} res = httpx.post(base\_url + logfile\_catalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def log\_log(file\_path: str, log\_line: str): try: data = {'log\_file\_path': file\_path, 'log\_line': log\_line} res = httpx.post(base\_url + logfile\_catalog + 'log', json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def log\_get(file\_path: str): try: res = httpx.get(base\_url + logfile\_catalog + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def log\_move(file\_path: str, destination\_path: str): try: data = {'log\_file\_path': file\_path, 'destination\_path': destination\_path} res = httpx.post(base\_url + logfile\_catalog + move\_subcatalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def log\_delete(file\_path: str): try: res = httpx.delete(base\_url + logfile\_catalog + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def bin\_create(path: str, file\_name: str, content: str): try: data = {'directory\_path': path, 'file\_name': file\_name, 'content': content} res = httpx.post(base\_url + binaryfile\_catalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def bin\_get(file\_path: str): try: res = httpx.get(base\_url + binaryfile\_catalog + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def bin\_delete(file\_path: str): try: res = httpx.delete(base\_url + binaryfile\_catalog + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def bin\_move(file\_path: str, destination\_path: str): try: data = {'binary\_file\_path': file\_path, 'destination\_path': destination\_path} res = httpx.post(base\_url + binaryfile\_catalog + move\_subcatalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def buff\_create(path: str, file\_name: str): try: data = {'directory\_path': path, 'file\_name': file\_name} res = httpx.post(base\_url + bufferfile\_catalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def buff\_delete(file\_path: str): try: res = httpx.delete(base\_url + bufferfile\_catalog + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def buff\_get(file\_path: str): try: res = httpx.get(base\_url + bufferfile\_catalog + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def buff\_move(file\_path: str, destination\_path: str): try: data = {'buffer\_file\_path': file\_path, 'destination\_path': destination\_path} res = httpx.post(base\_url + bufferfile\_catalog + move\_subcatalog, json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def buff\_push(file\_path: str, push\_element: str): try: data = {'buffer\_file\_path': file\_path, 'push\_element': push\_element} res = httpx.post(base\_url + bufferfile\_catalog + 'push', json=data, headers=headers) print(json.dumps(res.json())) except Exception: print('Error occurred')@app.command()def buff\_pop(file\_path: str): try: res = httpx.get(base\_url + bufferfile\_catalog + 'pop/' + file\_path) print(json.dumps(res.json())) except Exception: print('Error occurred')if \_\_name\_\_ == "\_\_main\_\_": app() |

Функціонал модуля включає в себе:

1. Керування каталогами:
   * отримання списку каталогів (dir\_get)
   * створення каталогу (dir\_create)
   * видалення каталогу (dir\_delete)
   * переміщення каталогу (dir\_move)
2. Робота з текстовими файлами (логами):
   * створення лог-файлу (log\_create)
   * додавання рядка у лог-файл (log\_log)
   * отримання вмісту лог-файлу (log\_get)
   * переміщення лог-файлу (log\_move)
   * видалення лог-файлу (log\_delete)
3. Робота з двійковими файлами:
   * створення двійкового файлу (bin\_create)
   * отримання вмісту двійкового файлу (bin\_get)
   * видалення двійкового файлу (bin\_delete)
   * переміщення двійкового файлу (bin\_move)
4. Робота з буферними файлами:
   * створення буферного файлу (buff\_create)
   * видалення буферного файлу (buff\_delete)
   * отримання вмісту буферного файлу (buff\_get)
   * переміщення буферного файлу (buff\_move)
   * додавання елемента до буферного файлу (buff\_push)
   * видалення елемента з буферного файлу (buff\_pop)

**Docker**

|  |
| --- |
| **Dockerfile** |
| # syntax=docker/dockerfile:1  FROM python:3.9  WORKDIR /home/mykola/Desktop/Testing  COPY requirements.txt /home/mykola/Desktop/Testing/requirements.txt RUN pip freeze > requirements.txt RUN pip3 install flask RUN pip3 install --upgrade pip -r requirements.txt  COPY . /home/mykola/Desktop/Testing  EXPOSE 5000  CMD [ "python3", "main.py"] |

