МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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»

тема “Файлова система в пам'яті ”

|  |  |  |  |
| --- | --- | --- | --- |
| Виконав  Студент IІІ курсу  групи КП-01  Беліцький Олександр Сергійович |  | Оцінка   |  | | --- | |  |   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (дата, підпис) |

Київ 2023

Мета роботи

Опанувати навички програмування мовою Python, опанувати роботу з інструментами для тестування Pytest та Robot Framework. Набути знань та навичок роботи з Docker.

Інструменти виконання роботи

Python - 3.8.10

PyTest - 7.2.0

Flask - 2.2.2

Robot - 6.0.1

Docker - 20.10.12

Linux/Ubuntu - 20.04

Visual Studio Code - 1.74

PyCharm - 17.0.5

Завдання

Завдання 1

Створити in-memory File System (FS). Файлова система складається з 4 типів вузлів:

1. Directory - може містити інші каталоги та файли. Каталог може бути порожнім або містити декілька елементів. Кількість елементів у каталозі має бути <= DIR\_MAX\_ELEMS

Дозволені операції::

* Create directory
* Delete directory
* List files and subdirectories
* Move file or subdirectory to another location

2. Binary file - незмінний файл, який містить певну інформацію.

Дозволені операції::

* Create file
* Delete file
* Move file
* Readfile (returns file content)

3. Log text file - текстовий файл, який можна змінювати, додаючи рядки в кінець файлу.

Дозволені операції::

* Create file
* Delete file
* Move file
* Readfile (returns file content)
* Append a line to the end of the file

4. Buffer file - це особливий тип файлу, який працює як черга. Деякі потоки надсилають елементи до файлу, інші витягують елементи з файлу. Номер елемента у файлі дорівнює <= MAX\_BUF\_FILE\_SIZE

Дозволені операції:

* Create file
* Delete file
* Move file
* Push element
* Consume element

Завданння 2

З точки зору поточного завдання вам потрібно розробити додаток HTTP (Restful), який дозволить вам виконувати абсолютно ті ж команди, що і в попередньому завданні, використовуючи протокол HTTP.

Слід визначити наступні ресурси:

* /directory
* /binaryfile
* /logtextfile
* /bufferfile

Завдання 3

Додаток з Завдання 2 запустити в докер контейнері.

Створити додаток - клієнт. CLI - command line interface.

Код виконання завдань

Завдання 1

|  |
| --- |
| BinaryFile.py |
| from directory import Directory  from logtextfile import LogTextFile  class BinaryFile:  #Constr  def \_\_init\_\_(self, directory, logg):  self.\_\_name = "BinaryFile.bin"  self.context = "Something is here"  self.\_\_directory = directory  directory.list.append(self)  self.log = logg  self.log.append\_context("\n" + self.get\_name() + ": created")  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name + ".bin"  self.log.append\_context("\n" + self.get\_name() + ": was renamed")    #Move  def move(self, new\_repo):  new\_repo.list.append(self)  self.\_\_directory.list.remove(self)  self.\_\_directory = new\_repo  self.log.append\_context("\n" + self.get\_name() + ": moved to " + new\_repo.get\_name())  #Read  def get\_context(self):  return self.context  def get\_direcrory\_name(self):  return self.\_\_directory.get\_name()  # Destruc  def delete(self):  if(self.\_\_name == "None"):  raise FileExistsError("Binary file not exists")  self.\_\_directory.list.remove(self)  self.append\_context("\n" + self.get\_name() + ": was removed")  self.\_\_name = "None" |

|  |
| --- |
| BufferFile.py |
| from directory import Directory  from logtextfile import LogTextFile  class BufferFile:  #Constr  def \_\_init\_\_(self, size, directory, log):  self.\_\_name = "Buffer.buf"  self.\_\_size = size  self.list = list()  self.\_\_directory = directory  directory.list.append(self)  self.log = log  self.log.append\_context("\n" + self.get\_name() + ": created")  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name + ".buf"  self.log.append\_context("\n" + self.get\_name() + ": was renamed")      #Move  def move(self, new\_repo):  new\_repo.list.append(self)  self.\_\_directory.list.remove(self)  self.\_\_directory = new\_repo  self.log.append\_context("\n" + self.get\_name() + ": moved to " + new\_repo.get\_name())  #Read TODO  def get\_context(self):  return self.list  def get\_direcrory\_name(self):  return self.\_\_directory.get\_name()  #Append  def append\_queue(self, item):  if len(self.list) == self.\_\_size:  raise OverflowError("max size reached")  self.list.append(item)  self.log.append\_context("\n" + self.get\_name() + ": append queue")  def first\_out(self):  self.log.append\_context("\n" + self.get\_name() + ": poped")  return self.list.pop(0)    # Destruc  def delete(self):  if(self.\_\_name == "None"):  raise FileExistsError("Buffer not exist")  self.\_\_directory.list.remove(self)  self.append\_context("\n" + self.get\_name() + ": was removed")  self.\_\_name = "None" |

|  |
| --- |
| Directory.py |
| class Directory:    #Constr  def \_\_init\_\_(self, name = "autodir"):  self.\_\_name = name  self.list = list()  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name  #Move  def move\_repository(self, new\_repo):  if new\_repo in self.list:  self.list.remove(new\_repo)  new\_repo.list.append(self)  return  new\_repo.list.append(self)  self.directory = new\_repo  #Read  def sort\_list(self):  new\_dir\_list = []  for item in self.list:  if not item.get\_name().endswith(".bin") or not item.get\_name().endswith(".buf") or not item.get\_name().endswith(".lg"):  new\_dir\_list.append(item)  for item in self.list:  if item.get\_name().endswith(".bin"):  new\_dir\_list.append(item)  for item in self.list:  if item.get\_name().endswith(".buf"):  new\_dir\_list.append(item)  for item in self.list:  if item.get\_name().endswith(".lg"):  new\_dir\_list.append(item)  self.list = new\_dir\_list    def print\_list(self):  #self.sort\_list()  print("~~~~" + self.get\_name() + "~~~~")  for item in self.list:  if(item.get\_name().endswith(".bin")):  print("\033[31m{}\033[0m".format(item.get\_name()))  continue  elif(item.get\_name().endswith(".buf")):  print("\033[32m{}\033[0m".format(item.get\_name()))  continue  elif(item.get\_name().endswith(".lg")):  print("\033[33m{}\033[0m".format(item.get\_name()))  continue  else:  print("\033[34m{}\033[0m".format(item.get\_name()))    # Destruc  def delete(self):  if(self.\_\_name == "None"):  raise FileExistsError("Dir not exists")  print(self.get\_name() + " was removed")  self.\_\_name = "None"  self.list = list() |

