Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка та захист віддаленого сервера для інтерпретації Python коду

Виконав: студент IV курсу, групи СБс-42
спеціальності 125 Кібербезпека
(шифр і назва спеціальності)

_____ Зелінський А.О.
(підпис) (прізвище та ініціали)

Керівник _____ Скоренький Ю.Л.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Кареліна О.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Загородна Н.В.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль-2021

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Зелінському Андрію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та захист віддаленого сервера для інтерпретації Python коду

Керівник роботи Скоренький Юрій Любомирович, кандидат фізико-математичних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «__» _____ 2021 року № _____

2. Термін подання студентом завершеної роботи 2021р.

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Постановка завдання та аналіз передметної області.
1.1 Аналіз предметної області. 1.1.1 Аналіз існуючих рішень 1.1.2 Аналіз хмарних обчислень.
1.1.3 Аналіз середовищ розробки. 1.1.4 Аналіз вразливостей клієнт-серверної архітектури.
1.2 Постановка задачі. 1.2.1 Характеристика об'єкту проектування та постановка задачі.
1.2.2 Дерево проблем. 1.2.3 Дерево цілей. 1.2.4 Аналіз і вибір методів та алгоритмів
розв'язання задачі. 1.3 Висновок до першого розділу. 2 Проектування та тестування
клієнт-серверної архітектури. 2.1 Спрощена модель системи. 2.2 Проектування
клієнт-серверної архітектури. 2.3 Розробка та опис програмних рішень.
2.3.1 Діаграма компонентів. 2.3.2 Опис використаних сторонніх бібліотек і модулів.
2.3.3 Розробка та опис програмних модулів. 2.3.4 Розробка та опис інтерфейсу користувача.
2.3.5 Опис проблем, які виникали під час розробки. 2.4 Системи захисту серверної частини.
2.4.1 Захист каналу зв'язку між клієнтом та сервером. 2.4.2 Захист мережі від DoS атак
2.4.3 Захист файлової системи сервера. 2.5 Тестування захисту розробленої
клієнт-серверної архітектури. 2.6 Висновок до другого розділу. 3 Безпека життєдіяльності,
основи охорони праці. 3.1 Долікарська допомога при вивихах. 3.2 Соціальне значення
охорони праці. 3.3 Загальні вимоги безпеки з охорони праці для користувачів ПК.
3.4 Висновки до третього розділу. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу(з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи		*Виконано*
2.	Аналіз об`єкту проектування та постановка задачі		*Виконано*
3.	Огляд існуючих рішень		*Виконано*
4.	Виконання дослідження щодо вимог до застосунку		*Виконано*
5.	Розробка та захист клієнт серверної архітектури		*Виконано*
6.	Оформлення розділу «Постановка завдання та		*Виконано*
	аналіз передметної області»		
7.	Оформлення розділу «Проектування та		*Виконано*
	тестування клієнт серверної архітектури»		
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»		*Виконано*
9.	Виконання завдання до підрозділу «Основи охорони		*Виконано*
	праці»		
10.	Оформлення кваліфікаційної роботи		*Виконано*
11.	Нормоконтроль		*Виконано*
12.	Перевірка на плагіат		*Виконано*
13.	Попередній захист кваліфікаційної роботи		*Виконано*
14.	Захист кваліфікаційної роботи		

Студент _____ Зелінський А.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Скоренький Ю. Л.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка та захист віддаленого сервера для інтерпретації Python коду // Кваліфікаційна робота освітнього рівня «Бакалавр» // Зелінський Андрій Олегович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБс-42 // Тернопіль, 2021 // С. 80, рис. – 21 , табл. – 0, кресл. – 0, додат. – 18, бібліогр. – 25.

Ключові слова: Python, flutter, фреймворк, API, dart, сервер, клієнт, скрипт, додаток.

У кваліфікаційній роботі розроблено програмне рішення для віддаленої інтерпретації програм, написаних мовою Python.

Запрограмований у роботі сервіс нададає користувачам можливість інтерпретувати програми, написані мовою Python, з мобільних пристроїв на платформі iOS та Android. Реалізація програм виконується в ізольованому контейнері хмарного середовища. Користувач може завантажувати, видаляти, а також запускати на виконнання файли на севері. Результати виконання програми надходять користувачеві у режимі реального часу. Сервіс підтримує автоматичне встановлення пакетів, необхідних для роботи програми.

Проведено огляд літератури, проаналізовано існуючі сервіси віддаленої інтерпретації програм та обрано оптимальну стратегію для розробки. У процесі виконання системного аналізу були обрані ефективні методи, алгоритми та засоби для сервісу, що проектувався.

ANNOTATION

Remote server development and protection for interpretation of Python code // Qualification work of Bachelor educational degree // Zelinskyi Andrii // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Cybersecurity Department, SBs-42 group // Ternopil, 2021 // Pages – 80, figures – 21, tables – 0, sketches – 0, addendums – 18, references – 25.

Keywords: Python, flutter, фреймворк, API, dart, server, client, script, application.

In the qualification work developed the software solution for remote interpretation of the programs written in Python language.

The programmed service gives users the opportunity to interpret applications written in Python from mobile devices on the iOS or Android platforms. Implementation of programs is performed in an isolated container of the cloud environment. The user can download, delete, and run files in the server. The results of the program are delivered to the user in real time. The service supports automatic installation of packages required for the program.

Provided review of the literature, analyzed existing services for remote interpretation of programs and selected the optimal strategy for development. In the process of performing system analysis, effective methods, algorithms and tools for the designed service were selected.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скрипт – Програма або програмний файл сценарію, які автоматизують деяку задачу, яку користувач робив би вручну, використовуючи інтерфейс програми.

Фреймворк – Абстракція, в якій програмне забезпечення, що надає генеральні функціональні можливості, може бути вибірково змінено додатковим кодуванням, написаним користувачем, таким чином забезпечуючи програмне забезпечення для програми.

IDE (Integrated development environment) – Інтегроване середовище розробки — програмне забезпечення для розробки програмних продуктів. Зазвичай включає редактор коду, інструменти для автоматизації складання та відлагодження програм.

API (Application Programming Interface) – програмне забезпечення для розробки програмних продуктів. Зазвичай включає редактор коду, інструменти для автоматизації складання та відлагодження програм.

ERD (Entity-relationship diagram) – діаграма «сутність-зв’язок». Модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.

ER-модель – це мета-модель даних, тобто засіб опису моделей даних.

ОС – Операційна система – це базовий комплекс програм, що виконує керування апаратною складовою комп'ютера або віртуальної машини, забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем.

Експлоіт – це комп'ютерна програма, фрагмент програмного коду або послідовність команд, що використовують вразливості в програмному забезпеченні та призначені для проведення атаки на обчислювальну систему.

ЗМІСТ

ВСТУП

Актуальність теми. На сьогодні мова програмування Python є однією з передових і найпопулярніших у сфері розробки програмного забезпечення. Так за рейтингом TIOBE, на сьогоднішній день Python стала другою за популярністю мовою програмування в світі, обійшовши навіть такі відомі мови як C++, C# і Java [1]. Вона використовується в багатьох проектах та в різних якостях: і як основна мова програмування, і як мова для створення розширень та інтеграції додатків. Мовою Python реалізована велика кількість програмних продуктів, також вона активно використовується для створення прототипів майбутніх програм.

Сфера застосування мови програмування Python включає (але не обмежується): веб-розробку, розробку ігор, машинне навчання та штучний інтелект, опрацювання та візуалізацію даних, а також програми для Веб-скрапінгу [2].

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є проектування та реалізація програмного рішення для віддаленої інтерпретації, захисту та виконання програм, написаних мовою Python. Для досягнення цієї мети можна виокремити такі основні чотири завдання:

– Дослідити доступні програмні інструменти для розробки мобільного застосунку.

– Аналіз вже існуючих сервісів для віддаленої інтерпретації з метою вибору оптимального рішення.

– Розробка методів та алгоритмів вирішення задачі віддаленої інтерпретації програм.

– Вибір потрібних засобів для реалізації завдання.

1 ПОСТАНОВКА ЗАВДАННЯ ТА АНАЛІЗ ПЕРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

1.1.1 Аналіз існуючих рішень

Хмарні обчислення – модель забезпечення повсюдного та зручного доступу через Інтернет мережу до пулу обчислювальних ресурсів, що підлягають налаштуванню (наприклад, до комунікаційних мереж, серверів, засобів збереження даних, прикладних програм та сервісів тощо), які можуть оперативно надаватися користувачеві та бути звільнені з мінімальними управлінськими затратами та зверненнями до провайдера.

Майже всі розробники програмного забезпечення рано чи пізно стикаються з необхідністю запустити або швидко перевірити програмний код. Розглянемо декілька найбільш використовуваних хмарних сервісів, які забезпечують віддалене виконання чи компіляцію програмного коду.

Repl.it – це інтегроване середовище розробки ПЗ для більш, ніж 50 мов програмування. Серед його переваг: зручний інтерфейс на компютері, можливість імпорту/експорту проекту, можливість встановлення додаткових пакетів (але її потрібно робити вручну), інтегрований дебагер [3] . Приклад інтерфейсу зображено на рисунку 1.1.

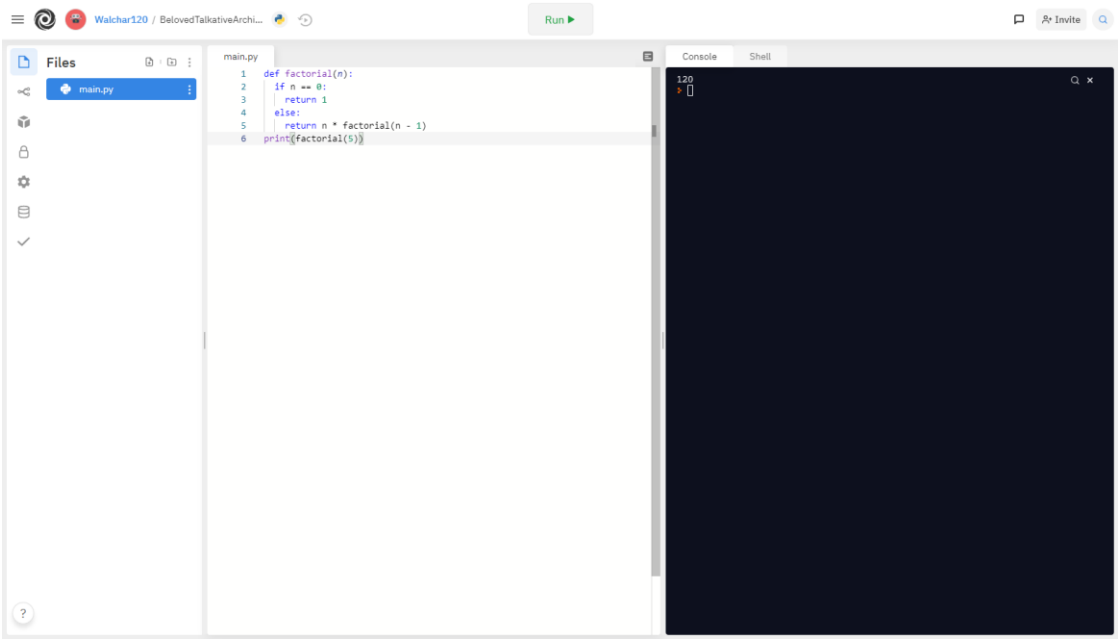

Рисунок 1.1 – Інтерфейс Repl.it

Основним недоліком цього серісу є те, що його дуже не зручно використовувати з телефону, оскільки його інтерфейс зовсім не розрахований на мобільні пристрої. Скріншот інтерфейсу з телефона зображео на рисунку 1.2

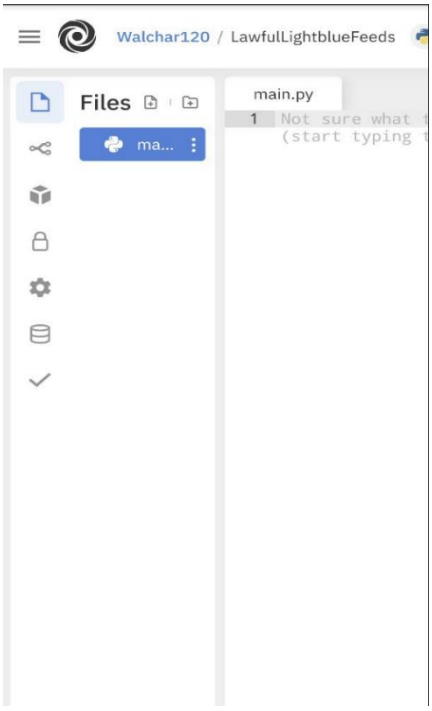

Рисунок 1.2 – Інтерфейс Repl.it на телефоні

Codepad – це мінімалістичний сервіс, в якому можна зберігати програмний код, ділитися ним і запускати з подальшим виведенням результатів його виконання. На вибір надається декілька найбільш поширених мов, але, на жаль, без вибору конкретних версій інтерпретаторів або компіляторів. Головною його перевагою є простота і легкість: сервіс буде швидко працювати навіть при повільному Інтернеті. Передбачено автопідключення стандартних пакетів, а також інтеграція з Vim або Emacs [4].

Інтерфейс сервісу можна переглянути на рисунку 1.3.

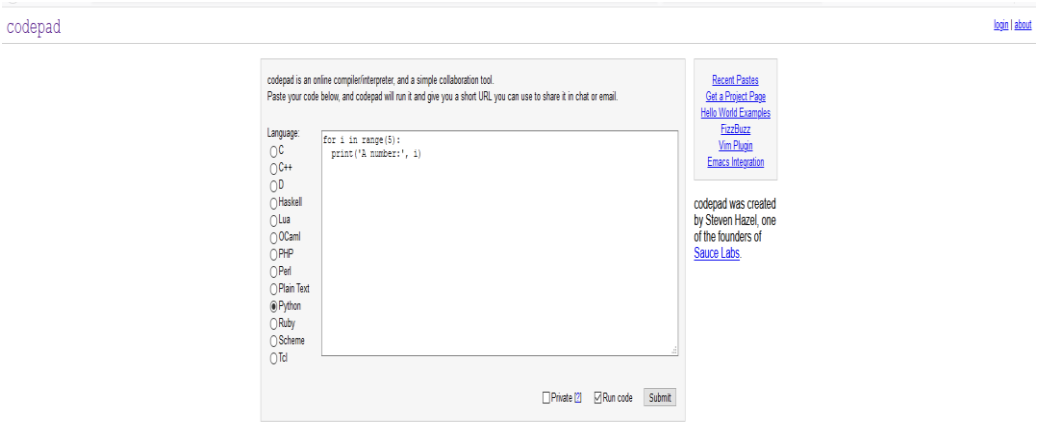

Рисунок 1.3 – Інтерфейс Codepad

Ideone – це онлайн компілятор коду, а також інструмент налагодження, який дає змогу безпосередньо в браузері виконати код на більш, ніж 60 мовах програмування (останні версії). Він досить добре підходить для роботи з телефону, можна легко редагувати код, запускати програму, вводити вхідні дані, вибирати обмеження по часу для програми. Проте він не підтримує складні проекти, тобто не вдасться встановити додаткові пакети або завантажити проект на декілька файлів [5]. Інтерфейс зображений на рисунку 1.4.

