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 “Тестування програмного забезпечення”

Виконала
студентка III курсу
групи КП-02
Кривошеєва Валерія Валеріївна
(*прізвище, ім’я, по батькові*)

Перевірив
“____” “_____” 20___ р.
викладач
Погорелов Володимир Володимирович
(*прізвище, ім’я, по батькові*)

Київ 2022

Постановка завдання

Загальні вимоги до завдання

Створити модель файлової системи:

- Directory - може містити інші директорії та файли. Директорія може бути порожньою або містити декілька елементів. Кількість елементів у директорії має бути <= DIR_MAX_ELEMS. Дозволені операції:
    - Створити директорію
    - Видалити директорію
    - Список файлів і піддиректорій
    - Перемістити файл або піддиректорію в інше місце

- Binary file - просто незмінний файл, який містить деяку інформацію. Дозволені операції:
    - Створити файл
    - Видалити файл
    - Перемістити файл
    - Прочитати файл (повертає вміст файлу)

- Log text file - текстовий файл, який можна змінювати, додаючи рядки в кінець файлу. Дозволені операції:
    - Створити файл
    - Видалити файл
    - Перемістити файл
    - Прочитати файл (повертає вміст файлу)
    - Додайти рядок у кінець файлу

- Buffer file - це особливий тип файлу, який працює як черга. Деякі потоки надсилають елементи до файлу, інші витягують елементи з файлу. Кількість елементів у файлі <= MAX_BUF_FILE_SIZE. Дозволені операції:
    - Створити файл
    - Видалити файл
    - Перемістити файл
    - Додати елемент в чергу

- Отримати елемент з черги

Розробити HTTP (Restful) додаток, який дозволить виконувати ті ж команди, використовуючи протокол HTTP. Ендпоінти мають такий вигляд:

- /directory
- /binaryfile
- /logtextfile
- /bufferfile

Створити Dockerfile для запуску HTTP додатку.

Розробити клієнт (CLI), для якого параметри HTTP запиту передаватимуться у вигляді агрументів команди запуску.

Всі можливі шляхи взаємодії з розробленим додатком покрити тестами.

Хід роботи

Для виконання даної роботи використано мову програмування Python.
На першому етапі було розроблено прототип системи у вигляді дерева для зберігання файлів й директорій. Далі написано тести для можливих випадків взаємодії, використовуючи PyTest. Після цього реалізовано необхідні методи, які зазначено у завданні.

На другому етапі було розроблено прототип HTTP додатку, що прийматиме HTTP запити різних методів. Далі написано тести для можливих випадків взаємодії, використовуючи Robot framework. Після цього реалізовано необхідні методи, які зазначено у завданні, використовуючи Flask. Загалом у додатку обробляються методи POST, PUT, PATCH, DELETE, GET. Також, передбачено повернення різних HTTP кодів результатів обробки запитів.

На третьому етапі було створено Dockerfile для запуску HTTP додатку. Також, створено клієнт (CLI), для якого параметри HTTP запиту передаються у вигляді агрументів команди запуску. Вони отримуються за допомогою sys.argv, формується запит певного вигляду та надсилається за адресою запущеного раніше HTTP додатку. Результат виконання записується в консоль. Крім того, написано тести для можливих випадків взаємодії з клієнтом, використовуючи Robot framework.

Тексти коду програм

directory.py

```
class Directory:
  def __init__(self, dirName, maxElements = 0, father = None):
      self.name = dirName
      if int(maxElements) < 0:
          raise SystemError('Dir max elems must be a positive number')
      self.DIR_MAX_ELEMS = maxElements
      self.father = father
      if father != None:
          if (self.father.DIR_MAX_ELEMS == len(self.father.children)):
              raise SystemError('Father directory ', father.name, ' is full')
          self.father.children.append(self)
      self.children = []
      print('directory ', self.name,' created!')

  def delete(self):
      if self.father != None:
          self.father.children.remove(self)
      print('directory ', self.name,' deleted!')
      return

  def listContent(self):
      return self.children

  def move(self, node, path):
      if not self.children.__contains__(node):
          raise SystemError("Directory doesn't contain entered node")
      if (len(path.children) == path.DIR_MAX_ELEMS):
          raise SystemError('Target directory ', path.name, ' is full')
      self.children.remove(node)
      node.father = path
      path.children.append(node)
      print('element ', node.name,' moved to ', path.name, '!')
      return

  def moveSelf(self, path):
      if (len(path.children) == path.DIR_MAX_ELEMS):
          raise SystemError('Target directory ', path.name, ' is full')
      if self.father != None:
          self.father.children.remove(self)
      self.father = path
      path.children.append(self)
      print('directory ', self.name,' moved to ', path.name, '!')
```

```
        return
```