|  |
| --- |
| LogTextFile.py |
| from directory import Directory  class LogTextFile:  #Constr  def \_\_init\_\_(self):  self.\_\_name = "Logs.lg"  self.context = "Begginning:"  self.\_\_directory = None  self.append\_context("\n" + self.get\_name() + ": created")  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name +".lg"  self.append\_context("\n" + self.get\_name() + ": was renamed")    #Move  def move(self, new\_repo):  new\_repo.list.append(self)  if not self.\_\_directory == None:  self.\_\_directory.list.remove(self)  self.\_\_directory = new\_repo  self.append\_context("\n" + self.get\_name() + ": moved to " + new\_repo.get\_name())  #Read  def get\_context(self):  return self.context  def get\_direcrory\_name(self):  return self.\_\_directory.get\_name()  #Append  def append\_context(self, message: str):  self.context += message  # Destruc  def delete(self):  if(self.\_\_name == "None"):  raise FileExistsError("Logger not exists")  self.\_\_directory.list.remove(self)  self.append\_context("\n" + self.get\_name() + ": was removed")  self.\_\_name = "None" |

|  |
| --- |
| main.py |
| from binaryFile import BinaryFile  from directory import Directory  from bufferFile import BufferFile  from logtextfile import LogTextFile  log = LogTextFile()  home\_dir= Directory("home")  log.move(home\_dir)  dir1 = Directory("dir1")  dir1.move\_repository(home\_dir)  dir2 = Directory("dir2")  dir2.move\_repository(dir1)  bin1 = BinaryFile(dir1, log)  bin2 = BinaryFile(dir1, log)  bin3 = BinaryFile(dir2, log)  buf = BufferFile(3, dir1, log)  home\_dir.print\_list()  dir1.print\_list()  dir2.print\_list()  #Editing  bin3.set\_name("RenamedBin")  print(bin3.get\_context())  #Buff  buf.append\_queue("Smth1")  buf.append\_queue("Smth2")  buf.append\_queue("Smth3")  print("~~~~" + buf.get\_name() +"~~~~")  print(buf.get\_context())  buf.first\_out()  print("~~~~" + buf.get\_name() +"~~~~")  print(buf.get\_context())  #Dir move  dir3 = Directory("dir3")  dir3.move\_repository(dir2)  dir2.print\_list()  dir2.move\_repository(dir3)  dir3.print\_list()  dir2.print\_list()  #Ending  print("~~~~" + log.get\_name() +"~~~~\n" + log.get\_context()) |

Завдання 2

|  |
| --- |
| BinaryFile.py |
| from directory import Directory  from logtextfile import LogTextFile  class BinaryFile:  # Constr  def \_\_init\_\_(self, directory, logg, name):  if name == "":  name = "BinaryFile"  for item in directory.list:  if item.get\_name() == name + ".bin":  name += "\*"  self.\_\_name = name + ".bin"  self.context = "Something is here"  self.\_\_directory = directory  directory.list.append(self)  self.log = logg  self.log.append\_context("\n" + self.get\_name() + ": created")  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name + ".bin"  self.log.append\_context("\n" + self.get\_name() + ": was renamed")  # Move  def move(self, new\_repo):  new\_repo.list.append(self)  self.\_\_directory.list.remove(self)  for item in new\_repo.list:  if item.get\_name() == self.\_\_name:  name = self.\_\_name[0: self.\_\_name.find('.')] + "`" + ".bin"  self.\_\_name = name  break  self.\_\_directory = new\_repo  self.log.append\_context("\n" + self.get\_name() + ": moved to " + new\_repo.get\_name())  # Read  def get\_context(self):  return self.context  def get\_direcrory\_name(self):  return self.\_\_directory.get\_name()  # Destruc  def delete(self):  if (self.\_\_name == "None"):  raise FileExistsError("Binary file not exists")  self.\_\_directory.list.remove(self)  self.log.append\_context("\n" + self.get\_name() + ": was removed")  self.\_\_name = "None" |

|  |
| --- |
| BufferFile.py |
| from directory import Directory  from logtextfile import LogTextFile  class BufferFile:  # Constr  def \_\_init\_\_(self, size, directory, log, name):  if name == "":  name = "Buffer"  for item in directory.list:  if item.get\_name() == name + ".buf":  name += "\*"  self.\_\_name = name + ".buf"  self.\_\_size = size  self.list = list()  self.\_\_directory = directory  directory.list.append(self)  self.log = log  self.log.append\_context("\n" + self.get\_name() + ": created")  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name + ".buf"  self.log.append\_context("\n" + self.get\_name() + ": was renamed")  # Move  def move(self, new\_repo):  new\_repo.list.append(self)  self.\_\_directory.list.remove(self)  for item in new\_repo.list:  if item.get\_name() == self.\_\_name:  name = self.\_\_name[0: self.\_\_name.find('.')] + "`" + ".buf"  self.\_\_name = name  break  self.\_\_directory = new\_repo  self.log.append\_context("\n" + self.get\_name() + ": moved to " + new\_repo.get\_name())  # Read  def get\_context(self):  return self.list  def get\_direcrory\_name(self):  return self.\_\_directory.get\_name()  # Append  def append\_queue(self, item):  if len(self.list) == self.\_\_size:  raise OverflowError("max size reached")  self.list.append(item)  self.log.append\_context("\n" + self.get\_name() + ": append queue")  def first\_out(self):  self.log.append\_context("\n" + self.get\_name() + ": poped")  return self.list.pop(0)  # Destruc  def delete(self):  if (self.\_\_name == "None"):  raise FileExistsError("Buffer not exist")  self.\_\_directory.list.remove(self)  self.log.append\_context("\n" + self.get\_name() + ": was removed")  self.\_\_name = "None" |