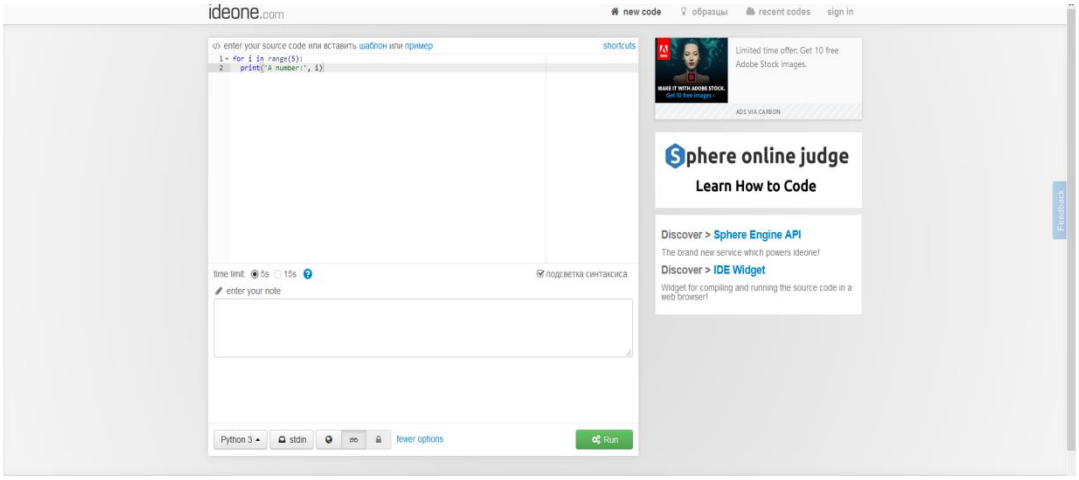

Рисунок 1.4 – Інтерфейс Ideone

Jdoodle – цей сервіс підтримує безліч мов програмуванні, в тому числі і тих, які не вдасться знайти в інших онлайн-компіляторах. Приємною особливістю JDoodle є можливість спільної роботи - можна відправити посилання на свою поточну сесію, що надасть змогу всім хто отримав посилання підключитись до написання коду та вносити свої зміни. Також в даного сервісу є власний програмний інтерфейс (API), який зможуть використати програмісти для виконання програм через API. До його слабких сторін можна віднести те, що він не підтримує компіляції складних проектів, імпорту сторонніх бібліотек, на телефоні незручний у використанні. [6] Інтерфейс з комп'ютера відображений на рисунку 1.5. Інтерфейс з телефону на рисунку 1.6

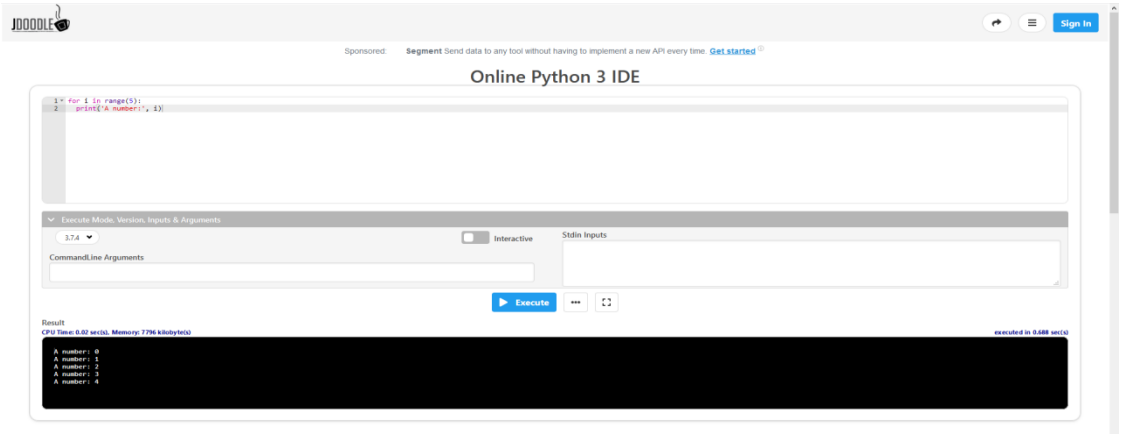

Рисунок 1.5 – Інтерфейс jdoodle

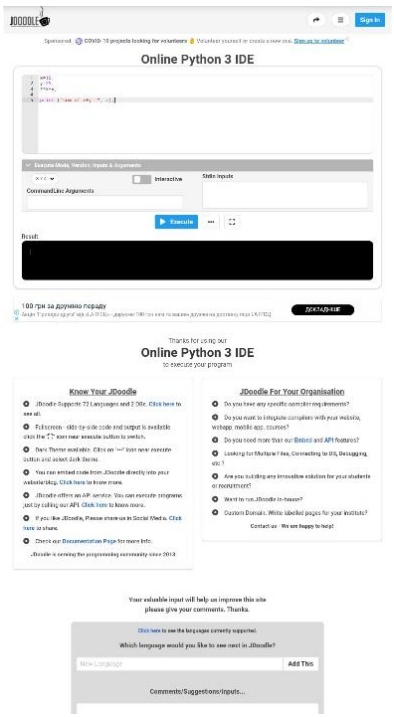

Рисунок 1.6 – Інтерфейс jdoodle на телефоні

1.1.2 Аналіз хмарних обчислень

Хмарні обчислення – це надання на замовлення IT-ресурсів через Інтернет із оплатою через передоплату. Замість того, щоб купувати, володіти та підтримувати фізичні центри обробки даних та сервери, можна отримати доступ до технологічних служб, таких як обчислювальна потужність, сховища та бази даних, за необхідності від хмарного постачальника послуг [14].

Організації будь-якого типу, розміру та галузі використовують хмарні сервіси для найрізноманітніших випадків використання, таких як резервне копіювання даних, електронна пошта, віртуальні настільні комп’ютери, розробка та тестування програмного забезпечення, аналітика великих даних та веб-додатків. Наприклад, медичні компанії використовують хмару для розробки більш персоналізованих методів лікування пацієнтів. Компанії, що надають фінансові послуги, використовують хмару для розкриття та запобігання шахрайству в реальному часі. А виробники відеоігор

використовують хмару, щоб доставляти онлайн-ігри мільйонам гравців у всьому світі.

Переваги хмарних обчислень:

– Швидкість роботи.

  o Хмара надає легкий доступ до широкого спектру технологій, щоб надати можливість швидше впроваджувати інновації. Ви можете швидко нарощувати ресурси за потребою починаючи від інфраструктурних служб, таких як обчислення, зберігання та бази даних, до Інтернету речей, машинного навчання, опрацювання даних та аналітики.

  o Можна розгорнути технологічні послуги за лічені хвилини і перейти від ідеї до впровадження. Це дає свободу для експериментів, випробовування нових ідей, щоб диференціювати досвід клієнтів та трансформувати бізнес.

– Гнучкість.

  o За допомогою хмарних обчислень не потрібно заздалегідь надмірно забезпечувати ресурси, щоб обробляти пікові рівні активності в майбутньому. Натомість можна використовувати лише необхідну кількість ресурсів.

  o Можна масштабувати ці ресурси вгору або вниз, щоб миттєво зростати та зменшувати потенціал по мірі зміни активності.

– Економія витрат.

  o Хмара дозволяє вибирати центри обробки даних та фізичні сервери для того щоб варіювати витрати і платити за ресурси лише тоді, коли вони використовуються.

Існує три основні типи хмарних обчислень: інфраструктура як послуга, платформа як послуга та програмне забезпечення як послуга. Кожен тип хмарних обчислень забезпечує різні рівні контролю, гнучкості та управління, щоб можна було вибрати правильний набір послуг для своїх потреб.

Існує три основні типи хмарних обчислень: інфраструктура як послуга, платформа як послуга та програмне забезпечення як послуга. Кожен тип хмарних обчислень забезпечує різні рівні контролю, гнучкості та управління, щоб можна було вибрати правильний набір послуг для своїх потреб.

Інфраструктура як послуга (IaaS) – містить основні будівельні блоки для хмарних систем. Зазвичай при використанні цього типу забезпечується доступ до мережевих функцій, комп'ютерів (віртуального або спеціального обладнання) та місця для зберігання даних. IaaS надає найвищий рівень гнучкості та управління управлінням вашими IT-ресурсами.

Платформа як послуга (PaaS) – знімає необхідність в управлінні базовою інфраструктурою (як правило, апаратними та операційними системами) та дозволяє зосередитись на розгортанні та управлінні програмами. Це допомагає бути більш ефективними, оскільки не потрібно турбуватися про закупівлю ресурсів, планування потужностей, технічне обслуговування програмного забезпечення та виправлення помилок.

Програмне забезпечення як послуга (SaaS) – надає повний продукт, яким керує постачальник послуг. У більшості випадків люди, які посилаються на SaaS, посилаються на додатки для кінцевих користувачів (наприклад, веб-електронна пошта). За допомогою SaaS не потрібно думати про те, як підтримується послуга або як керується базовою інфраструктурою.

Основними постачальниками хмарних обчислень виступають:

– Amazon Web Services (AWS).
– Microsoft Azure.
– Google Cloud.
– SalesForce
– IBM Cloud.
– Oracle Cloud.

Amazon Web Services (AWS) – це гігант, який дозволяє компаніям по всьому світу розвивати свою цифрову інфраструктуру повністю або частково за допомогою хмари. 2018 рік став роком повного верховенства для AWS. У 2019 році їх зростання сповільнилося, але їм все ж вдалося залишатися на вершині лідерів на ринку. Оскільки AWS - найстаріший постачальник хмарних послуг, вони пропонують безліч послуг. Для компаній, які вже використовують локальні центри обробки даних, Amazon нещодавно запустила шлюз AWS Storage Gateway. Це дає можливість клієнтам підключити наявні сховища та резервні копії до хмари за допомогою однієї консолі для доступу до конфігурацій пам'яті. AWS CloudFormation - це ще одна технологія Amazon, яка дозволяє компаніям контролювати ресурси та стеки додатків, необхідні для мобільних та веб-додатків. Клієнти отримують доступ до інфраструктурних компонентів хмарного сервісу з центральним інтерфейсом командного рядка для управління ними. Такі технологічні досягнення допомагають Amazon зарекомендувати себе світовим лідером хмарних сервісів протягом майже десятиліття. Amazon також формує стратегічні відносини з іншими технологічними фірмами, які допомогли в її глобальному розширенні. Його співпраця з компанією SaaS Apptio є яскравим прикладом, який дозволяє клієнтам приймати розумні рішення щодо переходу на AWS. Якщо говорити про свою частку ринку, то в третьому кварталі 2019 року компанія повідомила про дохід у 9 мільярдів доларів [15].

Microsoft Azure – є найближчим конкурентом хмарного короля AWS, і він стає ще ближчим. Microsoft наголошує на великій кількості нововведень для надання послуг. Як і AWS, вони також мають величезну кількість сервісів, що включає хмарні послуги на Office 365, бізнес-підписку на Dynamic 365 та LinkedIn. Це ускладнює порівняння їх послуг безпосередньо з конкурентами або іншими постачальниками хмарних послуг. Однією з корисних особливостей хмарних рішень Microsoft є Служба машинного навчання Microsoft Azure. Користувачі можуть розробляти, тестувати, розгортати,

керувати, переміщувати та навіть контролювати свою модель машинного навчання у хмарному середовищі за допомогою цієї послуги. Нещодавнє опитування Goldman Sachs серед 100 і більше CIO провідних міжнародних фірм повідомляє, що Microsoft Azure набула більшої популярності, ніж послуги AWS Amazon. Зв'язок численних постачальників у світі допоміг Microsoft привернути більше клієнтів до хмари. Безліч великих гігантів співпрацювало з Azure, щоб перемістити значну частину своїх операцій у хмару. Одним з відомих прикладів є п'ятирічне партнерство між Microsoft та Disney. У рамках цього партнерства будуть розроблені нові методи переміщення контенту виробництва в хмару. Останні технологічні розробки та співпраця з відомими організаціями в усьому світі показують потенціал Microsoft. У 2019 році темпи зростання Azure склали вісімдесят дев'ять відсотків. Він також розширив свою частку ринку та мав значний стрибок у доході [15].

Google Cloud – У 2018 році платформа Google Cloud була занадто новою, щоб стати конкурентом у списку кращих хмарних постачальників послуг. У квітні 2019 року GCP набрав темп з Google Next. Google Cloud Platform та VMware оголосили про партнерство в липні 2019 року. Це була одна з найбільших співпраць. Приблизно в той же час Google повідомив про середній дохід у 8 мільярдів доларів на рік через хмарний бізнес. Цей звіт заслуговує на увагу, оскільки лише два місяці тому компанія повідомила про середньорічний дохід у розмірі 4 мільярдів доларів на рік. Компанія регулярно вносила кілька змін, щоб відзначити себе як гігант корпоративних хмарних обчислень. Однією з таких змін було спрощення умов контракту, а також сертифікати на відповідність. Нещодавно компанія придбала платформу Looker, яка пропонує аналітичні рішення хмарним користувачам. GCP пропонує найдешевші хмарні послуги на ринку. За даними аналітиків Deutsche Bank, очікується, що хмарний сервіс до 2025 року досягне позначки в 38 мільярдів доларів щорічних продажів [15].

SalesForce – Очікується, що найбільший гравець на хмарному ринку Salesforce принесе дохід у розмірі 17 мільярдів доларів у 2020 році. Цей прогноз за допомогою інтерактивної платформи даних та аналізу Trefis також повідомив, що це зростання на 28% порівняно з минулим роком. Той факт, що більш ніж 20 найбільших банків з Європи та США використовують SalesForce, також відображає його позицію на хмарному риноку. Одним із фактів, що призвели до його успіху, є впровадження передових технологій у своїй CRM-платформі.

Клієнти SalesForce CRM можуть розраховувати на штучний інтелект, щоб визнати можливості продажу, що, в свою чергу, призводить до отримання більших доходів. Ще однією причиною його величезного зростання є рання перевага переходу. В останні кілька років SalesForce придбала численні фірми для розширення своєї екосистеми [15].

IBM Cloud – IBM повідомила про дохід у 5,3 мільярда доларів у III кварталі 2019 року, що на 6,4 відсотка більше, ніж у попередньому кварталі. Деякі функції, які привабили бізнес, включають: відкриті технології, що забезпечують взаємодію, інтегрована гібридна хмара, яка дозволяє підприємствам розблокувати гібридні дані, потужна система аналітики, розробки для підприємств. Найближчим часом IBM планує придбати невикористану область гібридних хмарних рішень. Нещодавно він придбав Red Hat, який, як очікується, допоможе ефективно поставляти послуги користувачам [15].