Модуль дозволяє запустити наш додаток в Докер-контейнері

**Тестування cli**

|  |
| --- |
| **TestAPI.robot** |
| \*\*\* Settings \*\*\* Library *OperatingSystem* Library *JSONLibrary* Library *Process*  \*\*\* Test Cases \*\*\* Create Binary File  ${result}= RUN PROCESS *python3 cli.py bin-create root/ bin1 content* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Delete Binary File  ${result}= RUN PROCESS *python3 cli.py bin-delete root/bin1* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Create Directory  ${result}= RUN PROCESS *python3 cli.py dir-create root/ dir76* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Create Log File  ${result}= RUN PROCESS *python3 cli.py log-create root/ log1* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Delete Log File  ${result}= RUN PROCESS *python3 cli.py log-delete root/log1* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Create Buffer File  ${result}= RUN PROCESS *python3 cli.py buff-create root/ buff1* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Delete Buffer File  ${result}= RUN PROCESS *python3 cli.py buff-delete root/buff1* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Create Directory With Same Name  ${result}= RUN PROCESS *python3 cli.py dir-create root/ dir76* Log ${result.stdout}  ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json["error\_message"]} *Directory creation failed*  Delete Directory  ${result}= RUN PROCESS *python3 cli.py dir-delete root/dir76* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Create File With Same Name  ${result}= RUN PROCESS *python3 cli.py log-create root/ log1* ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json["error\_message"]} *File creation failed*   Create Directory With With Wrong Path  ${result}= RUN PROCESS *python3 cli.py dir-create root/asdfsd/asdasd dir76* ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json["error\_message"]} *Directory creation failed*  Delete Not Existing File  ${result}= RUN PROCESS *python3 cli.py log-delete root/sdfsdf/sdfsdf* ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json['error\_message']} *LogFile not found*  Delete Not Existing Directory  ${result}= RUN PROCESS *python3 cli.py dir-delete root/sdfsdf/sdfsdf* ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json['error\_message']} *Directory not found*  Delete Wrong File Type  RUN PROCESS *python3 app.py log-create root/ log1* ${result}= RUN PROCESS *python3 cli.py buff-delete root/log1* ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json['error\_message']} *BufferFile not found*  Log Line To LogFile  ${result}= RUN PROCESS *python3 cli.py log-log root/log1 line* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}  Log Line To Not LogFile  RUN PROCESS *python3 cli.py buff-create root/ buff1* ${result}= RUN PROCESS *python3 cli.py log-log root/buff1 line* ${json}= Convert String to JSON ${result.stdout}  should be equal as strings ${json['error\_message']} *File do not supports logging*  Push Element To BufferFile  ${result}= RUN PROCESS *python3 cli.py buff-push root/buff1 element1* ${json}= Convert String to JSON ${result.stdout}  should be true ${json['success']}    \*\*\* Keywords \*\*\* Read And Parse Output  ***[Arguments]*** ${path}  ${file\_content}= Get File ${path}  ${json}= Load JSON From String ${file\_content}  Should Be True ${json['success']} |

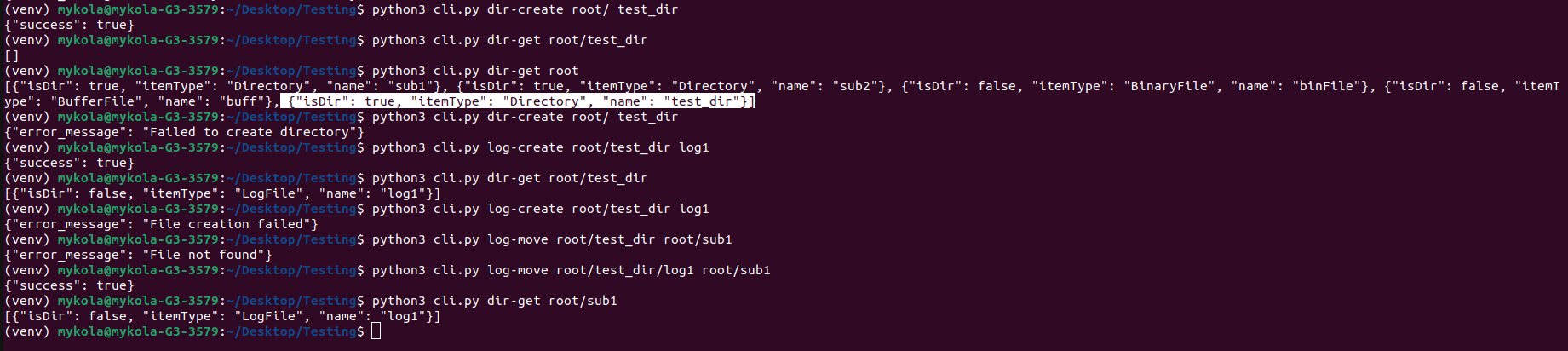
Модуль тестує усі підкласи за допомогою фреймворку Robot, порівнюючи відповідь від сервера з очікуваною.