|  |
| --- |
| Directory.py |
| class Directory:  # Constr  def \_\_init\_\_(self, name="autodir"):  self.\_\_name = name  self.list = list()  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name  # Move  def move\_repository(self, new\_repo):  for item in new\_repo.list:  if item.get\_name() == self.\_\_name:  name = self.\_\_name + "`"  self.\_\_name = name  break  if new\_repo in self.list:  self.list.remove(new\_repo)  new\_repo.list.append(self)  return  new\_repo.list.append(self)  self.directory = new\_repo  # Read  def sort\_list(self):  new\_dir\_list = []  for item in self.list:  if not item.get\_name().endswith(".bin") or not item.get\_name().endswith(  ".buf") or not item.get\_name().endswith(".lg"):  new\_dir\_list.append(item)  for item in self.list:  if item.get\_name().endswith(".bin"):  new\_dir\_list.append(item)  for item in self.list:  if item.get\_name().endswith(".buf"):  new\_dir\_list.append(item)  for item in self.list:  if item.get\_name().endswith(".lg"):  new\_dir\_list.append(item)  self.list = new\_dir\_list  return new\_dir\_list  def print\_list(self):  # self.sort\_list()  print("~~~~" + self.get\_name() + "~~~~")  for item in self.list:  if (item.get\_name().endswith(".bin")):  print("\033[31m{}\033[0m".format(item.get\_name()))  continue  elif (item.get\_name().endswith(".buf")):  print("\033[32m{}\033[0m".format(item.get\_name()))  continue  elif (item.get\_name().endswith(".lg")):  print("\033[33m{}\033[0m".format(item.get\_name()))  continue  else:  print("\033[34m{}\033[0m".format(item.get\_name()))  # Destruc  def delete(self):  if (self.\_\_name == "None"):  raise FileExistsError("Dir not exists")  print(self.get\_name() + " was removed")  self.\_\_name = "None"  self.list = list()  def delete\_directory(self, dir):  if dir in self.list:  self.list.remove(dir)  dir = Directory() |

|  |
| --- |
| LogTextFile.py |
| from directory import Directory  class LogTextFile:  # Constr  def \_\_init\_\_(self):  self.\_\_name = "Logs.lg"  self.context = "Begginning:"  self.\_\_directory = None  self.append\_context("\n" + self.get\_name() + ": created")  def get\_name(self):  return self.\_\_name  def set\_name(self, name):  self.\_\_name = name + ".lg"  self.append\_context("\n" + self.get\_name() + ": was renamed")  # Move  def move(self, new\_repo):  new\_repo.list.append(self)  if not self.\_\_directory == None:  self.\_\_directory.list.remove(self)  self.\_\_directory = new\_repo  self.append\_context("\n" + self.get\_name() + ": moved to " + new\_repo.get\_name())  # Read  def get\_context(self):  return self.context  def get\_direcrory\_name(self):  return self.\_\_directory.get\_name()  # Append  def append\_context(self, message: str):  self.context += message  # Destruc  def delete(self):  if (self.\_\_name == "None"):  raise FileExistsError("Logger not exists")  self.\_\_directory.list.remove(self)  self.append\_context("\n" + self.get\_name() + ": was removed")  self.\_\_name = "None" |