Oracle Cloud – У порівнянні з іншими гравцями, Oracle Cloud відстає. Щоб підтримувати свою присутність у хмарній гонці, він використовує велику частку свого програмного забезпечення в центрах обробки даних існуючих клієнтів. Її зв'язок із Microsoft у постачанні хмарних сервісів також виявився вигідним. Компанія MarketsandMarkets прогнозувала, що ринок хмарних обчислень сягне 623,3 мільярда доларів у 2023 році. Щоб отримати

найбільший фрагмент, кожному гравцеві потрібно більше акцентувати увагу на унікальних технологіях та стратегічних партнерських відносинах.

1.1.3 Аналіз середовищ розробки

Інтегроване середовище розробки Pycharm – це гібридна платформа, розроблена JetBrains як IDE для Python. Вона підтримує обидві версії Python. PyCharm можна запускати в Windows, Linux або Mac OS. Крім того, він містить модулі та пакети, які допомагають програмістам розробити програмне забезпечення, використовуючи Python за менший час та з мінімальними зусиллями. Крім того, він також може бути налаштований відповідно до вимог розробників [16].

Можливості PyCharm:

– Інтелектуальний редактор коду. Він має кольорові схеми для ключових слів, класів та функцій. Це допомагає збільшити читабельність та розуміння коду, також допомагає легко шукати помилки.

– Навігація по коду. За допомогою навігації по коду розробник може легко перейти до функції, класу чи файлу. Програміст може знайти елемент, символ або змінну у вихідному коді протягом короткого часу. Крім того, використовуючи режим об'єктива, розробник може ретельно перевірити і налагодити весь вихідний код.

– Рефакторинг. Рефакторинг в PyCharm дозволяє розробникам вдосконалити внутрішню структуру, не змінюючи зовнішню логіку коду. Також IDE допомагає розділити більш великі класи та функції.

– Підтримка багатьох веб-технологій. Це допомагає розробникам створювати веб-додатки в Python. Він підтримує популярні веб-технології, такі як HTML, CSS та JavaScript. Розробники мають змогу редагувати код веб-додатків в реальному часі за допомогою цього IDE. PyCharm також підтримує AnglularJS та NodeJS для розробки веб-додатків. Також PyCharm підтримує

популярні веб фреймворки, такі як Django. Він надає функцію автозаповнення та пропозиції щодо параметрів Django.

– Підтримка наукових бібліотек мови Python. PyCharm підтримує наукові бібліотеки Python, такі як Matplotlib, NumPy та Anaconda. Ці наукові бібліотеки допомагають будувати проекти наукових даних та машинного навчання. Він складається з інтерактивних графіків, які допомагають розробникам візуалізувати дані. Pycharm може інтегруватися з різними інструментами, такими як IPython, Django та Pytest.

Інтегроване середовище розробки Android Studio – офіційне середовище розробки для Android, яке пропонує Google. Воно пропонує інструменти, призначені для розробників Android, включаючи багате редагування коду, налагодження, тестування та профілювання інструментів.

Android Studio має такі функції:

– Інтелектуальний редактор коду. Редактор коду допомагає писати кращий код, працювати швидше та бути більш продуктивним, пропонуючи вдосконалене завершення коду, рефакторинг та аналіз коду. Під час друку Android Studio надає пропозиції у випадаючому списку. Для того щоб вставити код необхідно настиснути Tab.

– Швидкий та багатофункціональний емулятор. Емулятор Android встановлює та запускає програми швидше, ніж реальний пристрій, і дозволяє прототипувати та тестувати додаток на різних конфігураціях пристроїв Android: телефонів, планшетів, Android Wear та Android TV пристроїв. Також можна імітувати різні апаратні функції, такі як GPS-розташування, затримка мережі, датчики руху та багатокористувацький вхід.

– Багатий редактор макетів. Він дозволяє перетягувати компоненти інтерфейсу користувача, переглядаючи макети на кількох екранах. Попередній перегляд оновлюється миттєво під час зміни в редакторі макетів.

– Миттєвий запуск. Функція миттєвого запуску Android Studio надсилає зміни коду та ресурсів до запущеного додатка. Він розумно оновлює

зміни та часто доставляє їх, не перезавантажуючи додаток і не створючи нові APK файли, тож ефект від зміни коду видно відразу.

– Налаштування побудови проекту. Структура проектів Android Studio та побудова на основі Gradle забезпечують гнучкість, необхідну для створення APK-файлів для всіх типів пристроїв

– Надійна і гнучка система складання. Android Studio пропонує автоматизацію побудови, управління залежностями та настроювані конфігурації збірки. Можна налаштувати свій проект так, щоб він включав локальні та віддалені бібліотеки, і визначав варіанти збірки, які включають різні частини коду та ресурси, а також застосовувати різні скорочення коду та конфігурації підпису програм.

– Оптимізації для всіх пристроїв Android. Android Studio забезпечує єдине середовище, де можна створювати додатки для телефонів, планшетів, Android Wear, Android TV та Android Auto. Структуровані модулі коду дозволяють розділити проект на одиниці функціональності, які можна складати, тестувати та налагоджувати.

– Інструменти для тестування. Android Studio пропонує широкі інструменти, які допоможуть протестувати додатки для Android на JUnit 4 та функціональних тестових структурах інтерфейсу користувача. За допомогою тестового рекордера Espresso можна генерувати тестовий код для користувацького інтерфейсу, записуючи взаємодію з додатком на пристрої чи емуляторі. Можна запустити свої тести на пристрої, емуляторі, середовищі безперервної інтеграції або в тестовій лабораторії Firebase.

– Підтримка C ++ та NDK. Android Studio повністю підтримує редагування файлів проекту C / C ++, щоб ви могли швидко створювати компоненти JNI у додатку. IDE забезпечує підсвічування синтаксису та рефакторинг для C / C ++ та налагоджувач на основі LLDB, що дозволяє одночасно налагоджувати код Java та C / C ++. Інструменти збирання також

можуть виконувати ваші сценарії CMake та ndk-build без будь-яких модифікацій, а потім додавати спільні об'єкти до APK.

1.1.4 Аналіз вразливостей клієнт-серверної архітектури

На сьогоднішній день дуже багато пристроїв, серверів, великих хмарних сервісів мають безліч вразливостей про які ніхто навіть не здогадується. Навіть такі IT гіганти як Google, Intel, IBM, Amazon щороку оновлюють свої системи захисту як і своїх ресурсів так і конфіденційної інформації по користувачам. Якщо переглянути статистику, то тільки за 2020 рік було розроблено безліч експлоітів:

– Aruba ClearPass Policy Manager – неавтентифіковане віддалене виконання команд.

– Microsoft SQL Server Reporting Services 2016 – віддалене виконання коду.

– Sony IPELA Network Camera 1.82.01 – віддалене переповнення буфера стека.

– Adtec Digital Multiple Products – жорстко закодовані права доступу, віддалений root доступ

– GoAhead Web Server 5.1.1 – дайджест аутентифікація, захоплення, повторне використання.

Також за минулий рік були массові взломи веб сайтів на сервері Inter-Telecom, знайдені вразливості в Xiaomi Mi Temperature and Humidity Monitor, знайдені вразливості в плагінах WordPress, витік даних Monobank, Facebook і це тільки маленька частина опублікованої інформації, а неопублікованих взломів в рази більше.

Отож питання безпеки сервісу стоїть майже на першому місці. З основних вразливостей які потрібно покрити це:

– захист передачі даних між сервером та клієнтом;

– захист від DoS/DDoS атак на сервер;

– захист від несанкціонованого доступу до файлів системи на якій розміщений сервер.

– захист від запуску небажаної активності на сервері

Для забезпечення захисту передачі даних потрібно розробити систему передачі даних та використати для неї захищені протоколи передачі інформації в інтернеті відповідно до потреб сервера.

Захист від DoS/DDoS атак можна реаалізувати декількома способами.

Один із способів це використання додаткових серверів, які використовуючи свої проксі сервера будуть фільтрувати трафік. Такі сервіси дуже ефективні у використанні та можуть подолати дуже потужні DDoS атаки. Більшість хмарних сервісів мають інтегровані системи фільтрації трафіка. Для прикладу розміщений веб-сервер в хмарному сервісі Azure буде захищений від DDoS засобами самого Azure без додаткових конфігурацій та налаштувань зі сторони користувача сервісом.

Другий спосіб це налаштування маршрутизатора для фільтрації трафіку і відсіювання так званих "ботів". Цей метод ефективний проти слабких DoS атак, проте захистити від DDoS атаки один маршрутизатор не зможе.

Для захисту системи сервера від отримання доступу до файлів потрібно дослідити методи ізолювання запуску програм користувача без доступу до головної системи. На ринку присутня велика кількість різних сервісів, методів та підходів ізоляції виконання, починаючи від методів самого Python і закінчюючи методами самої операційної системи.

1.2 Постановка задачі

1.2.1 Характеристика об’єкту проектування та постановка задачі

Завданням бакалаврської кваліфікаційної роботи є розробка програмного рішення для віддаленої інтерпретації програм, написаних мовою Python [7]. Сервіс призначений для оперативного та зручного використання мови Python на будь-якому пристрої, оскільки надає можливості швидкої інтерпретації програмного коду, написаного не тільки з використанням стандартних python-бібліотек, але й інших бібліотек, які можна завантажити за допомогою системи керування програмними пакетами - pip.

При виконанні бакалаврської роботи, я поставив перед собою такі задачі:

– реалізувати сервіс, який зможе швидко та ефективно інерпретувати програми написані мовою Python;

– забезпечити автоматичне встановлення залежностей для клієнтських програм за допомогою пакетного менеджера pip;

– реалізувати виконання клієнтського коду в ізольованому контейнері;

– розробити додаток, який дасть змогу наочно та зручно корстуватись сервісом з мобільного пристрою;

– забезпечити збереження завантажених клієнтом файлів та залежностей в сховищі сервісу для подальшого використання користувачем.

Для функціонування серверного програмного рішення необхідні:

– сервер з встановленою UNIX- подібною операційною системою;

– доступ до мережі Інтернет.

Для функціонування клієнтської частини програмного рішення необхідні:

– телефон з операційною системою iOS або Android;

– доступ до мережі Інтернет.

Функціональні вимоги:

– сервер повинен інтерпретувати отримані програми користувачів та повертати їм результати опрацювання;

– сервер повинен надавати змогу завантажити та виконати готовий виконачий файл;

– сервер повинен автоматично встановлювати додаткові модулі, які необхідні для виконання клієнтської програми;

– сервер повинен забезпечити ізольованість виконання програм різних користувачів;

– клінтський додаток повинен мати інтерфейс для завантаження файлу на сервер;

– клієнтський додаток повиненен відображати результати віддаленого виконання програми .

Нефункціональні вимоги:

– зручний користувацький інтерфейс;

– можливість в реальному часі бачити результати виконання програми.

Для роботи користувача з програмою достатньо:

– запустити клієнтський додаток;робота клієнта на операційних системах iOS та Android;

– обрати файл для виконання;

– мати вихід в до мережі Інтернет.

Вимоги до складу і параметрів технічних засобів:

– для використання продукту потрібно мати мобільний телефон на базі Android версії 4.1 або вище.

1.2.2 Дерево проблем

Аналіз проблеми, об'єктивності та стратегії є одним із інструментів спільного визначення основних проблем, а також їх причин та наслідків.

На схемі рисунок 1.7 зображено дві основні причнни виникнення проблеми складності запуску Python програм на мобільних пристроях [8], що випливають з ряду інших причин – “Обмеженість ресурсів на мобільних пристроях” та “Відсутність доступу до стаціонарних робочих станцій”.

Ці причнини обумовлені неможливістю забезпечити всі переваги стаціонарних робочих станцій при збереженні малих розмірів мобільних пристрої та сучасним темпом життя, який спонукає людей до постійного переміщення.

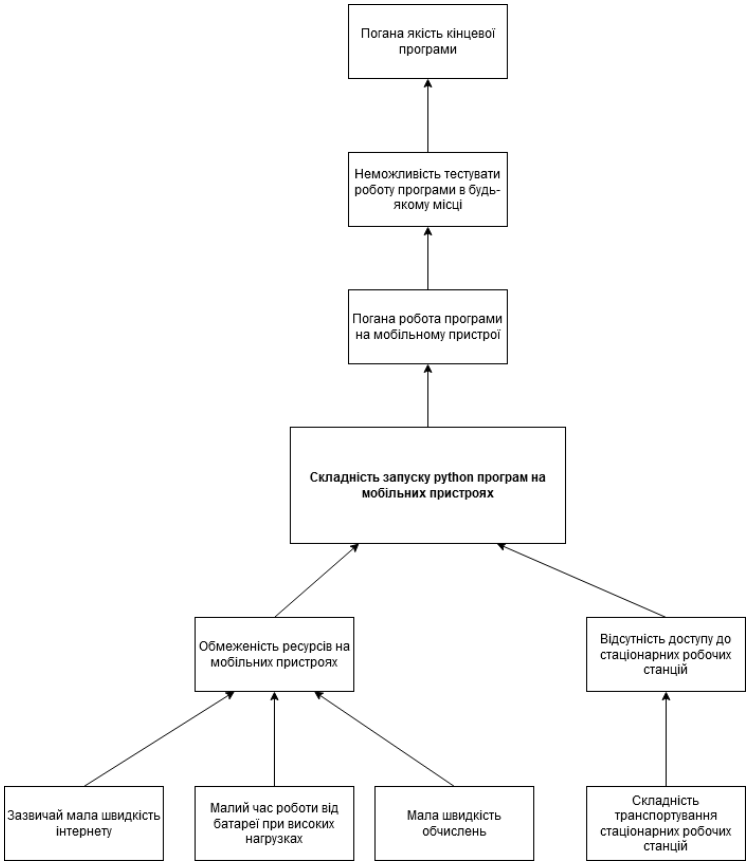

Рисунок 1.7 – Схема дерева проблем

З’являється основна проблема – «Складність запуску Python програм на мобільних пристроях». Вона породжує наслідки: в першу чергу, погану роботу програми на мобільному пристрої, що в свою чергу спричинює те, що програму стає важко тестувати, не маючи доступу до стаціонарної робочої станції, а це веде до погіршення якості кінцевої програми.

1.2.3 Дерево цілей

Дерево цілей – це графічне представлення ієрархії цілей, а також відображення розподілу основної мети на цілі, підцілі, завдання та окремі дії. Дерева мети - це головний універсальний метод системного аналізу.

Ієрархія цілей грає дуже важливу роль, оскільки вона встановлює структуру системи і забезпечує орієнтацію діяльності усіх одиниць системи на досягнення цілей верхнього рівня.

Головна мета створення дерева мети – декомпозиція. Декомпозиція – це метод, який базується на заміні вирішення однієї великої проблеми, вирішенням серії менших проблем, хоч і пов’язаних, але простіших.

Якщо ієрархія цілей належним чином сконструйована, то кожна одиниця, робить необхідний внесок у діяльність системи, щоб досягти основної мети системи.