binary_file.py

```
class BinaryFile:
    def __init__(self, fileName, father = None, info = None):
        self.name = fileName
        self.father = father
        self.info = info
        if father != None:
            if (self.father.DIR_MAX_ELEMS == len(self.father.children)):
                raise SystemError('Father directory ', father.name, ' is full')
            self.father.children.append(self)
        print('binary file ', self.name,' created!')

    def delete(self):
        if self.father != None:
            self.father.children.remove(self)
        print('binary file ', self.name,' deleted!')
        return

    def readFile(self):
        return self.info

    def move(self, path):
        if (len(path.children) == path.DIR_MAX_ELEMS):
            raise SystemError('Target directory ', path.name, ' is full')
        if self.father != None:
            self.father.children.remove(self)
        self.father = path
        path.children.append(self)
        print('binary file ', self.name,' moved to ', path.name, '!')
        return
```

buffer_file.py

```
class BufferFile:
    def __init__(self, fileName, maxSize = 0, father = None):
        self.name = fileName
        if int(maxSize) < 0:
            raise SystemError('Max size must be a positive number')
```

```
        self.MAX_BUF_FILE_SIZE = maxSize
        self.father = father
        self.queue = []
        if father != None:
            if (self.father.DIR_MAX_ELEMS == len(self.father.children)):
                raise SystemError('Father directory ', father.name, ' is full')
            self.father.children.append(self)
        print('buffer file ', self.name,' created!')

    def delete(self):
        if self.father != None:
            self.father.children.remove(self)
        print('buffer file ', self.name,' deleted!')
        return

    def move(self, path):
        if (len(path.children) == path.DIR_MAX_ELEMS):
            raise SystemError('Target directory ', path.name, ' is full')
        if self.father != None:
            self.father.children.remove(self)
        self.father = path
        path.children.append(self)
        print('buffer file ', self.name,' moved to ', path.name, '!')
        return

    def pushElement(self, element):
        if self.MAX_BUF_FILE_SIZE == len(self.queue):
            raise SystemError('Buffer file ', self.name, ' is full')
        self.queue.append(element)
        return

    def consumeElement(self):
        if len(self.queue) == 0:
            raise SystemError('Buffer file ', self.name, ' is empty')
        lineToReturn = self.queue[0]
        self.queue.pop(0)
        return lineToReturn
```

log_text_file.py

```
class LogTextFile:
    def __init__(self, fileName, father = None):
        self.name = fileName
        self.father = father
```

```
        self.info = ''
        if father != None:
            if (self.father.DIR_MAX_ELEMS == len(self.father.children)):
                raise SystemError('Father directory ', father.name, ' is full')
            self.father.children.append(self)
        print('log text file ', self.name,' created!')

    def delete(self):
        if self.father != None:
            self.father.children.remove(self)
        print('log text file ', self.name,' deleted!')
        return

    def readFile(self):
        return self.info

    def move(self, path):
        if (len(path.children) == path.DIR_MAX_ELEMS):
            raise SystemError('Target directory ', path.name, ' is full')
        if self.father != None:
            self.father.children.remove(self)
        self.father = path
        path.children.append(self)
        print('log text file ', self.name,' moved to ', path.name, '!')
        return

    def appendLine(self, lineToAdd):
        print(lineToAdd)
        print(self.info + lineToAdd)
        self.info = self.info + lineToAdd
        return
```