|  |
| --- |
| main.py |
| from flask import Flask, render\_template, request, url\_for, redirect  from binary\_file import BinaryFile  from directory import Directory  from bufferFile import BufferFile  from logtextfile import LogTextFile  app = Flask(\_\_name\_\_)  home = Directory("home")  log = LogTextFile()  log.move(home)  binary = BinaryFile(home, log, "Binn")  binaryf = BinaryFile(home, log, "Bins")  bufferfile = BufferFile(5, home, log, "")  nested\_dir = Directory("nested\_dir")  nested\_dir.move\_repository(home)  @app.route('/')  def main\_page():  return render\_template("main\_page.html", home=home)  #####Bin block  @app.route('/binaryfile')  def binaryfile\_page():  return render\_template("binaryfile.html")  @app.route('/binaryfile\_create', methods=['POST', 'GET'])  def binaryfile\_create():  if request.method == 'POST':  name = request.form['name']  binaryfile = BinaryFile(home, log, name)  return redirect('/')  else:  return render\_template('binaryfile\_create.html')  @app.route('/binaryfile\_read/<string:name>')  def binaryfile\_read(name):  binF = ""  for item in home.list:  if item.get\_name() == name + ".bin":  binF = item  if binF == "":  return redirect('/binaryfile')  text = binF.get\_context()  return render\_template("binaryfile\_read.html", text=text)  @app.route('/binaryfile\_del/<string:name>', methods=['POST', 'GET'])  def binaryfile\_delete(name):  binF = ""  for item in home.list:  if item.get\_name() == name + ".bin":  binF = item  if binF == "":  return redirect('/binaryfile')  if request.method == 'POST':  binF.delete()  return redirect('/')  else:  return render\_template("binaryfile\_del.html")  @app.route('/binaryfile\_move/<string:name>', methods=['POST', 'GET'])  def binaryfile\_move(name):  binF = ""  for item in home.list:  if item.get\_name() == name + ".bin":  binF = item  if binF == "":  return redirect('/binaryfile')  if request.method == 'POST':  name = request.form['name']  dir = ""  for item in home.list:  if item.get\_name() == name:  dir = item  if dir == "":  return redirect('/binaryfile')  binF.move(dir)  return redirect('/')  else:  return render\_template("binaryfile\_move.html")  #####Buf block  @app.route('/bufferfile')  def bufferfile\_page():  return render\_template("bufferfile.html")  @app.route('/bufferfile\_create', methods=['POST', 'GET'])  def bufferfile\_create():  if request.method == 'POST':  name = request.form['name']  size = request.form['size']  bufferfile = BufferFile(size, home, log, name)  return redirect('/')  else:  return render\_template('bufferfile\_create.html')  @app.route('/bufferfile\_read/<string:name>')  def bufferfile\_read(name):  bufF = ""  for item in home.list:  if item.get\_name() == name + ".buf":  bufF = item  if bufF == "":  return redirect('/bufferfile')  text = bufF.get\_context()  return render\_template("bufferfile\_read.html", text=text)  @app.route('/bufferfile\_add/<string:name\_buf>', methods=['POST', 'GET'])  def bufferfile\_add(name\_buf):  bufF = ""  for item in home.list:  if item.get\_name() == name\_buf + ".buf":  bufF = item  if bufF == "":  return redirect('/bufferfile')  if request.method == 'POST':  name = request.form['name']  bufF.append\_queue(name)  return redirect('/bufferfile\_read/<string:name>')  else:  return render\_template('bufferfile\_add.html')  @app.route('/bufferfile\_pop/<string:name\_buf>', methods=['POST', 'GET'])  def bufferfile\_pop(name\_buf):  bufF = ""  for item in home.list:  if item.get\_name() == name\_buf + ".buf":  bufF = item  if bufF == "":  return redirect('/bufferfile')  if request.method == 'POST':  bufF.first\_out()  return redirect('/bufferfile\_read/<string:name>')  else:  return render\_template('bufferfile\_pop.html')  @app.route('/bufferfile\_del/<string:name>', methods=['POST', 'GET'])  def bufferfile\_delete(name):  bufF = ""  for item in home.list:  if item.get\_name() == name + ".buf":  bufF = item  if bufF == "":  return redirect('/bufferfile')  if request.method == 'POST':  bufF.delete()  return redirect('/')  else:  return render\_template("bufferfile\_del.html")  @app.route('/bufferfile\_move/<string:name>', methods=['POST', 'GET'])  def bufferfile\_move(name):  bufF = ""  for item in home.list:  if item.get\_name() == name + ".buf":  bufF = item  if bufF == "":  return redirect('/bufferfile')  if request.method == 'POST':  name = request.form['name']  dir = ""  for item in home.list:  if item.get\_name() == name:  dir = item  if dir == "":  return redirect('/bufferfile')  bufF.move(dir)  return redirect('/')  else:  return render\_template("bufferfile\_move.html")  #####Log block  @app.route('/logtextfile')  def logtextfile\_page():  return render\_template("logtextfile.html")  @app.route('/logtextfile\_create', methods=['POST', 'GET'])  def logtextfile\_create():  if request.method == 'POST':  return redirect('/')  else:  return render\_template('logtextfile\_create.html')  @app.route('/logtextfile\_read')  def logtextfile\_read():  text = log.get\_context()  return render\_template("logtextfile\_read.html", text=text)  @app.route('/logtextfile\_del', methods=['POST', 'GET'])  def logtextfile\_delete():  if request.method == 'POST':  return redirect('/')  else:  return render\_template("logtextfile\_del.html")  ### Directory block  @app.route('/directory')  def directory\_page():  return render\_template("directory.html")  @app.route('/directory\_create', methods=['POST', 'GET'])  def directory\_create():  if request.method == 'POST':  name = request.form['name']  directory = Directory(name)  directory.move\_repository(home)  return redirect('/')  else:  return render\_template('directory\_create.html')  @app.route('/directory\_read/<string:name>')  def directory\_read(name):  dirF = ""  for item in home.list:  if item.get\_name() == name:  dirF = item  if dirF == "":  return redirect('/directory')  list = dirF.list  return render\_template("directory\_read.html", list=list)  @app.route('/directory\_del/<string:name>', methods=['POST', 'GET'])  def directory\_delete(name):  dirF = ""  for item in home.list:  if item.get\_name() == name:  dirF = item  if dirF == "":  return redirect('/directory')  if request.method == 'POST':  home.delete\_directory(dirF)  dirF.delete()  return redirect('/')  else:  return render\_template("directory\_del.html")  @app.route('/directory\_move/<string:name>', methods=['POST', 'GET'])  def directory\_move(name):  dirF = ""  for item in home.list:  if item.get\_name() == name:  dirF = item  if dirF == "":  return redirect('/directory')  if request.method == 'POST':  name = request.form['name']  dir = ""  for item in home.list:  if item.get\_name() == name:  dir = item  if dir == "":  return redirect('/directory')  home.delete\_directory(dirF)  dirF.move\_repository(dir)  return redirect('/')  else:  return render\_template("directory\_move.html")  if \_\_name\_\_ == "\_\_main\_\_":  app.run(debug=True) |

Шаблони з використанням Flask доступні за посиланням:

*https://github.com/BelitskyiAlexandr/qa-kp01-Belitskyi/tree/main/sem\_1\_lab\_2/ templates*