У процесах аналізу і синтезу організаційних та економічних систем, дерево цілей відіграє важливу роль, тому що це дає змогу дати повну відповідь на питання "Що повинна зробити система ?".

Водночас, конструюючи дерево мети, можна в основному визначити і саму структуру системи.

Шляхи побудови дерева залежать від характеру мети, методологічного підходу і від людини, яка розвиває "дерево цілей", від її бачення поставлених перед нею завданнь та розуміння їхнього взаємозв’язку. Схема дерева цілей для покращення запуску Python програм на мобільних пристроях зображена на рисунку 1.8.

Рисунок 1.8 – Схема дерева цілей для покращення запуску Python програм на мобільних пристроях

1.2.4 Аналіз і вибір методів та алгоритмів розв’язання задачі

Оскільки сервіс складається з двох частин: клієнтської і серверної, то вибір методів та засобів потрібно робити окремо для кожної з них. Основу сервісу складає серверна частина, тому спочатку було обрано набір інструменів для її розробки. Оскільки мова програмування Python має зручні фреймворки для розробки серверів, то доцільно буде використати саме її для реалізації серверної частини проекту.

Наступним завданням був вибір фреймворку. Серед найпопулярніших фреймворків для таких задач можна виділити два відомі представники: Django та Flask. Вибір був зроблений на користь Flask [9], оскільки він зручний в використанні, не містить зайвих залежностей, може бути масштабований до складних програм та має чудову документацію. Як операційна система для сервера була обрана Ubuntu server [11], яка де-факто є стандартом для таких завдань. Для цієї операційної системи доступні усі необхідні пакети і їх можна легко встановити за допомогою пакетного менеджера.

Після написання серверної частини та її тестування за допомогою допоміжних інструментів необхідно було розробити мобільний додаток. Перш за все потрібно було визначитись для якої платформи розробляти додаток Android чи iOS. Цей вибір було зробити вкрай не легко, бо є аргументи на користь кожної з них. В результаті було обрано рішення, яке зможе задовольнити відразу обидві ці платформи і не потрібно буде обирати між ними. Цим рішенням став UI фреймворк для мобільних пристроїв Flutter [12]. Цей фреймворк був розроблений компанією Google саме для того, щоб можна було використовувати одинаковий код для усіх платформ. Програмування з використанням цього фреймворку ведеться за допомогою мови Dart [13], яка на етапі збирання вихідної програми перетворюється в нативний код для конкретної платформи.

1.3 Висновок до першого розділу

В першому розділі кваліфікаційної роботи був проведений аналіз предметної області, аналіз існуючих рішень. Були досліджені та представлені переваги хмарних обчислень, проведений аналіз середовищ розробки. Була сформована постановка задачі – описана характеристика об’єкту проектування, візуалізовано дерево проблем та дерево цілей, проведений аналіз методів та алгоритмів для розв’язання задачі.

2 ПРОЕКТУВАННЯ ТА ТЕСТУВАННЯ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

2.1 Спрощена модель системи

Спрощена модель зображена на рисунку 2.1. Вона відображає дві важливі якості системи – цілісність і відокремленість від довкілля. Система не є ізольованою від зовнішнього середовища і пов`язана з ним зв`язками, через які взаємодіє з користувачем. Крім цього, повинні існувати зв`язки іншого типу, що забезпечують її використання, тобто вплив на систему з боку середовища.

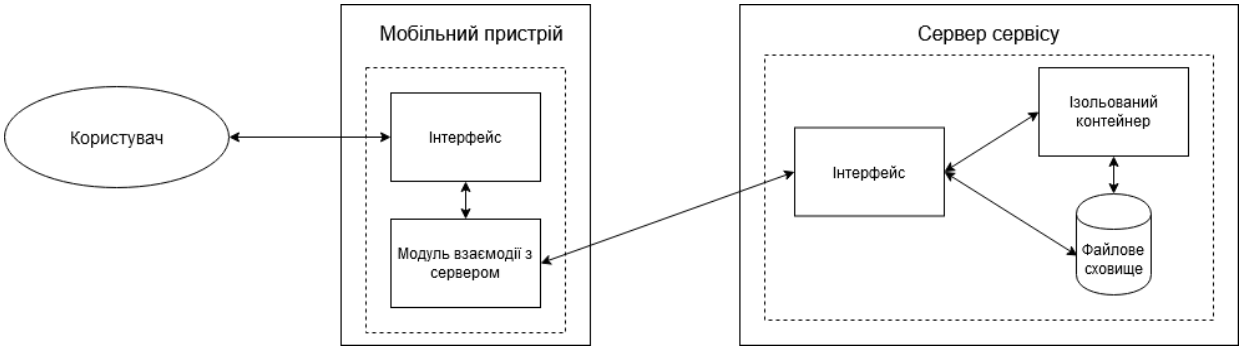

Рисунок 2.1 – Схема спрощеної моделі системи

Користувачами даного сервісу можуть бути всі, хто має доступ до віддаленого інтерпретатора. Користувач взаємодіє з інтерфейсом програми на мобільному пристрої, а саме: за його допомогою надає команду на надсилання кодів програм написаних мовою Python на сервер сервісу і їх запуск. Мобільний пристрій звязується з сервером сервісу за допомогою API і виконує дії задані користувачем. В свою чергу сервер сервісу запускає програму користувача в ізольованому контейнері і надсилає результати виконання на мобільний пристрій. Користувач зчитує отримані від сервера результати інтерпретації чи виконання програми через відповідний інтерфейс на мобільному пристрої.

2.2 Проектування клієнт-серверної архітектури

Блок-схема алгоритму – це графічне представлення логічної структури алгоритму, кожний етап обробки інформації якої зображується у вигляді геометричних блоків. Блок-схема алгоритму зображена на рисунку 2.2.

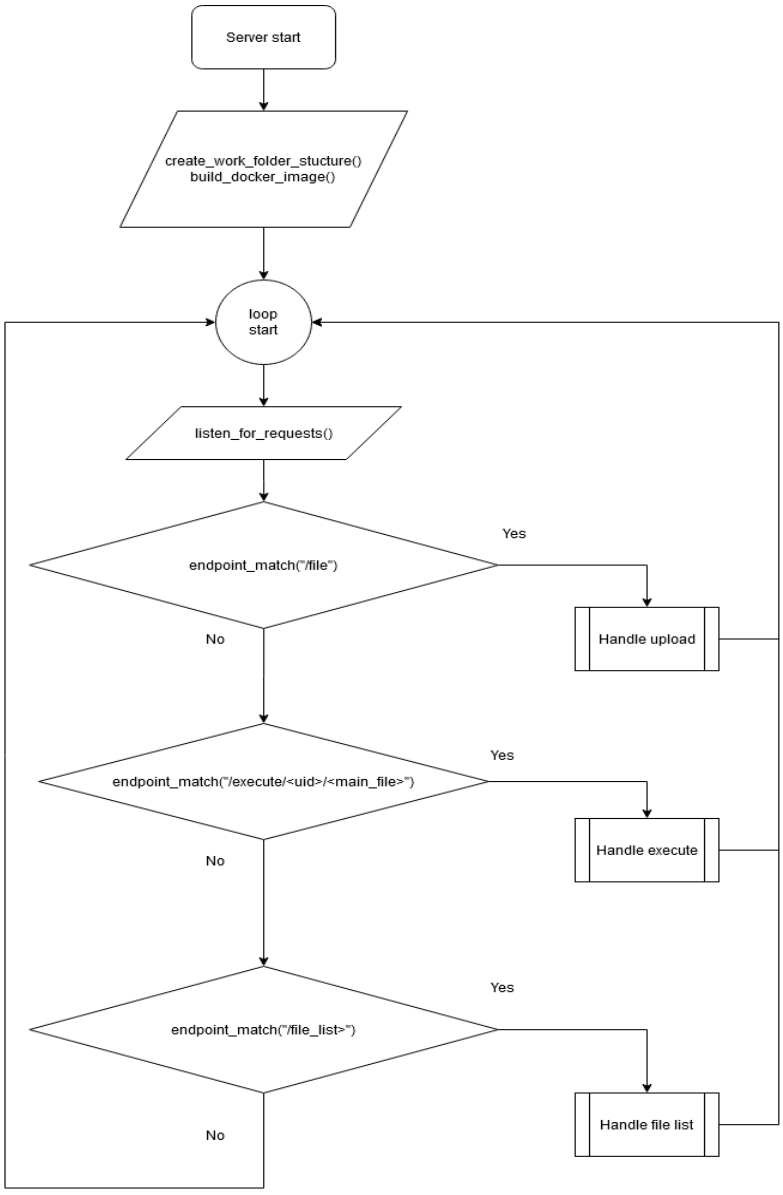

Рис. 2.2 – Блок-схема алгоритму функціонування сервера

З блок-схеми алгоритму роботи сервера на рисунку 2.2 можна побачити основну поведінку програмного рішення. Сервер в циклі очікує на запити від

клієнтів та обробляє їх, обираючи обробник в залежності від вибраного в запиті endpoint-у. Блок-схема алгоритму завантаження файлу на сервер показана на рисунку 2.3.

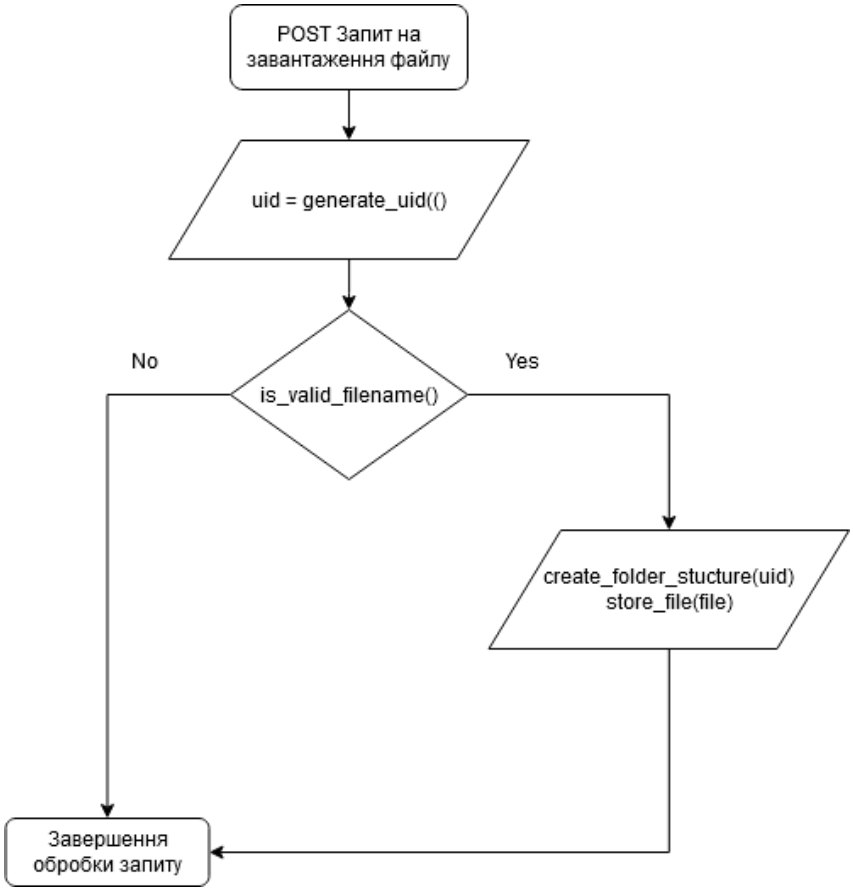

Рис. 2.3 – Блок-схема алгоритму завантаження файлу на сервер

З алгоритму завантаження файлу на сервер на рисунку 2.2 видно, що при завантаженні файлу створюється унікальний id, для цього завантаження, далі відбувається валідація файлу. Якщо валідація пройшла успішно, то з використанням згенерованого унікального id створюється нова директорія в робочій папці сервера та виконується збереження файлу в цю директорію.

Алгоритм виконання програмного коду на сервері зображений на рисунку 2.4.

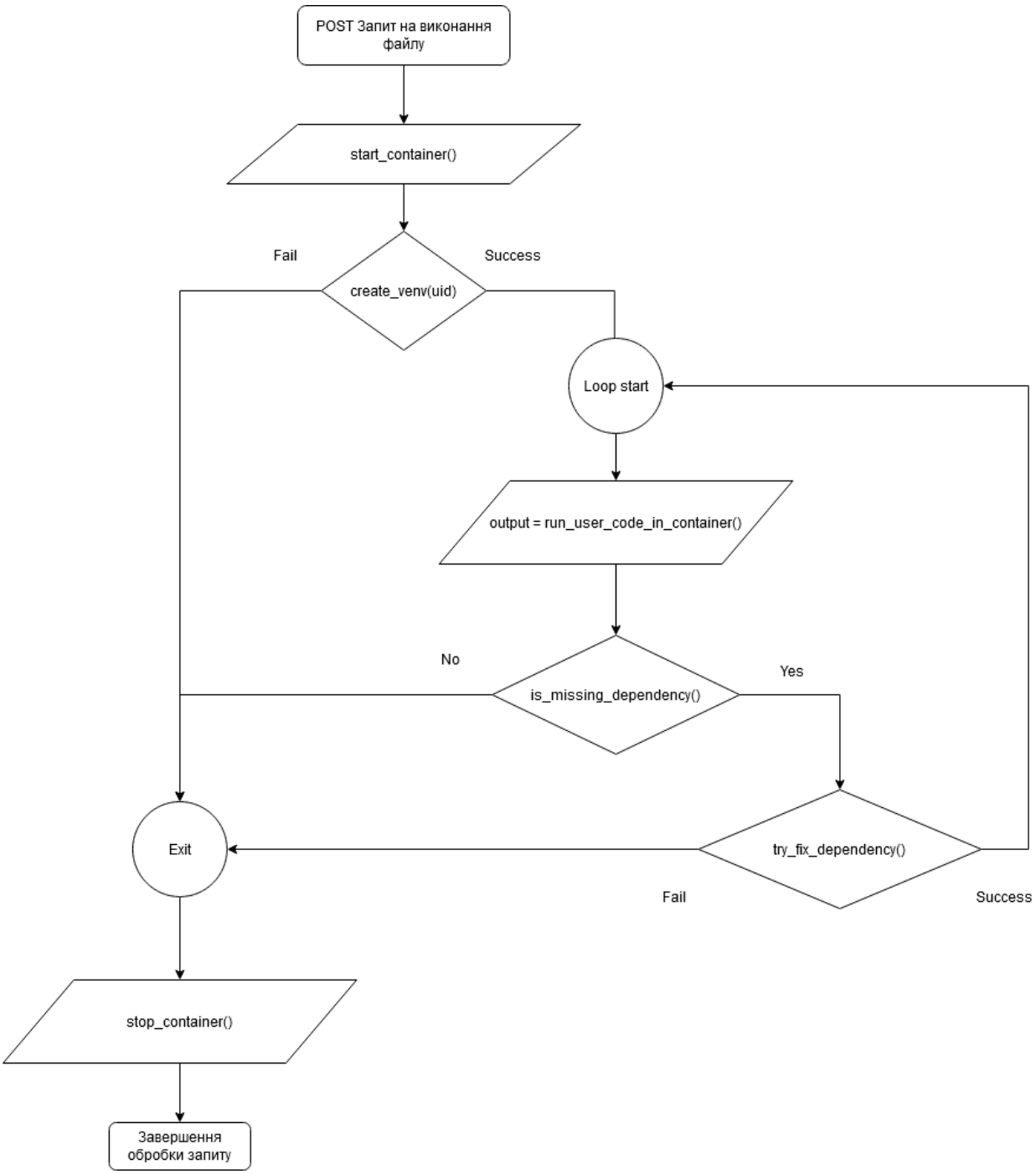

Рис. 2.4 – Блок-схема алгоритму виконання програмного коду на сервері

З алгоритму виконання файлу програми на сервері (рис 4.3) можна побачити, що для виконання програмного коду файлу спочатку створюється ізольований контейнер в якому буде виконуватись користувацька програма. Після цього користувацький код виконується у створеному ізольованому контейнері та, у разі успішного виконання, отримуються результати роботи

програми. Якщо в програмному файлі є помилки, які вказують на те, що для запуку програми не вистачає додаткових пакетів, то ці пакети встановлюються і проводиться повторний запуск користувацьої програми. Ці дії повторюються доти, доки програма не буде виконана успішно, або помилки в її виконанні не будуть повязані з додатковими пакетами. Після цього ізольований контейнер вимикається і обробка запиту закінчується.

