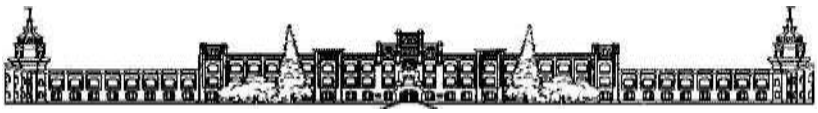

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №9

Тема: РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Варіант 14

Виконав

студент групи ІА – 24:

Любченко І.М

Перевірив:

Мягкий М. Ю

Київ 2024

Зміст

Мета:

метою виконання лабораторної роботи є вивчення та практичне застосування взаємодії клієнта та сервера, таких як РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED
ARCHITECTURE при розробці корпоративних додатків. Ця лабораторна робота спрямована на розвиток навичок у використанні різних звязків для вирішення задач у розробці програмного забезпечення.

Теоретичні відомості

Клієнт-серверна архітектура (Client-Server)

Клієнт-серверна архітектура є одним із найбільш поширених підходів до побудови додатків. У цій архітектурі виділяється дві основні компоненти:

- Клієнт: Це додаток, що використовується кінцевим користувачем. Клієнтська частина відповідає за взаємодію з користувачем, збирання даних та відправлення їх на сервер для обробки. У нашому випадку, клієнтська частина реалізована на HTML та JavaScript.

- Сервер: Це додаток або набір додатків, які обробляють запити від клієнтів, виконують основну бізнес-логіку, зберігають дані в базі даних та повертають результати клієнтам. У нашому випадку, серверна частина реалізована на Flask (Python).

Основні переваги клієнт-серверної архітектури:

- Централізація даних і логіки: Сервер відповідає за обробку та зберігання даних, що дозволяє централізувати контроль над ними.
- Масштабованість: Сервер може бути масштабований для обробки великої кількості запитів від багатьох клієнтів.
- Безпека: Зменшується ризик втрати або зловмисної зміни даних, оскільки вони зберігаються на сервері, а не на клієнтських пристроях.

Архітектура "peer-to-peer" (P2P)

Архітектура "peer-to-peer" (P2P) відрізняється від клієнт-серверної тим, що кожен вузол мережі виконує функції як клієнта, так і сервера. У цій моделі кожен учасник (peer) може запитувати ресурси у інших учасників і надавати свої ресурси для інших.

Переваги P2P архітектури:

- Децентралізація: Відсутність центрального сервера, що зменшує ризик єдиного місця відмови.
- Розподіленість ресурсів: Розподіл обчислювальних потужностей і зберігання даних між учасниками мережі.
- Стійкість до збоїв: Висока стійкість до збоїв, оскільки відмова одного учасника не впливає на загальну роботу системи.

Недоліки P2P архітектури:

- Складність керування: Ускладнене управління і підтримка через відсутність централізованого контролю.
- Безпека: Високий ризик зловживань і атак, оскільки кожен учасник може потенційно бути небезпечним.

Сервіс-орієнтована архітектура (SOA)

Сервіс-орієнтована архітектура (SOA) побудована навколо концепції сервісів, які є окремими, автономними компонентами, що виконують певні функції і можуть взаємодіяти між собою через стандартизовані інтерфейси (наприклад, через веб-сервіси або API).

Переваги SOA:

- Модульність: Легке розширення і модифікація системи за рахунок додавання нових сервісів або зміни існуючих.
- Повторне використання: Сервіси можуть бути повторно використані в різних додатках або бізнес-процесах.
- Інтероперабельність: Можливість інтеграції з іншими системами і сервісами через стандартизовані протоколи.

Недоліки SOA:

- Складність реалізації: Вимагає ретельного планування та розробки, а також використання стандартизованих протоколів і інтерфейсів.
- Високі вимоги до продуктивності: Може викликати додаткові затримки в комунікації між сервісами, що впливає на продуктивність системи.

Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.

3. Реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

Тема: Архіватор

Чому клієнт-серверна архітектура краще за P2P для архіватора:

1. Головний аргумент - надійність доступу до даних:

- В P2P архіві файли розподілені між користувачами і доступні лише коли вони онлайн

- Якщо користувач з потрібним файлом офлайн - отримати файл неможливо

- На сервері файли доступні постійно, 24/7, незалежно від інших користувачів

2. Важливий аспект - цілісність даних:

- В P2P складно гарантувати, що файл не був модифікований іншими вузлами

- Сервер забезпечує єдину точку контролю версій та змін файлів

- Легко відстежити хто і коли вносив зміни

3. Практичний момент - простота розробки та підтримки:

- P2P вимагає складної логіки синхронізації між вузлами

- Потрібно вирішувати проблеми з NAT, брандмауерами тощо

- Клієнт-серверна архітектура має чітку і зрозумілу структуру

4. Критично важлива безпека:

- В P2P кожен вузол потенційно може бути зловмисним

- Неможливо повністю контролювати, хто має доступ до даних

- Сервер дозволяє централізовано керувати правами та доступом

5. Підтвердження на практиці:

- Більшість успішних архіваторів (Dropbox, Google Drive) використовують клієнт-серверну модель

- Це підтверджує її ефективність для даного випадку

Отже, хоча P2P має свої переваги для інших задач (як BitTorrent для розповсюдження файлів), для архіватора клієнт-серверна архітектура надає кращу надійність, безпеку та простоту реалізації, що доведено успішними прикладами.

Реалізація клієнт-серверної архітектури

Для реалізації архіватора я обрав клієнт-серверну архітектуру з наступних причин:

1. Централізація даних: Всі архіви і файли зберігаються на сервері, що забезпечує їхню безпеку і централізований контроль.
2. Розвантаження клієнтських пристроїв: Основна обробка даних і бізнес-логіка виконуються на сервері, що дозволяє використовувати менш потужні клієнтські пристрої.
3. Масштабованість: Сервер може бути масштабований для обробки великої кількості запитів від різних клієнтів, забезпечуючи високу продуктивність і надійність системи.
4. Безпека: Централізоване зберігання даних на сервері зменшує ризик втрати або зловмисної зміни даних.

backend > app.py > ...

```python
import sys
import os
from datetime import datetime
from flask import Flask, request, jsonify, render_template, redirect, url_for

sys.path.append(os.path.join(os.path.dirname(__file__), '..', 'src'))

from archive_handler import ArchiveHandler
from database_manager import DatabaseManager

app = Flask(__name__)
db_manager = DatabaseManager("archiver.db")
archive_handler = ArchiveHandler(db_manager)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/create_archive', methods=['GET', 'POST'])
def create_archive():
    if request.method == 'POST':
        try:
            data = request.json
            archive_name = data.get('archive_name')
            files = data.get('files')
            if not archive_name or not files:
                return jsonify({"error": "Invalid input"}), 400
            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            archive_id = archive_handler.create_archive(archive_name, files, timestamp)
            return jsonify({"message": "Archive created", "archive_id": archive_id})
        except Exception as e:
            print(f"Error: {e}")
            return jsonify({"error": f"An error occurred during archive creation: {e}"}), 500
    return render_template('create_archive.html')

@app.route('/extract_archive', methods=['GET', 'POST'])
def extract_archive():
    if request.method == 'POST':
        archive_name = request.form['archive_name']
        destination_folder = request.form['destination_folder']
        if not archive_name or not destination_folder:
            return jsonify({"error": "Invalid input"}), 400
        archive_handler.extract_archive(archive_name, destination_folder)
        return redirect(url_for('home'))
    return render_template('extract_archive.html')

@app.route('/show_database')
def show_database():
```

```python
@app.route('/show_database')
def show_database():
    archives = db_manager.get_archives()
    db_info = ""
    for archive in archives:
        archive_id, archive_name, timestamp = archive
        db_info += f"Архів: {archive_name} (створено: {timestamp})\n"
        files = db_manager.get_files_in_archive(archive_id)
        for file in files:
            db_info += f"  Файл: {file[0]}\n"
    return render_template('show_database.html', database_info=db_info)

@app.route('/clear_database', methods=['POST'])
def clear_database():
    try:
        db_manager.clear_database()
        return redirect(url_for('show_database'))
    except Exception as e:
        print(f"Error: {e}")
        return jsonify({"error": "An error occurred during database clearing"}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Рис 1-2. - файл відповідає за обробку запитів від клієнтської частини та взаємодію з бізнес-логікою.

backend > database_manager.py > ...