|  |
| --- |
| test\_server.py |
| import requests  import json  from flask import Flask, render\_template, request, url\_for, redirect  from main import app  flask\_app = app  def test\_main\_page():  with flask\_app.test\_client() as test\_client:  response = test\_client.get('/')  assert response.status\_code == 200  def test\_binary\_pages():  with flask\_app.test\_client() as test\_client:  response = test\_client.get('/binaryfile')  assert response.status\_code == 200  response = test\_client.get('/binaryfile\_create')  assert response.status\_code == 200  response = test\_client.get('/binaryfile\_read/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/binaryfile\_del/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/binaryfile\_move/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/binaryfile\_read')  assert response.status\_code == 404  response = test\_client.get('/binaryfile\_del')  assert response.status\_code == 404  response = test\_client.get('/binaryfile\_move')  assert response.status\_code == 404  def test\_buffer\_pages():  with flask\_app.test\_client() as test\_client:  response = test\_client.get('/bufferfile')  assert response.status\_code == 200  response = test\_client.get('/bufferfile\_create')  assert response.status\_code == 200  response = test\_client.get('/bufferfile\_read/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/bufferfile\_del/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/bufferfile\_move/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/bufferfile\_add/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/bufferfile\_pop/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/bufferfile\_read')  assert response.status\_code == 404  response = test\_client.get('/bufferfile\_del')  assert response.status\_code == 404  response = test\_client.get('/bufferfile\_move')  assert response.status\_code == 404  response = test\_client.get('/bufferfile\_add')  assert response.status\_code == 404  response = test\_client.get('/bufferfile\_pop')  assert response.status\_code == 404  def test\_logtextfile\_pages():  with flask\_app.test\_client() as test\_client:  response = test\_client.get('/logtextfile')  assert response.status\_code == 200  response = test\_client.get('/logtextfile\_create')  assert response.status\_code == 200  response = test\_client.get('/logtextfile\_del')  assert response.status\_code == 200  response = test\_client.get('/logtextfile\_read')  assert response.status\_code == 200  response = test\_client.get('/logtextfile\_del/<string:name>')  assert response.status\_code == 404  response = test\_client.get('/logtextfile\_read/<string:name>')  assert response.status\_code == 404  response = test\_client.get('/logtextfile\_move')  assert response.status\_code == 404  def test\_directory\_pages():  with flask\_app.test\_client() as test\_client:  response = test\_client.get('/directory')  assert response.status\_code == 200  response = test\_client.get('/directory\_create')  assert response.status\_code == 200  response = test\_client.get('/directory\_read/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/directory\_del/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/directory\_move/<string:name>')  assert response.status\_code == 302  response = test\_client.get('/directory\_read')  assert response.status\_code == 404  response = test\_client.get('/directory\_del')  assert response.status\_code == 404  response = test\_client.get('/directory\_move')  assert response.status\_code == 404  def test\_binaryfile\_create():  ENDPOINT = "http://127.0.0.1:5000/binaryfile\_create"  response = requests.get(ENDPOINT)  #data = response.text  #print(data)  payload = {  "name": "Binn"  }  response = requests.post(ENDPOINT, json=payload)  data = response.text  print(data)  name = "Binn"  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/binaryfile\_read/{name}")  assert check\_existance\_response.status\_code == 200  data = check\_existance\_response.text  print(data)  def test\_binary\_delete():  name = "Binary"  ENDPOINT = f"http://127.0.0.1:5000/binaryfile\_del/{name}"  response = requests.post(ENDPOINT)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/binaryfile\_read/{name}")  assert check\_existance\_response.status\_code == 200 #redirect  data = check\_existance\_response.text  print(data)  def test\_binary\_move():  name = "Binary"  ENDPOINT = f"http://127.0.0.1:5000/binaryfile\_move/{name}"  payload = {  "name": "home"  }  response = requests.post(ENDPOINT, json=payload)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/binaryfile\_read/{name}")  assert check\_existance\_response.status\_code == 200 # redirect  data = check\_existance\_response.text  print(data)  def test\_bufferfile\_create():  ENDPOINT = "http://127.0.0.1:5000/bufferfile\_create"  response = requests.get(ENDPOINT)  payload = {  "name": "Buffer"  }  response = requests.post(ENDPOINT, json=payload)  data = response.text  print(data)  name = "Buffer"  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/bufferfile\_read/{name}")  assert check\_existance\_response.status\_code == 200  data = check\_existance\_response.text  print(data)  def test\_buffer\_add():  name = "Buffer"  ENDPOINT = f"http://127.0.0.1:5000/bufferfile\_add/{name}"  payload = {  "name": "smth"  }  response = requests.post(ENDPOINT, json=payload)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/bufferfile\_read/{name}")  assert check\_existance\_response.status\_code == 200 # redirect  data = check\_existance\_response.text  print(data)  def test\_buffer\_pop():  name = "Buffer"  ENDPOINT = f"http://127.0.0.1:5000/bufferfile\_pop/{name}"  response = requests.post(ENDPOINT)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/bufferfile\_read/{name}")  assert check\_existance\_response.status\_code == 200 # redirect  data = check\_existance\_response.text  print(data)  def test\_bufferfile\_delete():  name = "Buffer"  ENDPOINT = f"http://127.0.0.1:5000/bufferfile\_del/{name}"  response = requests.post(ENDPOINT)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/bufferfile\_read/{name}")  assert check\_existance\_response.status\_code == 200 #redirect  data = check\_existance\_response.text  print(data)  def test\_buffer\_move():  name = "Buffer"  ENDPOINT = f"http://127.0.0.1:5000/bufferfile\_move/{name}"  payload = {  "name": "home"  }  response = requests.post(ENDPOINT, json=payload)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/bufferfile\_read/{name}")  assert check\_existance\_response.status\_code == 200 # redirect  data = check\_existance\_response.text  print(data)  def test\_logtextfile\_create():  ENDPOINT = "http://127.0.0.1:5000/logtextfile\_create"  response = requests.get(ENDPOINT)  assert response.status\_code == 200  data = response.text  print(data)  def test\_logtextfile\_delete():  ENDPOINT = "http://127.0.0.1:5000/logtextfile\_del"  response = requests.get(ENDPOINT)  assert response.status\_code == 200  data = response.text  print(data)  def test\_logtextfile\_move():  ENDPOINT = "http://127.0.0.1:5000/bufferfile\_move"  response = requests.get(ENDPOINT)  assert response.status\_code == 404  def test\_binaryfile\_create():  ENDPOINT = "http://127.0.0.1:5000/directory\_create"  response = requests.get(ENDPOINT)  payload = {  "name": "Directory"  }  response = requests.post(ENDPOINT, json=payload)  data = response.text  print(data)  name = "Directory"  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/directory\_read/{name}")  assert check\_existance\_response.status\_code == 200  data = check\_existance\_response.text  print(data)  def test\_directory\_delete():  name = "Directory"  ENDPOINT = f"http://127.0.0.1:5000/directory\_del/{name}"  response = requests.post(ENDPOINT)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/directory\_read/{name}")  assert check\_existance\_response.status\_code == 200 #redirect  data = check\_existance\_response.text  print(data)  def test\_directory\_move():  name = "Directory"  ENDPOINT = f"http://127.0.0.1:5000/directory\_move/{name}"  payload = {  "name": "dir1"  }  response = requests.post(ENDPOINT, json=payload)  check\_existance\_response = requests.get(f"http://127.0.0.1:5000/directory\_read/{name}")  assert check\_existance\_response.status\_code == 200 # redirect  data = check\_existance\_response.text  print(data) |