2.3 Розробка та опис програмних рішень

2.3.1 Діаграма компонентів

Діаграма компонентів описує особливості фізичного представлення системи. Діаграма компонентів зображена на рисунку 2.5 та дає змогу показати архітектуру системи, отримавши залежності між програмними компонентами, в ролі яких може виступати різний код.

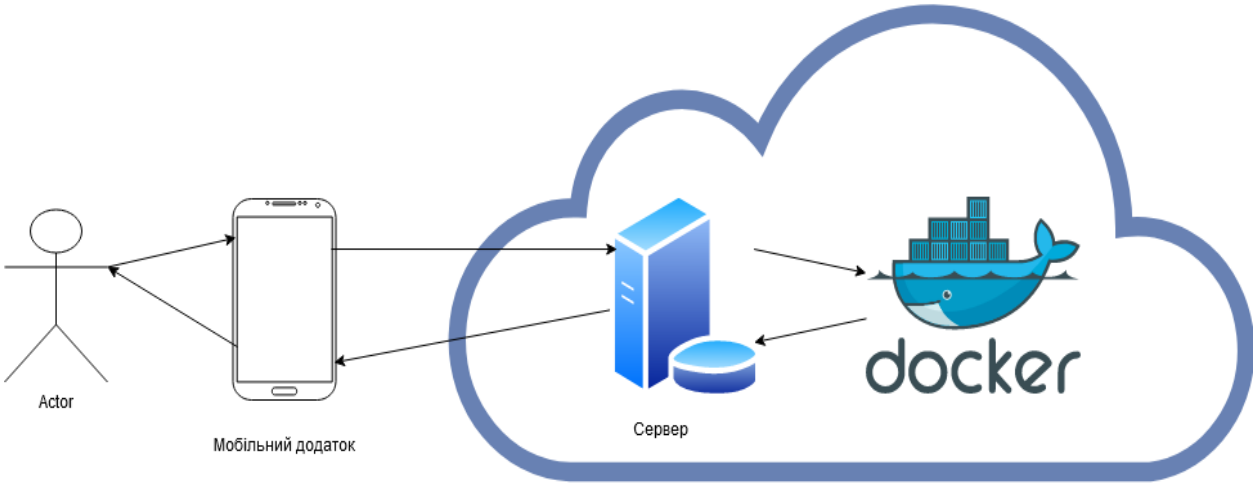

Рис. 2.2 – Діаграма компонентів

З діаграми компонентів можна побачити, що сервіс умовно можна розділити на 2 частини. Перша – це клієнтський мобільний додаток, з яким

взаємодіє користувач. Друга – це сервер і docker контейрер [10], які базуються на хмарі. Клієнт за допомогою мобільного додатку надсилає код програми, написаної мовою Python, на сервер, який виконує її в Docker контейнері та надсилає результати виконання. Важливо зазначити, що мобільний додаток може бути встановлений і ефективно працювати як на платформі Android так і на платформі iOS. Взаємодія мобільного додатку і сервера може відбуватись як за допомгою протоколу HTTP. В свою чергу сервер взаємодіє з docker за допомгою Docker API, яке доступне в мові Python як додатковий модуль.

2.3.2 Опис використаних сторонніх бібліотек і модулів

Для розробки сервісу було використано інтегроване середовище розробки Pycharm. Згідно стилю програмування під назвою PEP8, який є де-факто стандартом для Python програм, використані додаткові модулі повинні подаватись на початку файлу. На рисунку 2.3 можна побачити, які модулі були використані для розробки серверної частини.

```
1   import os
2   import re
3   import uuid
4   import shutil
5   import docker
6   import threading
7   import subprocess
8
9   from flask import Flask, request, Response, jsonify
10  from werkzeug.utils import secure_filename
```

Рис. 2.3 – Підключені сторонні модулі та бібліотеки

Для розробки серверної частини був обраний фреймворк Flask, який дозволяє швидко і ефективно розробляти backend програми за допомогою мови програмування Python. Лістинг серверу подано у додатку А. З його

допомогою були створені усі endpoint-ти, які використовує клієнтський додаток для взаємодії за допомогою протоколу HTTP. Також Flask має засоби для переторення Python типів в JSON, що значно пришвидшує розробку. Ще за допомгою цього фреймворку відбувається парсинг аргументів, які передає клієнт при надсиланні запиту. Важливою особливістю Flask є те, що він підтримує потокове надсилання результатів виконання запиту. Наявність цієї можливості дала змогу надсилати результати виконання клієнтської програми під час виконання, не очікуючи завершення її роботи.

Також, щоб забезпечити взаємодію серверної програми з Docker контейнерами було використанно Docker API. З його допомогою компілюються та створюються контейнери з необхідними параметрами, а також передаються клієнтські програми і отримуються результати виконання.

Для розробки мобільного додатку було використано UI фреймворк Flutter. Його основною перевагою є те, що код написаний з його допомогою буде однаково добре працювати на обох основних мобільних платформах: Ios та Android. Для написання програм за допомогою цього фреймворка використовується мова Dart, яка була створена Google, як мова яка оптимізована для розробки клієнтських додатків.

2.3.3 Розробка та опис програмних модулів

Інтегроване середовище розробки Pycharm дозволяє переглядати файли проекту під час розробки як дереововидну струкутру.

На рисунку 2.4 видно, що код сервісу поділений на 2 частини client та server, а також файлова структура проекту містить папку з прикладами (examples), в якій зберігаються коди програм, написані мовою Python і викорстовюються для тестуваня роботи сервісу. Код усіх програмнх модулів подано в Додатку.

Клієнтська частина складаєьться з 3-ох основних компонентів:

– main.dart – модуль який містить точку входу та координує запуск мобільного додатку;

– home_secreen.dart – модуль який містить структуру головного екрану(який корстувач бачить при запуску програми), а також логі ку його роботи

– output_screen.dart – модуль який містить структуру екрану для відображення робои програми, а також логіку для отримання резульататів роботи користувацького коду з сервера

А також, з великої кількості компонентів які генеруються UI фреймворком Flutter, як наприклад папки ios та android, які містять код специфічнний для кожної платформи.

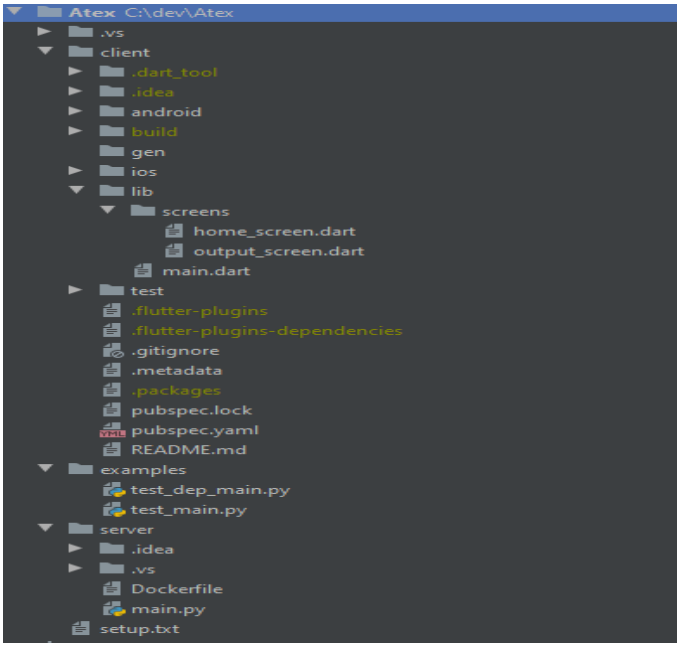

Рис. 2.4 – Структура програмних модулів системи

Серверна частина містить такі модулі:

– main.py – містить основну логіку сервера, з яким взаємодіє клієнтський додаток;

– Dockerfile – файл яки містить налаштування docker контейнера і використовується згодом для його компіляції і запуску;

– setup.txt – файл який містить дії необхідні для запуску сервера в новому середовищі.

2.3.4 Розробка та опис інтерфейсу користувача

Розробка користувацького інтерфейсу являє собою використання заздалегідь готових графічних елементів, наявних у графічному фреймфорку, або їх модифікації. До таких елементів належать списки, кнопки, зображення, текстові поля, меню тощо. На рисунку 2.5 зображений інтерфейс мобільного додатку лістинг якого подано у додатку Б.

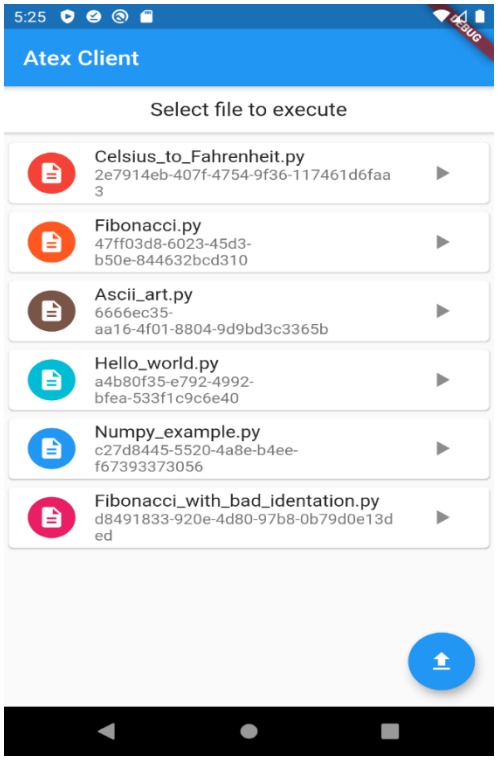

Рис. 2.5 – Інтерфейс головного екрану мобільного додатку

Відразу після відкриття мобільного додатку користувач опиняється на головному екрані програми. Базовим компонентом для екрану є Scaffold, який реалізує мінімальний дизайн екрану в стилі material і надає візуальний макет. Верхню частину екрану займає AppBar, який зазвичай містить назву програми, або меню, в якому знаходиться користувач, і додатково може містити кнопки

для взаємодії з програмою. Під ним знахождиться компонент під назвою Card, який містить текст-підказку для користувача, що потрібно вибрати файл для виконання. Далі знаходиться список з Card-ів, кожен з яких відображає один завантажений файл на сервер. Кожен з елементів, своєю чергою, складається з: CircleAvatar (який містить Icon) – компонент, який відповідає за піктограму в лівій частині картки, її колір визначається з унікального id конкретного завантаження. Сам унікальний id розташований в частині subtitle та має сірий колір шрифту. Також в частині title картка містить назву файлу, завантаженого на сервер. Важливим компонентом картки є кнопка IconButton, яка знаходиться в правій частині і відповідає за запуск завантаженої програми на виконання, – при натисканні на неї надсилається запит на виконання і відкривається екран з результатами виконання програми. Завдяки компоненті під назвою Dismissible файли можна видаляти з сервера звичайним зсуванням картки в сторону. Процес видалення проілюстровано на рисунку 2.6.

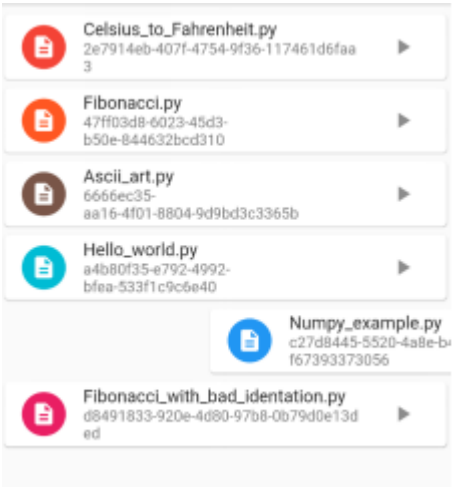

Рис. 2.6 – Інтерфейс видалення файлу з сервера

Після завантаження файлу на сервер або після вибору файлу користувачем з уже завантажених файлів відкривається екран результаттів запуску користувацької програми, його можна побачити на рисунку 2.7.

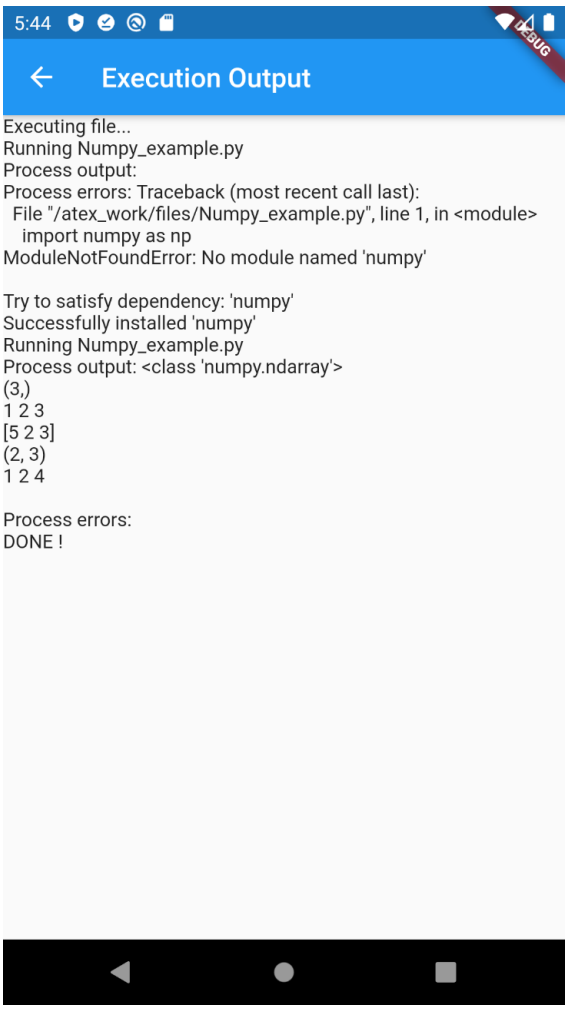

Рис. 2.7 – Інтерфейс виведення результатів запуску програми

Екран для виведення результатів роботи програми, як і головний екран базується на компоненті під назвою Scaffold, а також в верхній частині екрану міститься AppBar, який має додаткову кнопку для повернення на головний екран. Основну частину екрану займає відображення даних результату, що надійшли з сервера. Лістинг подано у додатку В.