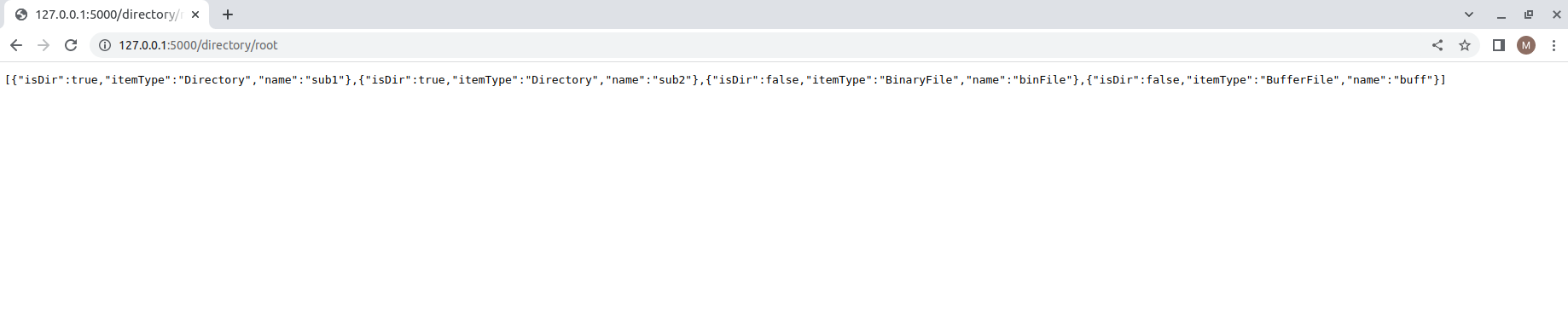
**Модуль файлової системи**

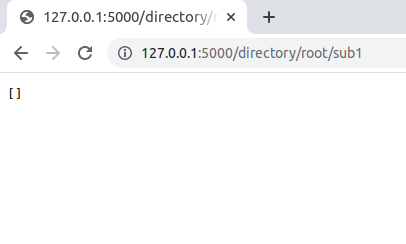
|  |
| --- |
| **fs.py** |
| from queue import Queue   class Node:  def \_\_init\_\_(self, itemName: str) -> None:  self.\_\_itemName = itemName  self.\_isDir = False   def getName(self) -> str:  return self.\_\_itemName   def isDir(self) -> bool:  return self.\_isDir   class BinaryFile(Node):  def \_\_init\_\_(self, fileName: str, fileContent: str) -> None:  self.\_content = fileContent  super().\_\_init\_\_(fileName)   def read(self) -> str:  return self.\_content   class LogFile(BinaryFile):  def \_\_init\_\_(self, fileName: str, fileContent: str) -> None:  super().\_\_init\_\_(fileName, fileContent)   def append(self, line) -> None:  if len(self.\_content) == 0:  self.\_content += line  else:  self.\_content += f"\n{line}"   class BufferFile(Node):  def \_\_init\_\_(self, fileName: str) -> None:  self.\_\_buffer = Queue()  super().\_\_init\_\_(fileName)   def push(self, element: str) -> bool:  if (self.\_\_buffer.qsize() + 1) < 1000:  self.\_\_buffer.put(element)  return True  else:  return False   def pop(self) -> str:  if self.\_\_buffer.empty():  return None  else:  element = self.\_\_buffer.get()  return element   def get\_elements(self):  return list(self.\_\_buffer.queue)   class Directory(Node):  def \_\_init\_\_(self, itemName: str) -> None:  super().\_\_init\_\_(itemName)  self.\_\_items = []  self.\_\_parent = None  self.\_\_itemsCount = 0  self.\_isDir = True   def addItem(self, item: Node) -> bool:   if self.getItemByName(item.getName()) != None or item == None:  return False   if self.\_\_itemsCount < 10:  self.\_\_itemsCount += 1  self.\_\_items.append(item)   if item.isDir():  item.setParent(self)   return True   return False   def popItem(self, itemName) -> Node:  item = self.getItemByName(itemName)  self.\_\_items.remove(item)  return item   def getItemByName(self, name: str) -> Node:  for item in self.\_\_items:  if item.getName() == name:  return item   return None   def items(self) -> list:  return self.\_\_items.copy()   def setParent(self, newParent):  self.\_\_parent = newParent   def getParent(self):  return self.\_\_parent   class FileSystem:  def \_\_init\_\_(self) -> None:  self.\_\_root = Directory("root")   def getRoot(self) -> Directory:  return self.\_\_root   def addItem(self, item, path: str) -> bool:  if '/' in item.getName():  return False   itemDir = self.findDir(path)  if itemDir == None or item == None:  return False   return itemDir.addItem(item)   def removeItem(self, itemPath: str) -> bool:  if len(itemPath.split('/')) == 1:  return False   item = self.\_\_getItemFromDir(itemPath)  if item == None:  return False   return True   def moveItem(self, itemPath: str, path: str) -> bool:  itemDir = self.findDir(path)   if itemDir == None or len(itemPath.split('/')) == 1:  return False   item = self.\_\_getItemFromDir(itemPath)   if item == None:  return False   if itemDir.addItem(item):  return True  else:  pathList = itemPath.split('/')  pathList.pop()  fileDirPath = '/'.join(pathList)  fileDir = self.findDir(fileDirPath)  fileDir.addItem(item)  return False   def \_\_getItemFromDir(self, itemPath: str) -> Node:  pathList = itemPath.split('/')  itemName = pathList.pop()  itemDirPath = '/'.join(pathList)  itemDir = self.findDir(itemDirPath)  item = itemDir.popItem(itemName)  return item   def findDir(self, path: str) -> Directory:  folders = path.split('/')  folders = list(filter(lambda item: item != '', folders))   if len(folders) == 0:  return None   rootName = folders.pop(0)   if self.\_\_root.getName() != rootName:  return None   searchedDir = self.\_\_root  for folder in folders:  subDir = searchedDir.getItemByName(folder)   if subDir == None or not subDir.isDir():  return None  searchedDir = subDir   return searchedDir   def findFile(self, filePath: str):  if filePath is None:  return None   pathList = filePath.split('/')  fileName = pathList.pop()  fileDirPath = '/'.join(pathList)  fileDir = self.findDir(fileDirPath)   if fileDir is None:  return None   return fileDir.getItemByName(fileName)   def get\_item\_type(item) -> str:  if item.isDir():  return 'Directory'   if type(item) is LogFile:  return 'LogFile'   if type(item) is BinaryFile:  return 'BinaryFile'   if type(item) is BufferFile:  return 'BufferFile'   return 'Unknown'   def get\_file\_content(file\_type: str, file: Node):  if file\_type == 'BinaryFile' or file\_type == 'LogFile':  return file.read()   if file\_type == 'BufferFile':  return file.get\_elements()   return ""   def get\_item\_info(item: Node):  return {'name': item.getName(), 'isDir': item.isDir(), 'itemType': get\_item\_type(item)}   def get\_items(items: list):  result = []  for item in items:  result.append(get\_item\_info(item))   return result   def get\_file\_data(file):  file\_data = get\_item\_info(file)  file\_content = get\_file\_content(file\_data['itemType'], file)  file\_data['file\_content'] = file\_content  return file\_data |