```python
import sqlite3

class DatabaseManager:
    def __init__(self, db_name="archiver.db"):
        self.db_name = db_name
        self.create_tables()
        self.migrate_data()

    def create_connection(self):
        connection = sqlite3.connect(self.db_name)
        cursor = connection.cursor()
        return connection, cursor

    def create_tables(self):
        connection, cursor = self.create_connection()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS archives_new (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                timestamp TEXT NOT NULL
            )
        """)
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS files (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                archive_id INTEGER,
                path TEXT NOT NULL,
                FOREIGN KEY (archive_id) REFERENCES archives_new (id)
            )
        """)
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS notifications (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                message TEXT NOT NULL,
                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
            )
        """)
        connection.commit()
        connection.close()

    def migrate_data(self):
        connection, cursor = self.create_connection()
        cursor.execute("""
            INSERT INTO archives_new (id, name, timestamp)
            SELECT id, name, 'N/A' FROM archives
        """)
        cursor.execute("DROP TABLE archives")
        cursor.execute("ALTER TABLE archives_new RENAME TO archives")
        connection.commit()
```

```
50          connection.close()
51
52      def add_archive(self, name, timestamp):
53          connection, cursor = self.create_connection()
54          cursor.execute("INSERT INTO archives (name, timestamp) VALUES (?, ?)", (name, timestamp))
55          connection.commit()
56          archive_id = cursor.lastrowid
57          connection.close()
58          return archive_id
59
60      def add_file(self, archive_id, file_path):
61          connection, cursor = self.create_connection()
62          cursor.execute("INSERT INTO files (archive_id, path) VALUES (?, ?)", (archive_id, file_path))
63          connection.commit()
64          connection.close()
65
66      def add_notification(self, message):
67          connection, cursor = self.create_connection()
68          cursor.execute("INSERT INTO notifications (message) VALUES (?)", (message,))
69          connection.commit()
70          connection.close()
71
72      def get_archives(self):
73          connection, cursor = self.create_connection()
74          cursor.execute("SELECT id, name, timestamp FROM archives")
75          archives = cursor.fetchall()
76          connection.close()
77          return archives
78
79      def get_files_in_archive(self, archive_id):
80          connection, cursor = self.create_connection()
81          cursor.execute("SELECT path FROM files WHERE archive_id = ?", (archive_id,))
82          files = cursor.fetchall()
83          connection.close()
84          return files
85
86      def clear_database(self):
87          connection, cursor = self.create_connection()
88          cursor.execute("DELETE FROM archives")
89          cursor.execute("DELETE FROM files")
90          cursor.execute("DELETE FROM notifications")
91          connection.commit()
92          connection.close()
```

Рис 3-4. – файл відповідає за взаємодію з базою даних.

backend > archive_handler.py > ...

```python
from archiver import Archiver
from database_manager import DatabaseManager
from composite import Composite
from leaf import Leaf

class ArchiveHandler:
    def __init__(self, db_manager):
        self.db_manager = db_manager

    def create_archive(self, archive_name, files, timestamp):
        root_component = Composite("root")
        for file_path in files:
            leaf = Leaf(file_path.split("/")[-1], file_path)
            root_component.add(leaf)
        archive_id = self.db_manager.add_archive(archive_name, timestamp)
        archiver = Archiver(archive_name.split('.')[-1])
        archiver.create_archive(archive_name)
        root_component.archive(archiver, self.db_manager, archive_id)
        return archive_id

    def extract_archive(self, archive_name, destination_folder):
        archiver = Archiver(archive_name.split('.')[-1])
        archiver.extract_archive(archive_name, destination_folder)
```

Рис 5. - файл відповідає за обробку логіки створення та розпакування архівів.(серверна частина)

backend > templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Архіватор</title>
</head>
<body>
    <h1>Архіватор сервер працює!</h1>
    <p>Використовуйте форми нижче для взаємодії з сервером.</p>
    <ul>
        <li><a href="/create_archive">Створити архів</a></li>
        <li><a href="/extract_archive">Розпакувати архів</a></li>
        <li><a href="/show_database">Переглянути базу даних</a></li>
    </ul>
</body>
</html>
```

Рис 6. - файл забезпечує головну сторінку додатка з посиланнями на сторінки створення архіву, розпакування архіву та перегляду бази даних.