2.3.5 Опис проблем, які виникали під час розробки

Під час розробки сервісу виникало багато проблем та нестандартних ситуацій, однією з них стало те, що користувач отримував результати

виконання програми тільки після її повного виконання. Це створюало великі незручності для користувача, коли здійснювався запуск програм які мають довгий час виконання, а під час своєї роботи поступово надають результати. Для вирішення цієї проблеми було використанно можливість серверного фреймворку Flask для відправки результатів виконання запиту по мірі їх надходження – так званий streamed response. Також у клієнтському додатку було використано спеціальний засіб для обробки результатів таких запитів, цей засіб дозволяє оновлювати дані в додатку у разі надходження нових даних із сервера. Цей підхід забезпечив те, що користувач може бачити результати виконання своєї програми в реальному часі і не чекати повного завершення її виконання.

2.4 Системи захисту серверної частини

2.4.1 Захист каналу зв'язку між клієнтом та сервером

Одним із основних засобів захисту зв’язку між клієнтом і сервером виступає протокол передачі даних в мережі. Оскільки сервер виступає в ролі API, то самим оптимальниим рішенням для захисту запитів та передачі даних буде використання протоколу HTTPS.

HTTPS – є розширенням протоколу передачі гіпертексту (HTTP). Він використовується для безпечного спілкування через комп’ютерну мережу і широко використовується в Інтернеті. У протоколі HTTPS протокол зв'язку шифрується за допомогою захисту транспортного рівня (TLS) або, раніше, рівня захищених сокетів (SSL). Тому протокол також називають HTTP через TLS, або HTTP через SSL.

Основними мотивами HTTPS [19] є автентифікація веб-сайту, до якого здійснюється доступ, та захист конфіденційності та цілісності даних, якими обмінюються під час передачі. Він захищає від атак "посередника", а

двонаправлене шифрування комунікацій між клієнтом і сервером захищає комунікації від прослуховування та фальсифікацій. Аспект автентифікації HTTPS вимагає від третьої сторони, якій довіряють, підписувати цифрові сертифікати на стороні сервера. Це була історично дорога операція, що означало, що повністю автентифіковані з'єднання HTTPS зазвичай знаходили лише в захищених послугах платіжних операцій та інших захищених корпоративних інформаційних системах у Всесвітній павутині. У 2016 році кампанія Electronic Frontier Foundation за підтримки розробників веб-браузерів призвела до того, що протокол став більш поширеним. HTTPS зараз користується веб-користувачами частіше, ніж оригінальний незахищений HTTP, першочергово для захисту автентичності сторінок на всіх типах веб-сайтів, захисту рахунків, зберігання конфіденційності спілкування користувачів, ідентифікації та перегляду веб-сторінок.

На даному етапі розробки був використаний протокол HTTP, оскільки для зручного використання протоколу HTTPS потрібен ліцензійний SSL Certificate. Актуальна версія системи призначена для тестування, дебагінгу та презентації можливостей даної системи. При комерційному використанні розробленого сервісу рекомендується розгорнути його у хмарі. Після розгортання прописавши шлях до згенерованих ключів та добавивши відкритий ключ у код додатку буде доступний зв’язок між сервером та клієнтом по HTTPS.

2.4.2 Захист мережі від DoS атак

У обчислювальних системах, атака відмови в обслуговуванні (DoS-атака) - це кібератака, при якій винуватець прагне зробити машину або мережевий ресурс недоступними для призначених користувачів тимчасово або на невизначений час, порушуючи послуги хоста, підключеного до Інтернету. Відмова в обслуговуванні зазвичай досягається заповненням цільової машини

або ресурсу зайвими запитами, намагаючись перевантажити системи та не допустити виконання деяких або всіх законних запитів [20].

У розподіленій атаці відмови в обслуговуванні (DDoS-атака) вхідний трафік, що затоплює жертву, походить із багатьох різних джерел. Це фактично унеможливлює зупинку атаки, просто заблокувавши одне джерело.

Атака DoS або DDoS є аналогічною групі людей, які переповнюють вхідні двері магазину, ускладнюючи доступ законних покупців, тим самим порушуючи торгівлю. Візуалізація DDoS атаки на рисунку 2.8.

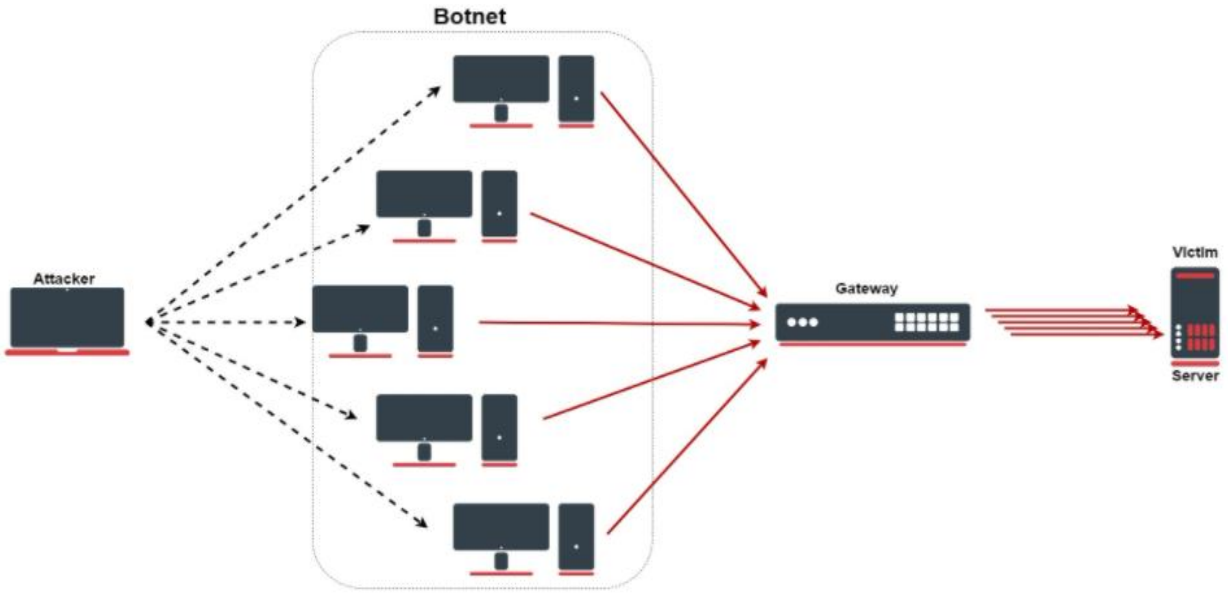

Рис. 2.8 – Схема DDoS атаки на сервер

Зловмисники, що здійснюють атаки на DoS, часто націлюються на сайти або служби, розміщені на гучних веб-серверах, таких як банки або шлюзи оплати за допомогою кредитних карток. Помста, шантаж та активізм можуть мотивувати ці напади.

Для забезпечення захисту сервера від DoS атаки, було прийнято рішення розпізнавати та блокувати хости які виконують DoS атаку на мережевому рівні за допомогою маршрутизатора Mikrotik [21].

Конфігурація Mikrotik firewall зображена на рисунку 2.9.

```
/ip firewall address-list
add list=ddos-attackers
add list=ddos-target
/ip firewall filter
add action=return chain=detect-ddos dst-limit=32,32,src-and-dst-addresses/10s
add action=add-dst-to-address-list address-list=ddos-target address-list-timeout=10m chain=detect-ddos
add action=add-src-to-address-list address-list=ddos-attackers address-list-timeout=10m chain=detect-ddos
/ip firewall raw
add action=drop chain=prerouting dst-address-list=ddos-target src-address-list=ddos-attackers
```

Рис. 2.9 – Конфігурація Mikrotik firewall

В конфігурації відбуваються наступні процеси:

– кожне нове підключення буде надсилатись до певного ланцюжка брандмауера, де буде виявлятись DDoS;

– додається правило з параметром "dst-limit". Цей параметр записаний у наступному форматі: dst-limit = count [/ time], пакет, режим [/ expire]. Воно буде порівнювати 32 пакети з 32 пакетами залежно від потоку адреси призначення та джерела, який поновлюється кожні 10 секунд. Правило діятиме доти, доки не буде перевищено задану норму.

– Весь законний трафік повинен проходити через "action = return", але у випадку DoS / DDoS "dst-limit" буфер буде переповнений, і правило не буде "ловити" будь-який новий трафік. Далі відбувається створення списку для зловмисників та жертв, які будуть заблоковані

– За допомогою розділу фільтра брандмауера зловмисників буде добавлено у список "DDoS-зловмисники" та жертв у список "DDoS-цілі"

2.4.3 Захист файлової системи сервера

На етапі проектування системи розглядалось багато альтернативних підходів, найважливішими з них були альтернативи для ізольованих

контейнерів та підходу до розробки мобільного додатку для забзпечення захисту системи та персональних даних.

Щодо ізольованих контейнерів, то потрібно було мінімізувати ризики, які повязані з тим, що користувацька програма може вплинути на роботу сервера сервісу або на роботу інших програм, які виконує сервіс.

Одна з перших ідей для вирішення цієї проблеми полягала в тому, щоб реалізувати ізоляцію засобами самої мови програмування Python. Для прикладу, мова програмування Go має вбудований sandbox в якому можна запускати програми, які були отримані з не надійних джерел. Проте, офіційна імлементація мови Python не має такої функціональності. Потрібна функціональність є в альтернативній імлементації мови під назвою PyPy, яка позиціонує себе як швидша версія мови Python. Але, нажаль, режим sandbox дуже лімітований, він не дозволяє використовувати сторонні модулі, а підтримка навіть стандартних модулів мови Python дуже обмежена, тому було вирішено відмовитись від цієї альтернативи.

Наступною альтернативою було використання додаткових програминих засобів, які надають можливість ізоляції виконання. На ринку присутня велика кількість засобів, призначених для того, щоб ізолювати виконання програми. Серед них було обрано 3 основних альтернативи: Linux jail, Virtual Box та Docker. Linux jail був відкинутий на етапі дослідження прикладів використання через складність налаштування та великий ризик помилки. Virtual Box має значно кращі характеристики, щодо рівня надійності ізоляції, але має надто великі накладні витрати на запуск цілої операційної системи для виконання однієї користувацької програми. Саме тому ця альтернатива була відкинута і був обраний Docker.

Docker дозволив ізоювати виконання користувацької програми від основної системи, відповідно доступ до системи був повністю заблокований.

Також був внесений ряд обмежень по часу виконання та нагрузці на систему, Docker надав можливості повного аналізу трафіку та коду користувача який запускається.

Маючи ці дані, можна легко вивчити активність користувачів і позначити її як підозрілу.

Docker дозволив мінімізувати інтерфейс клієнта та вилучити етап авторизації, оскільки зловмисник маючи доступ до виконання свого коду на сервері не зможе отримати доступ до конфіденційних даних інших користувачів або до налаштувань самої системи на якій розташований сервер.

2.5 Тестування захисту розробленої клієнт-серверної архітектури

Тестування – це будь-яка діяльність із системою, спрямована на виявлення помилок у програмному продукті. Тестування проводиться з метою знайти помилки в програмі чи неправильні елементи системи і тим самим підвищити її надійність, а отже, цінність.

Було вирішено використати декілька підходів до взлому сервера:

– DoS атака на відкритий REST API сервера;
– отримати доступ до файлів системи сервера;
– зациклити виконання програми цим самим навантажити систему;
– запустити небажану активність на сервері.

DoS атака на відкритий REST API сервера. При спробі виконати велику кількість запитів на хост сервера, налаштований Mikrotik одразу блокує ip адрес з якого почалась атака. В залежності від вибору місця розгортання сервера можна піднімати рівень захищеності від атак на хост з метою блокування для інших користувачів.

Отримання доступу до файлів системи сервера. При спробі запустити код для аналізу файлів структури файлів, було виявлено, що нам для перегляду доступні тільки файли самого докер контейнера в якому запускається наш код.

На рисунку 2.9 можна побачити, що до переліку директорій системи входить тільки дві основних папки:

– files;

– venv.

Папка files відповідає за код користувача який користувач добавляє та запускає на сервері.

Папка venv – це віртуальне середовище мови програмування Python, яке містить сам інтерпретатор мови відповідної версії, в даному випадку 3.9, стандартні та завантажені бібліотеки, власні залежності, кеш, менеджер пакетів pip та інше.

Отож, можна зробити висновок, що файлова система сервера надійно захищена від несанкціонованого доступу, перегляду, та зміни файлів, а також від перегляду конфіденційної інформації інших користувачів.