**Опис результатів роботи програми**

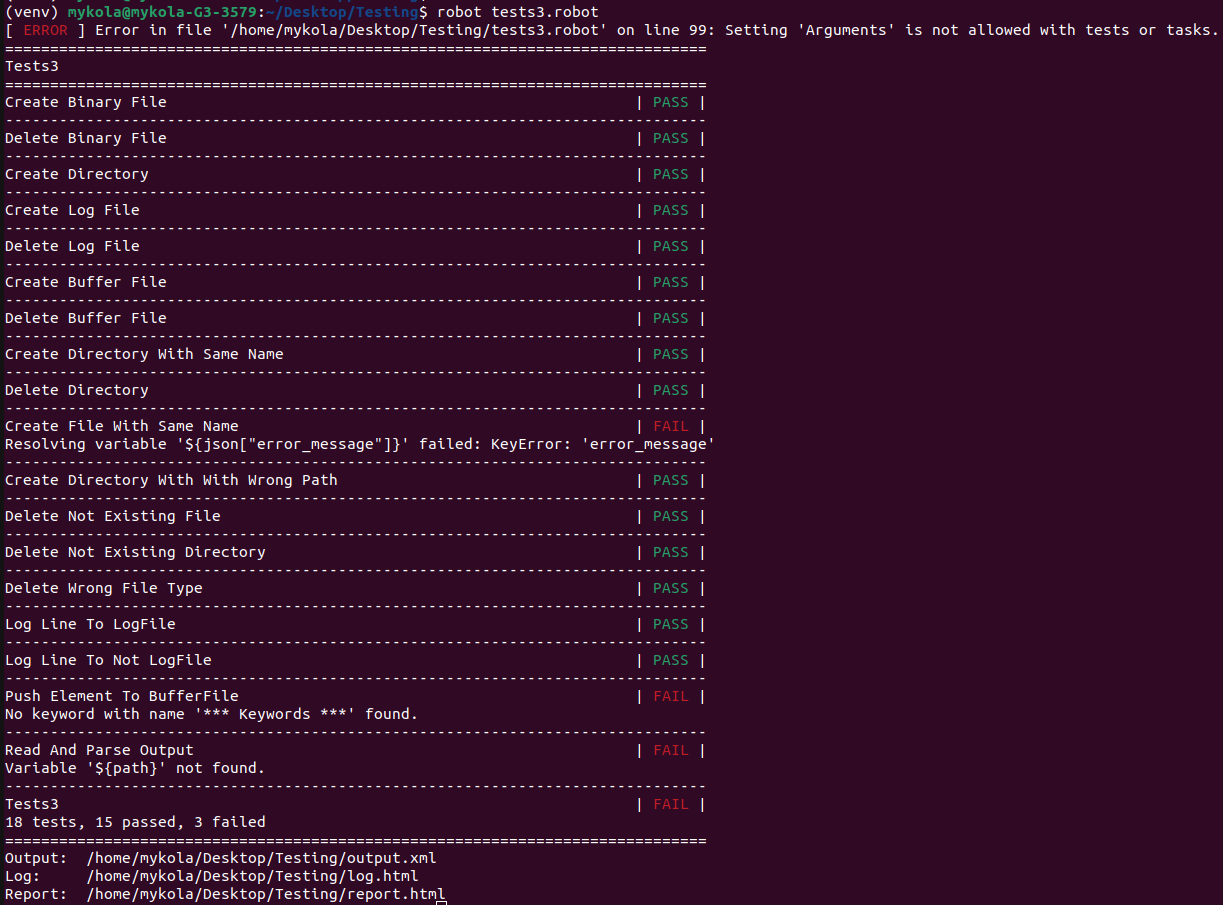
**Приклади виконання команд через інтерфейс користувача**