|  |
| --- |
| test\_bufferfile.py |
| import pytest  from bufferFile import BufferFile  from directory import Directory  from logtextfile import LogTextFile  def test\_init():  with pytest.raises(TypeError):  BufferFile()  def test\_name():  dir = Directory()  log = LogTextFile()  buf = BufferFile(5, dir, log, "")  assert buf.get\_name() == "Buffer.buf"  buf.set\_name("New name")  assert buf.get\_name() == "New name.buf"  def test\_good\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir2")  log = LogTextFile()  buf = BufferFile(5, dir1, log, "")  buf.move(dir2)  assert buf.get\_direcrory\_name() == "dir2"  def test\_name\_after\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir2")  log = LogTextFile()  buf1 = BufferFile(5, dir1, log, "")  buf2 = BufferFile(5, dir2, log, "")  buf1.move(dir2)  assert buf1.get\_name() == "Buffer`.buf"  def test\_content():  dir = Directory()  log = LogTextFile()  buf = BufferFile(5, dir, log, "")  assert buf.get\_context() == []  def test\_delete():  dir = Directory()  log = LogTextFile()  buf = BufferFile(5, dir, log, "")  buf.delete()  assert dir.list == []  def test\_queue():  dir = Directory()  log = LogTextFile()  buf = BufferFile(1, dir, log, "")  buf.append\_queue("ss")  with pytest.raises(OverflowError):  buf.append\_queue("qq")  def test\_pop():  dir = Directory()  log = LogTextFile()  buf = BufferFile(1, dir, log, "")  buf.append\_queue("ss")  assert buf.first\_out() == "ss"  assert buf.list == []  def test\_redelete():  dir = Directory()  log = LogTextFile()  buf = BufferFile(3, dir, log, "")  buf.delete()  with pytest.raises(FileExistsError):  buf.delete() |

|  |
| --- |
| test\_binaryFile.py |
| import pytest  from binary\_file import BinaryFile  from directory import Directory  from logtextfile import LogTextFile  def test\_init():  with pytest.raises(TypeError):  BinaryFile()  def test\_name():  dir = Directory()  log = LogTextFile()  bin = BinaryFile(dir, log, "")  assert bin.get\_name() == "BinaryFile.bin"  bin.set\_name("New name")  assert bin.get\_name() == "New name.bin"  def test\_good\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir2")  log = LogTextFile()  bin = BinaryFile(dir1, log, "")  bin.move(dir2)  assert bin.get\_direcrory\_name() == "dir2"  def test\_name\_after\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir2")  log = LogTextFile()  bin1 = BinaryFile(dir1, log, "")  bin2 = BinaryFile(dir2, log, "")  bin1.move(dir2)  assert bin1.get\_name() == "BinaryFile`.bin"  def test\_content():  dir = Directory()  log = LogTextFile()  bin = BinaryFile(dir, log, "")  assert bin.get\_context() == "Something is here"  def test\_delete():  dir = Directory()  log = LogTextFile()  bin = BinaryFile(dir, log, "")  bin.delete()  assert dir.list == []  def test\_redelete():  dir = Directory()  log = LogTextFile()  bin = BinaryFile(dir, log, "")  bin.delete()  with pytest.raises(FileExistsError):  bin.delete() |

|  |
| --- |
| test\_directory.py |
| import pytest  from directory import Directory  def test\_init():  dir = Directory()  assert dir.get\_name() == "autodir"  dir.set\_name("dir1")  assert dir.get\_name() == "dir1"  def test\_good\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir2")  dir1.move\_repository(dir2)  assert dir1 in dir2.list  def test\_name\_after\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir1")  dir3 = Directory("dir1")  dir1.move\_repository(dir3)  dir2.move\_repository(dir3)  assert dir2.get\_name() == "dir1`"  def test\_delete():  dir = Directory()  dir.delete()  assert dir.list == []  assert dir.get\_name() == "None"  def test\_redelete():  dir = Directory()  dir.delete()  with pytest.raises(FileExistsError):  dir.delete() |

|  |
| --- |
| test\_logtextfile.py |
| import pytest  from directory import Directory  from logtextfile import LogTextFile  def test\_name():  log = LogTextFile()  assert log.get\_name() == "Logs.lg"  log.set\_name("New name")  assert log.get\_name() == "New name.lg"  def test\_good\_move():  dir1 = Directory("dir1")  dir2 = Directory("dir2")  log = LogTextFile()  log.move(dir2)  assert log.get\_direcrory\_name() == "dir2"  def test\_content():  log = LogTextFile()  assert log.get\_context() == "Begginning:\nLogs.lg: created"  log.append\_context(" + some")  assert log.get\_context() == "Begginning:\nLogs.lg: created + some"  def test\_delete():  dir = Directory()  log = LogTextFile()  log.move(dir)  log.delete()  assert dir.list == []  def test\_redelete():  dir = Directory()  log = LogTextFile()  log.move(dir)  log.delete()  with pytest.raises(FileExistsError):  log.delete() |