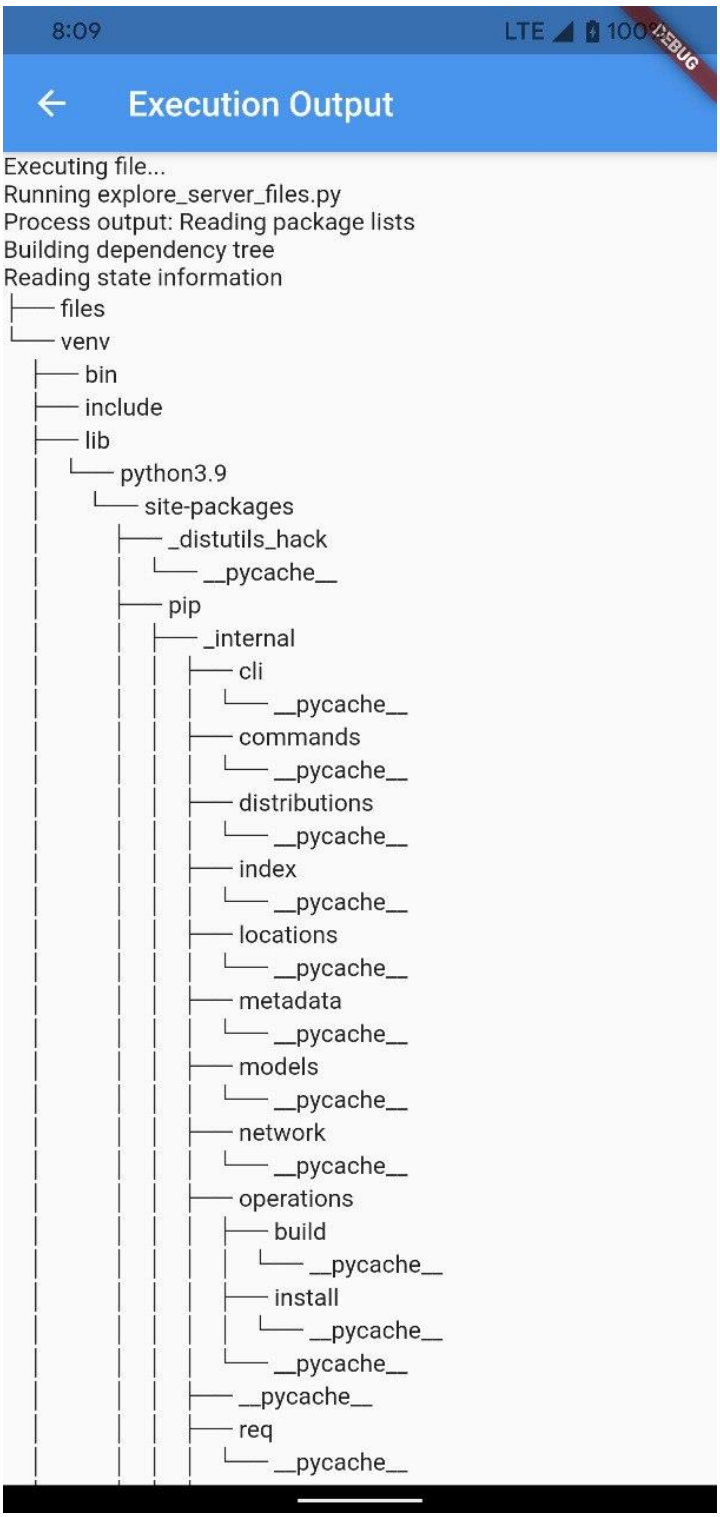

Рис. 2.9 – Дерево директорій ізольованого контейнера

При зацикленні виконання програми, піднятий контейнер автоматично зупиняється після визначеного максимального часу життя контейнера на розсуд адміністратора.

При запуску небажаної активності, для прикладу майнинг біткоіна чи DDoS, то ця вся інформація буде логуватись і адміністратор на базі цих логів може припинити небажану активність.

Також адміністратор має можливість добавити список бібліотек та фреймворків які буде заборонено встановлювати на сервері. Це можуть бути ті ж самі бібліотеки які будуть допомагати виконувати DDoS атаки на інші сервера або майнити біткоін.

2.6 Висновок до другого розділу

В другому розділі кваліфікаційної роботи був описаний процес проектування, розробки, захисту та тестування серверу для віддаленої інтерпретації python коду. Було спроектовано та розроблено додаток з використанням мови програмування Dart. Були описані сторонні програмні модулі, протестовано захист сервера від несанкціонованого доступу до системи, спроектований захист передачі даних між клієнтом і сервером та розроблений захист від DoS атак на сервер.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Долікарська допомога при вивихах

Вивих – різновид травм, що характеризується як порушення конфігурації суглобових поверхонь. Це вид травми, за якої суглобовий кінець однієї кістки зміщується за межі суглобової поверхні іншої, котра разом із першою утворює суглобову пару.

Також термін «вивих» і «підвивих» застосовують до деяких уражень кришталика ока. Коли відбувається повний розрив зонулярних волокон, а кришталик в результаті цього розміщений за межами зіниці – іде мова про вивих кришталика. Як відбувається частковий розрив зонулярних волокон, а кришталик децентралізований, але частково розміщений у межах зіниці, то тоді це трактують як підвивих кришталика.

Вивихи за часом поділяються на [22]:

– свіжі – до 3-х діб після травми;

– несвіжі – від 3-х діб до 3-х тижнів;

– застарілі – більше 3-х тижнів.

Також вивихи за особливістю поділяються на:

– звичний вивих – це багаторазові вивихи без надмірних фізичних зусиль;

– невправний вивих – це, як правило, свіжий вивих, який через різні причини не вдається усунути. Причиною може бути інтерпозиція капсули, зв'язок, сухожилків;

Ознаки вивиху:

– сильний біль у кінцівці у ділянці суглоба, особливо при рухах;

– значний набряк ураженого сглоба;

– блідність шкіри навколо суглоба;

– деформація ділянки суглоба;

– відсутність активних і неможливість пасивних рухів у суглобі;

– кінцівка зафіксована у неприродньому положенні;

– зміна довжини кінцівки. (видиме укорочення або подовження).

Надання домедичної допомоги при вивихах:

1) при наявності кровотечі із судин – зупинка кровотечі джгутом чи імпровізованими джгутами (закрутка, перетягання ременем), накладання стискальної пов'язки.

2) при відкритому вивиху накладання первинної (асептичної) пов'язки з метою профілактики вторинного мікробного забруднення.

3) транспортна іммобілізація: аутоіммобілізація, підручними засобами.

Задачі транспортної іммобілізації:

a) кінцівка зафіксована у неприродньому положенні;

b) зупинка кровотечі (артеріальної);

c) попередження травматичного шоку;

d) накладання стерильної пов'язки на рану;

e) проведення іммобілізації табельними або підручними засобами.

Основні принципи транспортної іммобілізації:

1) для знерухомлення ушкодженої кінцівки необхідно іммобілізувати два суміжних суглоби, а при вивиху плечової або стегнової кістки три суглоби.

2) моделювання шини слід проводити по здоровій кінцівці, або на тому, хто її буде накладати, тобто на медпрацівнику.

3) при іммобілізації кінцівки необхідно надати їй фізіологічне положення, а якщо це неможливо, то таке положення, при якому кінцівка менш за ве травмується.

4) при відкритих вивихах вправлення не виконують, накладають стерильну пов'язку і кінцівку фіксують в тому положенні, в якому вона знаходилась в момент ушкодження.

5) іммобілізацію накладають поверх одягу і взуття потерпілого, між шиною і кінцівкою потерпілого кладуть м'яку ватно-марлеву підкладку.

6) при відкритих вивихах на рану слід накласти стерильну пов'язку.

7) при фіксації шини не можна закрити місце накладання джгута, щоб була можливість коригувати стан джгута.

8) іммобілізована кінцівка перед транспортуванням в холодну пору року повинна бути обов'язково утеплена з метою профілактики відмороження.

9) під час перекладання потерпілого з нош, ушкоджену кінцівку повинен тримати помічник.

3.2 Соціальне значення охорони праці

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров’я і працездатності людини у процесі трудової діяльності [23].

Головна мета ОП – створення на кожному робочому місці безпечних умов праці, безпечної експлуатації обладнання, зменшення або повна нейтралізація дії шкідливих і небезпечних виробничих факторів на організм людини і, як наслідок, зниження виробничого травматизму та професійних захворювань.

Соціальне значення охорони праці полягає в сприянні росту ефективності суспільного виробництва шляхом безперервного вдосконалення і поліпшення умов праці, підвищення їх безпеки, зниження виробничого травматизму і профзахворювань.

Соціальне значення охорони праці проявляється в зростанні продуктивності праці, збереженні трудових ресурсів і збільшенні сукупного національного продукту.

Зростання продуктивності праці відбувається в результаті збільшення фонду робочого часу завдяки скороченню внутрішньо-змінних простоїв шляхом ліквідації мікротравм або зниження їх кількості, а також завдяки

запобіганню передчасного стомлення шляхом раціоналізації і покращення умов праці та введенню оптимальних режимів праці і відпочинку та інших заходів, які Сприяють підвищенню ефективності використання робочого часу.

Збереження трудових ресурсів і підвищення професійної активності працюючих відбувається завдяки покращенню стану здоров'я і подовженню середньої тривалості життя шляхом покращення умов праці, що супроводжується високою трудовою активністю і підвищенням виробничого стажу. Підвищується професійний рівень також завдяки зростанню кваліфікації і майстерності.

Збільшення сукупного національного продукту відбувається завдяки покращенню вищеперелічених показників та їх складових компонентів.

Охорона праці має соціальне, економічне та правове значення.

Кожний працівник має право на охорону праці, що є основним правом, закріпленим у ст. 43 Конституції та КЗпП України. Право на охорону праці працівник реалізує в процесі трудової діяльності.

Зміст права на охорону праці включає право працівника на:

– робоче місце, що відповідає вимогам охорони праці;

– загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності;

– відмову від виконання робіт у випадку виникнення небезпеки для його життя та здоров'я внаслідок порушення вимог охорони праці;

– забезпечення індивідуального та колективного захисту за рахунок роботодавця;

– навчання безпечним методам праці за рахунок роботодавця;

– звернення до органів державної влади та місцевого самоврядування, роботодавця, профспілки з питань охорони праці;

– особисту участь або участь через своїх представників у розгляді питань, пов'язаних із забезпеченням безпечних умов праці на його робочому

місці, і в розслідуванні нещасного випадку, що відбувся з ним, на виробництві або професійного захворювання;

– медичний огляд відповідно до медичних рекомендацій зі збереженням місця роботи (посади) і середнього заробітку під час його проходження;

– компенсації та пільги, встановлені законодавством, колективним договором чи угодою, трудовим договором, якщо працівник зайнятий на важких роботах і роботах зі шкідливими або небезпечними умовами праці.

3.3 Загальні вимоги безпеки з охорони праці для користувачів ПК

Загальні вимоги безпеки при роботі з комп'ютером та іншою оргтехнікою [24]:

1) до самостійної роботи з комп'ютером, ноутбуком, принтером, ксероксом, сканером, плазмовою панеллю, LCD-дисплеєм та іншою оргтехнікою допускаються особи, які досягли 18 річного віку, пройшли медичний огляд, ознайомлені з інструкцією з охорони праці при роботі з оргтехнікою, не мають протипоказань за станом здоров'я.Моделювання шини слід проводити по здоровій кінцівці, або на тому, хто її буде накладати, тобто на медпрацівнику.

2) під час роботи на комп'ютері та іншій оргтехніці на користувача можуть впливати наступні небезпечні та шкідливі фактори:

a) електрострум і випромінювання;

b) перенапруження зору під час роботи з електронними пристроями, монітором, особливо при не правильному розташуванні екрана по відношенню до очей.

3) освітлювальні установки повинні забезпечувати рівномірне освітлення і не повинні утворювати засліплюючих відблисків на клавіатурі, а також на екрані монітора за напрямом очей.

4) при роботі з комп'ютером, принтером, ксероксом та іншою периферійною технікою не допускається розташування робочого місця в приміщеннях без природного освітлення, без наявності природної або штучної вентиляції.

5) робоче місце з комп'ютером та оргтехнікою повинно розміщуватися на відстані не менше 1м від стіни, від стіни з віконними отворами - на відстані не менше 1,5 м.

6) кут нахилу екрана монітора або ноутбука по відношенню до вертикалі повинен складати 10-15 градусів, а відстань до екрана - 500-600 мм.

7) для захисту від прямих сонячних променів повинні передбачатися сонцезахисні пристрої (плівка з металізованим покриттям, регульовані жалюзі з вертикальними панелями).

8) освітлення повинно бути змішаним (природним та штучним).

9) у приміщенні кабінету і на робочому місці необхідно підтримувати чистоту і порядок, проводити систематичне провітрювання.

10) про всі виявлені під час роботи несправності обладнання необхідно доповісти керівнику, у випадку поломки необхідно припинити роботу до усунення аварійних обставин. При виявленні можливої небезпеки, попередити оточуючих та негайно повідомити керівнику; утримувати в чистоті робоче місце, не захаращувати його сторонніми предметами.

11) про нещасний випадок очевидець, працівник, який його виявив, або сам потерпілий повинні доповісти безпосередньо керівникові установи і вжити заходів з надання медичної допомоги.

12) особи, винні в порушенні вимог, вимагаємих данною інструкцією з охорони праці при роботі з комп'ютером, принтером, ксероксом та іншою оргтехнікою, притягаються до дисциплінарної відповідальності у відповідності з чинним законодавством.

3.4 Висновки до третього розділу

В третьому розділі кваліфікаційної роботи описані питання з безпеки життєдіяльності та охорони праці.

В першому питанні розглядається долікарська допомога при вивихах, що є важливим питанням, оскільки адміністратор даної системи повинен вміти надавати домедичну допомогу у випадку травмування співробітників при роботі з фізичними серверами.

В другому питанні описане соціальне значення охорони праці. Це одне з найважливіших питань, тому що створення безпечних умов праці, безпечної експлуатації обладнання для операторів серверної частини призведе до зниження виробничого травматизму та професійних захворювань.

Третє питання включає в собі загальні вимоги безпеки з охорони праці для користувачів ПК. Оскільки більшу частину часу працівники проводять за персональним комп’ютером (ПК) – це питання також відіграє ключову роль при обслуговуванні даної системи та описує правила поведінки включаючи техніку безпеки перед початком роботи, безпосередньо під час роботи та після закінчення роботи з ПК.

ВИСНОВКИ

Під час виконання даної бакалаврської кваліфікаційної роботи мною був розроблений сервіс для безпечної віддаленої інтерпретації програм, написаних мовою Python, у реальному часі та клієнтський додаток для зручного та

використання цього сервісу на мобільних пристроях з операційною системою iOS та Android.

Проаналізувавши актуальність та ситуацію на ринку програмних продуктів, я дійшов до висновку, що сервіси віддаленої інтерпретації програм є доволі популярним продуктом у сфері інтернет технологій, проте зазвичай вони орієнтовані на користувача стаціонарної робочої станції, а також встановив, що буде доцільною розробка сервісу, який дасть змогу віддалено інтерпретувати Python програми, надіслані з мобільних пристроїв.

Під час аналізу виявив, що велика кількість серверів, веб ресурсів, сервсів мають значну кількість вразливостей, та щоб забезпечити захист конфіденційної інформації, серверу та зв’язку з ним – були проаналізовані методи захисту клієнт-серверної архітектури від несанкціонованого доступу.

При проведенні системного аналізу були розроблені дерева проблем та цілей, що вказали на основні проблеми запуску Python програм на мобільних пристроях (зокрема, проблеми обмеженості ресурсів мобільних пристроїв, таких як швидкість Інтернету, малий час роботи від батареї при високих навантаженнях, мала швидкість обчислень) та запропоновано варіанти їх вирішення – інтерпретація програм на віддаленому сервері, що забезпечує значно вищу швидкість обчислень. Різноманітні діаграми діяльності системи допомогли покращити розуміння процесу віддаленої інтерпретації та роботи програмного продукту.

Для розробки сервісу ізольованого опрацювання Python програм та мобільного користувацького застосунку використовувалися такі технології та інструментарії:

- середовища розробки – Android Studio та PyCharm;
- мови програмування – Python (серверна частина) і Dart (клієнська частина);
- інструмент для створення ізольованих контейнерів – Docker;
- фреймворк клієнтської частини – Flutter;

– фреймворк серверної частини – Flask.

Використання перелічених технології та засобів дало змогу зручно та ефективно створити систему для оперативної віддаленої інтерпретації програм, що має достатньо широкий функціонал виконання користувацьких програм, забезпечує їхню захищеність та вирізняється добрими ергономічними характеристиками.