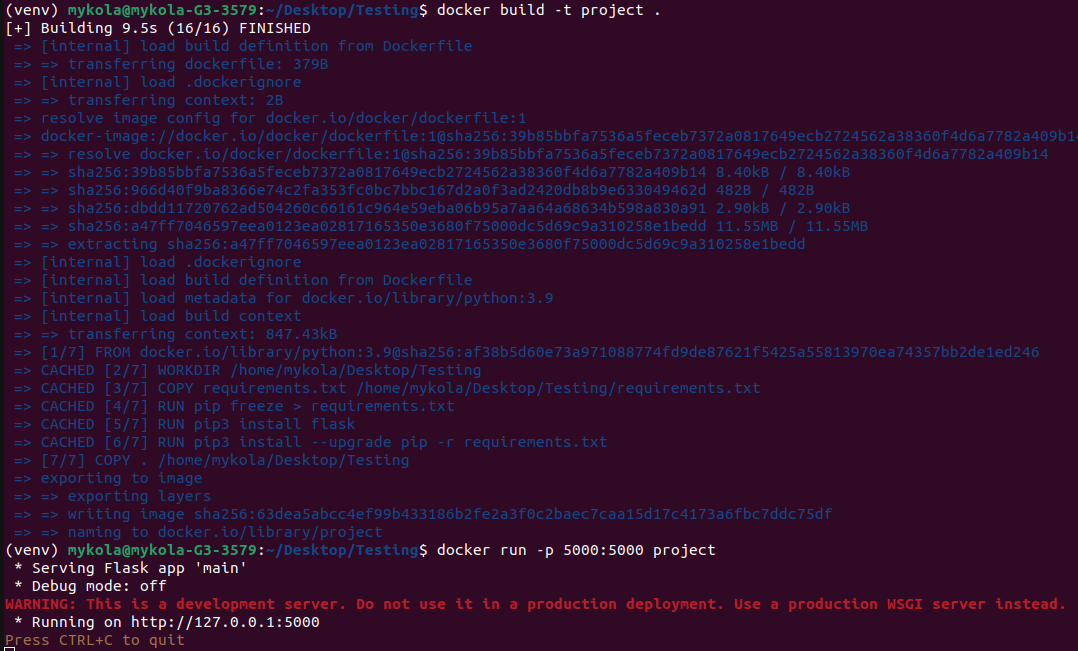
**Приклад роботи flask застосунку**

****

****

**Результат виконання robot тестів**

****

**Запуск програми з Docker container**

**ВИСНОВКИ**

Мета даного курсового проєкту полягала у створенні клієнт-серверної аплікації з Command Line Interface, що демонструє володіння навичками програмування та ефективність використання інструментів мови Python та її фреймворків. Проєкт передбачав проведення тестів для перевірки коректності роботи програми та виявлення можливих помилок у коді.