Завдання 3

|  |
| --- |
| main.py |
| from binary\_file import BinaryFile  from directory import Directory  from bufferFile import BufferFile  from logtextfile import LogTextFile  home = Directory("home")  log = LogTextFile()  log.move(home)  binary = BinaryFile(home, log, "Binn")  binaryf = BinaryFile(home, log, "Bins")  bufferfile = BufferFile(5, home, log, "")  nested\_dir = Directory("nested\_dir")  nested\_dir.move\_repository(home)  try:  while True:  command = input('Enter command: ')  command = command.split()  if command[0] == 'exit':  print("Exiting...")  break  # directory  elif command[0] == 'mkdir': # make  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: mkdir {name}")  continue  dir = Directory(command[1])  dir.move\_repository(home)  print("okmkdir")  elif command[0] == 'mvdir': # move  if len(command) != 3:  print(  "Incorrect number of args " + str(len(command)) + ", but need: mvdir {name\_of\_moving} {name\_of\_moveIn}")  continue  dirF = ""  for item in home.list:  if item.get\_name() == command[1]:  dirF = item  if dirF == "":  print("Directory " + str(command[1]) + " does not exist")  continue  dir = ""  for item in home.list:  if item.get\_name() == command[2]:  dir = item  if dir == "":  print("Directory " + str(command[2]) + " does not exist")  continue  home.delete\_directory(dirF)  dirF.move\_repository(dir)  print("okmvdir")  elif command[0] == 'rddir': # read  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: rddir {name}")  continue  if command[1] == "home":  home.print\_list()  continue  dirF = ""  for item in home.list:  if item.get\_name() == command[1]:  dirF = item  if dirF == "":  print("Directory " + str(command[1]) + " does not exist")  continue  dirF.print\_list()  print("okrddir")  elif command[0] == 'deldir': # delete  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: deldir {name}")  continue  dirF = ""  for item in home.list:  if item.get\_name() == command[1]:  dirF = item  if dirF == "":  print("Directory " + str(command[1]) + " does not exist")  continue  home.delete\_directory(dirF)  dirF.delete()  print("okdeldir")  # binary  elif command[0] == 'mkbin': # make  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: mkbin {name}")  continue  binaryfile = BinaryFile(home, log, command[1])  print("okmkbin")  elif command[0] == 'mvbin': # move  if len(command) != 3:  print(  "Incorrect number of args " + str(len(command)) + ", but need: mvbin {name\_of\_moving} {name\_of\_moveIn}")  continue  binF = ""  for item in home.list:  if item.get\_name() == command[1] + ".bin":  binF = item  if binF == "":  print("BinaryFile " + str(command[1]) + " does not exist")  continue  dir = ""  for item in home.list:  if item.get\_name() == command[2]:  dir = item  if dir == "":  print("Directory " + str(command[2]) + " does not exist")  continue  binF.move(dir)  print("okmvbin")  elif command[0] == 'rdbin': # read  if len(command) != 2:  print(  "Incorrect number of args " + str(len(command)) + ", but need: rdbin {name}")  continue  binF = ""  for item in home.list:  if item.get\_name() == command[1] + ".bin":  binF = item  if binF == "":  print("BinaryFile " + str(command[1]) + " does not exist")  continue  print(binF.get\_context())  print("okrdbin")  elif command[0] == 'delbin': # delete  if len(command) != 2:  print(  "Incorrect number of args " + str(len(command)) + ", but need: delbin {name}")  continue  binF = ""  for item in home.list:  if item.get\_name() == command[1] + ".bin":  binF = item  if binF == "":  print("BinaryFile " + str(command[1]) + " does not exist")  continue  binF.delete()  print("okdelbin")  # buff  elif command[0] == 'mkbuf': # make  if len(command) != 3:  print("Incorrect number of args " + str(len(command)) + ", but need: mkbuf {name} {size}")  continue  if not command[2].isdigit():  print("Size must be number: mkbuf {name} {size}")  continue  bufferfile = BufferFile(command[2], home, log, command[1])  print("okmkbuf")  elif command[0] == 'appqu': # add item  if len(command) != 3:  print("Incorrect number of args " + str(len(command)) + ", but need: appqu {name} {item}")  continue  bufF = ""  for item in home.list:  if item.get\_name() == command[1] + ".buf":  bufF = item  if bufF == "":  print("Buffer " + str(command[1]) + " does not exist")  continue  bufF.append\_queue(command[2])  print("okappqubuf")  elif command[0] == 'popqu': # pop item  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: popqu {name}")  continue  bufF = ""  for item in home.list:  if item.get\_name() == command[1] + ".buf":  bufF = item  if bufF == "":  print("Buffer " + str(command[1]) + " does not exist")  continue  bufF.first\_out()  print("okpopbuf")  elif command[0] == 'mvbuf': # move  if len(command) != 3:  print(  "Incorrect number of args " + str(len(command)) + ", but need: mvbuf {name\_of\_moving} {name\_of\_moveIn}")  continue  bufF = ""  for item in home.list:  if item.get\_name() == command[1] + ".buf":  bufF = item  if bufF == "":  print("Buffer " + str(command[1]) + " does not exist")  continue  dir = ""  for item in home.list:  if item.get\_name() == command[2]:  dir = item  if dir == "":  print("Directory " + str(command[2]) + " does not exist")  continue  bufF.move(dir)  print("okmvbuf")  elif command[0] == 'rdbuf': # read  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: rdbuf {name}")  continue  bufF = ""  for item in home.list:  if item.get\_name() == command[1] + ".buf":  bufF = item  if bufF == "":  print("BufferFile " + str(command[1]) + " does not exist")  continue  for i in range(0, len(bufF.get\_context())):  print(bufF.get\_context()[i])  print("okrdbuf")  elif command[0] == 'delbuf': # delete  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: delbuf {name}")  continue  bufF = ""  for item in home.list:  if item.get\_name() == command[1] + ".buf":  bufF = item  if bufF == "":  print("BufferFile " + str(command[1]) + " does not exist")  continue  bufF.delete()  print("okdelbuf")  # log  elif command[0] == 'mklog': # make  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: mklog {name}")  continue  print("Can be only 1 logger")  elif command[0] == 'mvlog': # move  if len(command) != 2:  print("Incorrect number of args " + str(len(command)) + ", but need: mvlog {name}")  continue  print("Cannot move logger")  elif command[0] == 'rdlog': # read  if len(command) != 1:  print("Incorrect number of args " + str(len(command)) + ", but need: rdlog")  continue  for i in range(0, len(log.get\_context())):  print(log.get\_context()[i])  elif command[0] == 'dellog': # delete  if len(command) != 1:  print("Incorrect number of args " + str(len(command)) + ", but need: dellpg")  continue  print("Cannot delete logger")  # ex  else:  print("Unknown command")  except EOFError as er:  print(er) |