Розроблений програмний продукт був протестований та захищений від DoS/DDoS атак, взлому головної системи сервера, перехоплення даних при передачі між сервером та клієнтом. Для цього були досліджені та використані наступні технології:

– docker контейнери для ізольованого виконання програм користувача;

– протокол HTTPS для шифрування зв’язку між сервером та клієнтом

– firewall на маршрутизаторі Mikrotik для фільтрування трафіку до сервера та блокування можливих DoS/DDoS атак.

ПЕРЕЛІК ДЖЕРЕЛ

1. TIOBE Index for June 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/ – Дата доступу: 02.05.2021

2. Top 10 Python Applications in the Real World You Need to Know [Електронний ресурс] – Режим доступу до ресурсу: https://www.edureka.co/blog/python-applications/ – Дата доступу: 02.05.2021

3. Repl.it [Електронний ресурс] – Режим доступу до ресурсу: https://repl.it/languages/Python3 – Дата доступу: 02.05.2021

4. Codepad [Електронний ресурс] – Режим доступу до ресурсу: http://codepad.org/ – Дата доступу: 04.05.2021

5. Ideone [Електронний ресурс] – Режим доступу до ресурсу: https://ideone.com/ – Дата доступу: 05.05.2021

6. Jdoodle [Електронний ресурс] – Режим доступу до ресурсу: https://www.jdoodle.com/python3-programming-online/ – Дата доступу: 05.05.2021

7. Python.org [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/doc/essays/blurb/ – Дата доступу: 05.05.2021

8. Tutorialspoint Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/python/python_overview.htm – Дата доступу: 07.05.2021

9. Flask. A simple framework for building complex web applications [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/Flask/ – Дата доступу: 08.05.2021

10. What is Docker? [Електронний ресурс] – Режим доступу до ресурсу: https://opensource.com/resources/what-docker – Дата доступу: 08.05.2021

11. What is Ubuntu? [Електронний ресурс] – Режим доступу до ресурсу: https://help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html – Дата доступу: 09.05.2021

12. What is Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/+ – Дата доступу: 13.05.2021

13. Why Flutter Uses Dart [Електронний ресурс] – Режим доступу до ресурсу: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf – Дата доступу: 14.05.2021

14. What is cloud computing [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is-cloud-computing – Дата доступу:

15. Top 6 cloud service providers who'll dominate the cloud war [Електронний ресурс] – https://www.techaheadcorp.com/blog/top-cloud-service-providers/ – Дата доступу: 16.05.2021

16. What is PyCharm? [Електронний ресурс] – Режим доступу до ресурсу: https://intellipaat.com/blog/what-is-pycharm/ – Дата доступу: 16.05.2021

17. What is the use of Android studio? [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/What-is-the-use-of-Android-studio – Дата доступу: 17.05.2021

18. Install flatter Windows [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/docs/get-started/install/windows – Дата доступу: 20.05.2021

19. HTTPS wiki [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HTTPS – Дата доступу: 21.05.2021

20. Denial-of-service attack wiki [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Denial-of-service_attack – Дата доступу: 21.05.2021

21. DoS/DDoS Protection [Електронний ресурс] – Режим доступу до ресурсу: https://help.mikrotik.com/docs/pages/viewpage.action – Дата доступу: 23.05.2021

22. Долікарська допомога при вивихах [Електронний ресурс] – Режим доступу до ресурсу: https://moz.gov.ua/article/health/travmuvannja-vidi-persha-dopomoga-ta-poradi – Дата доступу: 25.05.2021

23. Соціальне значення охорони праці [Електронний ресурс] – Режим доступу до ресурсу: https://oppb.com.ua/news/socialne-ta-ekonomichne-znachennya-ohorony-praci – Дата доступу: 25.05.2021

24. Загальні вимоги безпеки з охорони праці для користувачів ПК Режим доступу до ресурсу: https://uteka.ua/ua/publication/special-24-formy-ta-systemy-oplaty-praci-127-instrukciya-po-oxrane-truda-pri-rabote-na-personalnom-kompyutere-obrazec – Дата доступу: 25.05.2021

ДОДАТКИ

Додаток А

Код серверного модуля main.py

Лістинг файлу

```
import os
import re
import uuid
import shutil
import docker
import threading
import subprocess

from flask import Flask, request, Response, jsonify
from werkzeug.utils import secure_filename

WORK_FOLDER = '/root/atex_work'
VENV_FOLDER = 'venv'
FILES_FOLDER = 'files'

WORK_DIR_MOUNT_POINT = '/atex_work'
DOCKER_IMAGE_TAG = 'atex-python'

docker_client = docker.from_env()

exec_result = {}

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 # 16 MB

def create_folder(path):
```

```
    if not os.path.exists(path):
        os.makedirs(path)

def get_working_dir(uid):
    return os.path.join(WORK_FOLDER, uid)

def get_venv_dir(working_dir):
    return os.path.join(working_dir, VENV_FOLDER)

def get_files_dir(working_dir):
    return os.path.join(working_dir, FILES_FOLDER)

def allowed_file(filename):
    ALLOWED_EXTENSIONS = ['txt', 'py', 'json', 'csv']
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS

def handle_file_get():
    res = {}

    dirs = os.listdir(WORK_FOLDER)
    for uid in dirs:
        files = os.listdir(get_files_dir(get_working_dir(uid)))
        if files:
            # Temporary, should be extended later
            filename = files[0]
            res[uid] = filename

    return jsonify(res)
```

```
def handle_file_delete(uid):
    path = get_working_dir(uid)
    try:
        shutil.rmtree(path)
    except OSError as e:
        print("Error: %s : %s" % (path, e.strerror))
        return "Error"

    return "Success"

def create_folder_structure(uid):
    working_dir = get_working_dir(uid)
    create_folder(working_dir)

    create_folder(get_files_dir(working_dir))

    return working_dir

def handle_file_post():
    uid = str(uuid.uuid4())

    if 'file' not in request.files:
        return 'No file part'

    file = request.files['file']
    if not file or file.filename == '':
        return 'No selected file'
```

```
    if not allowed_file(file.filename):
        return 'Used unallowed filename !'

    working_dir = create_folder_structure(uid)

    filename = secure_filename(file.filename)
    file.save(os.path.join(get_files_dir(working_dir), filename))

    return jsonify(
        filename = filename,
        uid = uid
    )

def handle_execute_get(uid):
    if uid not in exec_result:
        return jsonify(
            result='Could not find result for uid: ' + uid,
        )
    return jsonify(result=exec_result[uid])

def start_container(name, working_dir):
    working_dir_mount = docker.types.Mount(
        target=WORK_DIR_MOUNT_POINT,
        source=working_dir,
        type='bind'
    )

    container = docker_client.containers.run(
        image=DOCKER_IMAGE_TAG,
```

```
        command='bash',
        mounts=[working_dir_mount],
        name = name,
        privileged=False,
        detach=True,
        tty=True,
        stdin_open=True
    )

    return container

def run_in_container(container, command):
    ret, outputs = container.exec_run(
        cmd=command,
        stdout=True,
        stderr=True,
        stdin=True,
        demux=True
    )

    stdout_bytes = outputs[0]
    stderr_bytes = outputs[1]

    stdout = str()
    if stdout_bytes:
        stdout = stdout_bytes.decode('utf-8')

    stderr = str()
    if stderr_bytes:
```

```
            stderr = stderr_bytes.decode('utf-8')

        print('For ' + command)
        print('Ret: ' + str(ret))
        print('Stdout: ' + stdout)
        print('Stderr: ' + stderr)
        return ret, stdout, stderr

    def run_python_in_container(container, main_file):
        main_file_path =
os.path.join(get_files_dir(WORK_DIR_MOUNT_POINT), main_file)
        command = 'python3 ' + main_file_path

        ret, stdout, stderr = run_in_container(container, command)

        return stdout, stderr

    def create_venv_in_container(container, venv_path):
        command = 'python3 -m venv ' + venv_path

        return run_in_container(container, command)

    def try_install_package_in_container(container, package_name):
        command = 'pip3 install ' + package_name

        return run_in_container(container, command)

    def exec_thread(main_file, uid):
        working_dir = get_working_dir(uid)
```

```
        container_name = "run_" + uid
        container = start_container(container_name, working_dir)

        venv_dir = get_venv_dir(WORK_DIR_MOUNT_POINT)
        ret, stdout, stderr = create_venv_in_container(container, venv_dir)
        if ret != 0:
            container.remove(force=True)
            yield "Filed to create venv inside container ret: " + str(ret) + " stderr: "
+ stderr + "\n"
            return

        while True:
            yield "Running " + main_file + "\n"
            output, error = run_python_in_container(container, main_file)

            yield "Process output: " + output + "\n"
            yield "Process errors: " + error + "\n"

            missing_dependency_list = re.findall(r"ModuleNotFoundError: No
module named '(\w*)'", error)
            if not missing_dependency_list:
                break

            missing_dependency = missing_dependency_list[0]
            yield "Try to satisfy dependency: '" + missing_dependency + "'\n"
            ret, stdout, stderr = try_install_package_in_container(container,
missing_dependency)
```

```
            if ret != 0:
                yield "Failed to install '" + missing_dependency + "'\n"
                break

            yield "Successfully installed '" + missing_dependency + "'\n"


    container.remove(force=True)

def handle_execute_post(uid, main_file):
    return Response(exec_thread(main_file, uid), mimetype='text/plain')

@app.route('/file_list', methods=['GET'])
def file_list():
    return handle_file_get()

@app.route('/file', methods=['POST'])
def file():
    return handle_file_post()

@app.route('/file/<uid>', methods=['DELETE'])
def file_delete(uid):
    return handle_file_delete(uid)

@app.route('/execute/<uid>', methods=['GET'])
def execute_get(uid):
   return handle_execute_get(uid)

@app.route('/execute/<uid>/<main_file>', methods=['POST'])
```

```
def execute_post(uid, main_file):
    return handle_execute_post(uid, main_file)

def build_docker_image(dockerfile_dir_path):
    docker_client.images.build(
        path=dockerfile_dir_path,
        tag=DOCKER_IMAGE_TAG,
    )

if __name__ == '__main__':
    create_folder(WORK_FOLDER)
    build_docker_image('.')

    app.debug = True
    app.run('0.0.0.0', 25555)
```

Додаток Б

Код клієнтського інтерфейсу головного екрана

Лістинг файлу

```
import 'dart:convert';

import 'package:atex/screens/output_screen.dart';
import 'package:file_picker/file_picker.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final baseUrl = 'http://5.58.208.122:25565';

  final colors = [
    Colors.red,
    Colors.pink,
    Colors.purple,
    Colors.deepPurple,
    Colors.indigo,
    Colors.blue,
    Colors.lightBlue,
    Colors.cyan,
    Colors.teal,
    Colors.green,
```

```
  Colors.lightGreen,
  Colors.lime,
  Colors.yellow,
  Colors.amber,
  Colors.orange,
  Colors.deepOrange,
  Colors.brown,
];

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Atex Client'),
    ),
    body: FutureBuilder<Map<String, String>>(
      future: loadFiles(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return Center(
            child: CircularProgressIndicator(),
          );
        } else if (snapshot.error != null) {
          return Center(
            child: Text('Cound not connect to server'),
          );
        } else if (snapshot.data.length == 0) {
          return Center(
            child: Text('Select file to execute'),
```

```
  );
}

List<Widget> children = [];

children.add(Card(
  elevation: 2,
  margin: EdgeInsets.only(left: 0, right: 0, top: 0, bottom: 6.0),
  shape: RoundedRectangleBorder(),
  child: Center(
    child: Padding(
      padding: const EdgeInsets.all(12.0),
      child: Text(
        'Select file to execute',
        style: TextStyle(fontSize: 18.0),
      ),
    ),
  ),
));

snapshot.data.forEach((key, value) {
  var colorIndex = int.parse(
    key.substring(key.length - 3, key.length - 1),
    radix: 16,
  );

  var color = colors[colorIndex % colors.length];

  children.add(Dismissible(
```

```
    key: Key(key),
    onDismissed: (direction) async {
      await http.delete('$baseUrl/file/$key');
      onDone();
    },
    child: Card(
      child: ListTile(
        leading: CircleAvatar(
          backgroundColor: color,
          foregroundColor: Colors.white,
          child: Icon(
            Icons.description,
          ),
        ),
        title: Text(value),
        subtitle: Text(key),
        trailing: IconButton(
          icon: Icon(Icons.play_arrow),
          onPressed: () => onFilePress(key, value),
        ),
      ),
    ),
  ));
});

children.add(SizedBox(
  height: 64.0 + 16.0,
));
```

```
        return ListView(
          children: children,
        );
      },
    ),
    floatingActionButton: FloatingActionButton(
      child: Icon(Icons.file_upload),
      onPressed: onFileUploadPress,
    ),
  );
}

Future<Map<String, String>> loadFiles() async {
  var response = await http.get('$baseUrl/file_list');

  return Map.castFrom(jsonDecode(response.body));
}

void onFilePress(String uid, String filename) {
  Navigator.of(context).push(MaterialPageRoute(
    builder: (context) => OutputScreen(uid, filename, onDone),
  ));
}

Future<void> onFileUploadPress() async {
  var file = await FilePicker.getFile();

  var filename = file.path.split("/")?.last;
  var fileData = await file.readAsString();
```

```
    var request = http.MultipartRequest('POST', Uri.parse('$baseUrl/file'));
    request.files.add(
      http.MultipartFile.fromString(
        'file',
        fileData,
        filename: filename,
      ),
    );

    var response = await request.send();
    var responseData = jsonDecode(await response.stream.bytesToString());

    var uid = responseData['uid'];

    Navigator.of(context).push(MaterialPageRoute(
      builder: (context) => OutputScreen(uid, filename, onDone),
    ));
  }

  void onDone() {
    setState(() {});
  }
}
```

Додаток В

Код клієнтського інтерфейсу відображення результатів

Лістинг файлу

```
import 'dart:convert';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
class OutputScreen extends StatefulWidget {
  final String uid;
  final String filename;
  final VoidCallback onDone;
  OutputScreen(
    this.uid,
    this.filename,
    this.onDone,
  );
  @override
  _OutputScreenState createState() => _OutputScreenState();
}
class _OutputScreenState extends State<OutputScreen> {
  final baseUrl = 'http://5.58.208.122:25565';
  String output = 'Executing file...\n';
  @override
  void initState() {
    super.initState();
    executeFile();}
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
            title: Text('Execution Output'),
          ),
          body: Text(output),
        );}
      void executeFile() async {
        var client = http.Client();
        var executeRequest = http.Request(
          'POST',
Uri.parse('$baseUrl/execute/${widget.uid}/${widget.filename}'));
        var streamedResponse = await client.send(executeRequest);
        var streamSubscription =
          streamedResponse.stream.transform(utf8.decoder).listen((String
value) {
          print(value);
          setState(() {
            output += value;
          });
        });
        streamSubscription.onDone(() {
          setState(() {
            output += "DONE !";
          });
          streamSubscription.cancel();
        });
        widget.onDone();
      }}
```