Приклад тестів з PyTest

test_binary_file.py

```
from nodes.binary_file import BinaryFile
from nodes.directory import Directory

class TestBinaryFile:

    def test_initBinaryFile(self):
        rootDirectory = Directory('root', 10)
        name = 'test_binary'
```

```
        fileInfo = 'test info for binary file!!!'
        binaryFile = BinaryFile(name, rootDirectory, fileInfo)

        assert binaryFile.name == name
        assert binaryFile.father == rootDirectory
        assert binaryFile.info == fileInfo
        assert binaryFile in rootDirectory.children

    def test_deleteBinaryFile(self):
        rootDirectory = Directory('root', 10)
        name = 'test_binary'
        fileInfo = 'test info for binary file!!!'
        binaryFile = BinaryFile(name, rootDirectory, fileInfo)

        binaryFile.delete()
        del binaryFile

        assert 'binaryFile' not in locals()
        assert not rootDirectory.children.__contains__(binaryFile)

    def test_readBinaryFile(self):
        rootDirectory = Directory('root', 10)
        name = 'test_binary'
        fileInfo = 'test info for binary file!!!'
        binaryFile = BinaryFile(name, rootDirectory, fileInfo)

        assert binaryFile.readFile() == fileInfo

    def test_moveBinaryFile(self):
        rootDirectory = Directory('root', 10)
        firstDirectory = Directory('first', 1, rootDirectory)

        name = 'test_binary'
        fileInfo = 'test info for binary file!!!'
        binaryFile = BinaryFile(name, firstDirectory, fileInfo)

        binaryFile.move(rootDirectory)

        assert not firstDirectory.children.__contains__(binaryFile)
         assert rootDirectory.children.__contains__(binaryFile)
```

Приклад тестів з Robot

BinaryFileTestSuite.robot

```
*** Settings ***
Documentation    A test suite for valid login.
...
...              Keywords are imported from the resource file
Library   Process
Library   OperatingSystem

*** Variables ***
${path}    /Users/valeria/Documents/qa/qa-kp02-kryvosheieva/lab3/cli.py

*** Test Cases ***

User can create an new binary file with valid name
   Run Process    python3    ${path}    post    directory    root    CuteAnimals    100
   ${result} =    Run Process    python3    ${path}    post    binaryfile    root/CuteAnimals
Kitten    Mrrrrr
   Should Contain    ${result.stdout}   Status code: 200

User should not be able to create a new binary file in the directory that does not exist
   ${result} =    Run Process    python3    ${path}    post    binaryfile    root/Lizzards
Kitten    Mrrrrr
   Should Contain    ${result.stdout}   Status code: 404

User should be able to find binary file
   Run Process    python3    ${path}    post    directory    root    CuteBirds    100
   ${result} =    Run Process    python3    ${path}    post    binaryfile    root/CuteBirds
Penguines    TipTop
   Should Contain    ${result.stdout}   Status code: 200

User should not be able to find binary files that do not exist
   Run Process    python3    ${path}    post    directory    root    CuteSpiders    100
   ${result} =    Run Process    python3    ${path}    get    binaryfile
path\=root/CuteSpiders/Tarantul
   Should Contain    ${result.stdout}   Status code: 404

User should be able to move file to a different location
   Run Process    python3    ${path}    post    directory    root    CuteLizzards    100
   Run Process    python3    ${path}    post    directory    root    TerribleMonsters    100
   ${result} =    Run Process    python3    ${path}    post    binaryfile
root/CuteLizzards    Varan    Shhhhh
   Should Contain    ${result.stdout}   Status code: 200

   ${result} =    Run Process    python3    ${path}    patch    binaryfile
move_from\=root/CuteLizzards/Varan    move_to\=root/TerribleMonsters
```

```
    Should Contain    ${result.stdout}    Status code: 200

    ${result} =    Run Process    python3    ${path}    get    binaryfile
path\=root/CuteLizzards/Varan
    Should Contain    ${result.stdout}    Status code: 404

    ${result} =    Run Process    python3    ${path}    get    binaryfile
path\=root/TerribleMonsters/Varan
    Should Contain    ${result.stdout}    Status code: 200

User should be able to delete an existing file
    ${result} =    Run Process    python3    ${path}    post    binaryfile    root    Doggo
Bark
    Should Contain    ${result.stdout}    Status code: 200

    ${result} =    Run Process    python3    ${path}    get    binaryfile    path\=root/Doggo
    Should Contain    ${result.stdout}    Status code: 200

    ${result} =    Run Process    python3    ${path}    delete    binaryfile
path\=root/Doggo
    Should Contain    ${result.stdout}    Status code: 200

    ${result} =    Run Process    python3    ${path}    get    binaryfile    path\=root/Doggo
    Should Contain    ${result.stdout}    Status code: 404
```

З повним кодом розробленого додатку можна ознайомитись у репозиторії на GitHub https://github.com/my-might/qa-kp02-kryvosheieva.

Висновки

Під час виконання даної роботи я навчилась писати тести, використовуючи мову програмування Python з допомогою PyTest та Robot. Крім того, закріпила знання з того, як розроблювати HTTP додатки мовою Python, використовуючи Flask.

Також, закріпила знання з написання Dockerfile для додатків. Навчилась працювати з агрументами командного рядка у Python.