|  |
| --- |
| tests.robot |
| \*\*\* Settings \*\*\*  Library Process  Suite Teardown Terminate All Processes kill=True  \*\*\* Test Cases \*\*\*  RunTest  ${result}= Start Process ${CURDIR}/main.py print('exit')  Log ${result.stdout}  Terminate All Processes  MkDirTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mkdir')  Log ${result.stdout}  Terminate All Processes  MkDirTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mkdir gg')  Log ${result.stdout}  Terminate All Processes  PrintDirTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('rddir home')  Log ${result.stdout}  Terminate All Processes  RdDirTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('rddir')  Log ${result.stdout}  Terminate All Processes  MvDirTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mvdir gg hh')  Log ${result.stdout}  Terminate All Processes  MvDirTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mvdir')  Log ${result.stdout}  ${result}= Start Process ${CURDIR}/main.py print('mvdir gg')  Log ${result.stdout}  Terminate All Processes  DelDirTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('deldir gg')  Log ${result.stdout}  Terminate All Processes  DelDirTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('deldir')  Log ${result.stdout}  Terminate All Processes  #bin  MkBinTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mkbin')  Log ${result.stdout}  Terminate All Processes  MkBinTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mkbin gg')  Log ${result.stdout}  Terminate All Processes  PrintBinTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('rdbin home')  Log ${result.stdout}  Terminate All Processes  RdBinTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('rdbin')  Log ${result.stdout}  Terminate All Processes  MvBinTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mvbin Binn hh')  Log ${result.stdout}  Terminate All Processes  MvBinTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mvbin')  Log ${result.stdout}  ${result}= Start Process ${CURDIR}/main.py print('mvbin Binn')  Log ${result.stdout}  Terminate All Processes  DelBinTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('delbin Binn')  Log ${result.stdout}  Terminate All Processes  DelBinTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('delbin')  Log ${result.stdout}  Terminate All Processes  #buf  MkBufTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mkbuf')  Log ${result.stdout}  Terminate All Processes  MkBufTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mkbuf Buffer')  Log ${result.stdout}  Terminate All Processes  PrintBufTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('rdbuf Buffer')  Log ${result.stdout}  Terminate All Processes  RdBufTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('rdbuf')  Log ${result.stdout}  Terminate All Processes  MvBufTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mvbuf Buffer hh')  Log ${result.stdout}  Terminate All Processes  MvBufTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mvbuf')  Log ${result.stdout}  ${result}= Start Process ${CURDIR}/main.py print('mvbuf Buffer')  Log ${result.stdout}  Terminate All Processes  AppBufTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('appqu Buffer 2')  Log ${result.stdout}  Terminate All Processes  AppBufTestIncorrect  ${result}= Start Process ${CURDIR}/main.py print('appqu Buffer')  Log ${result.stdout}  Terminate All Processes  PopBufTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('popqu Buffer')  Log ${result.stdout}  Terminate All Processes  DelBufTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('delbuf Buffer')  Log ${result.stdout}  Terminate All Processes  DelBufTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('delbuf')  Log ${result.stdout}  Terminate All Processes  #logger  MkLogTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mklog')  Log ${result.stdout}  Terminate All Processes  MkLogTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mklog logg')  Log ${result.stdout}  Terminate All Processes  PrintLogTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('rdlog logg')  Log ${result.stdout}  Terminate All Processes  RdLogTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('rdlog')  Log ${result.stdout}  Terminate All Processes  MvLogTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('mvlog Logg hh')  Log ${result.stdout}  Terminate All Processes  MvLogTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('mvlog')  Log ${result.stdout}  ${result}= Start Process ${CURDIR}/main.py print('mvlog Logg')  Log ${result.stdout}  Terminate All Processes  DelLogTestCorrect  ${result}= Start Process ${CURDIR}/main.py print('dellog Logg')  Log ${result.stdout}  Terminate All Processes  DelLogTestIncorrectValue  ${result}= Start Process ${CURDIR}/main.py print('dellog')  Log ${result.stdout}  Terminate All Processes  IncorrectCommand  ${result}= Start Process ${CURDIR}/main.py print('Something here')  Log ${result.stdout}  Terminate All Processes |

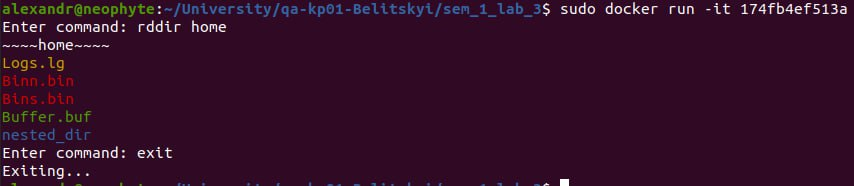
|  |
| --- |
| Dockerfile |
| FROM python:3  COPY . /sem\_1\_lab\_3  WORKDIR /sem\_1\_lab\_3  RUN python main.py  CMD [ "python3", "main.py" ] |

Результати виконання тестів

|  |
| --- |
| Завдання 2 (PyTest) |
|  |

|  |
| --- |
| Завдання 3 (Robot) |
|  |

Запуск проєкту через Docker Container



Висновки

В даній курсовій роботі було набуто навички тестування програмного забезпечення, вивчено інструменти для тестування ПЗ (Robot, Pytest), проведено ознайомлення з системою віртуалізації Docker та покращено навички володіння мовою Python. Розроблення файлової системи в пам'яті допомогло зрозуміти архітектуру та принцип роботи файлових систем, що допоможе при роботі з архітектурами файлових систем.

Виконання роботи відбувалося в операційній системі Ubuntu, що побудована на основі Linux. Це допомогло набути навичок володіння терміналом та підвищити швидкість роботи з ним.