backend > templates > <> create_archive.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Створити архів</title>
</head>
<body>
    <h1>Створити архів</h1>
    <form id="create-archive-form">
        <label for="archive_name">Назва архіву:</label>
        <input type="text" id="archive_name" name="archive_name">
        <br>
        <label for="files">Файли:</label>
        <input type="file" id="files" name="files" multiple>
        <br>
        <button type="submit">Створити</button>
    </form>
    <a href="/">Повернутися на головну</a>

    <script>
        document.getElementById('create-archive-form').addEventListener('submit', function(event) {
            event.preventDefault();
            const files = document.getElementById('files').files;
            let filesList = [];
            for (let i = 0; i < files.length; i++) {
                filesList.push(files[i].name);
            }
            const archiveName = document.getElementById('archive_name').value;
            fetch('/create_archive', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ archive_name: archiveName, files: filesList })
            }).then(response => {
                if (response.ok) {
                    alert('Архів успішно створений!');
                    window.location.href = '/';
                } else {
                    response.json().then(data => {
                        alert(`Помилка при створенні архіву: ${data.error}`);
                    });
                }
            }).catch(error => {
                alert(`Помилка при створенні архіву: ${error}`);
            });
        });
    </script>
```

Рис 7. - файл забезпечує форму для створення архіву з вибором файлів та їхньою відправкою на сервер.

backend > templates > <> extract_archive.html > ...

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Розпакувати архів</title>
</head>
<body>
    <h1>Розпакувати архів</h1>
    <form action="/extract_archive" method="post">
        <label for="archive_name">Назва архіву:</label>
        <input type="text" id="archive_name" name="archive_name">
        <br>
        <label for="destination_folder">Папка призначення:</label>
        <input type="text" id="destination_folder" name="destination_folder">
        <br>
        <button type="submit">Розпакувати</button>
    </form>
    <a href="/">Повернутися на головну</a>
</body>
</html>
```

Рис 8. - файл забезпечує форму для розпакування архіву з введенням назви архіву та папки призначення.

backend > templates > <> show_database.html > ...

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>База даних</title>
</head>
<body>
    <h1>База даних</h1>
    <pre>{{ database_info }}</pre>
    <form action="/clear_database" method="post">
        <button type="submit">Очистити базу даних</button>
    </form>
    <a href="/">Повернутися на головну</a>
</body>
</html>
```

Рис 9. - файл забезпечує відображення бази даних з архівами та їхніми файлами, а також дозволяє очищувати базу даних.

```
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab4trpz\backend> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 291-271-312
127.0.0.1 - - [11/Dec/2024 13:50:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [11/Dec/2024 13:50:23] "GET /favicon.ico HTTP/1.1" 404 -
```

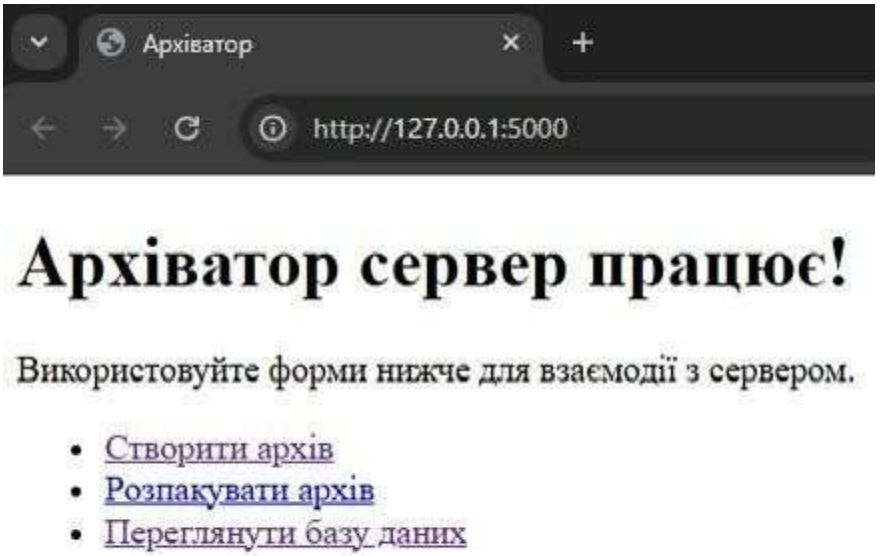

Рис 10-11. – запуск сервера

Рис 12. – Створення архіву

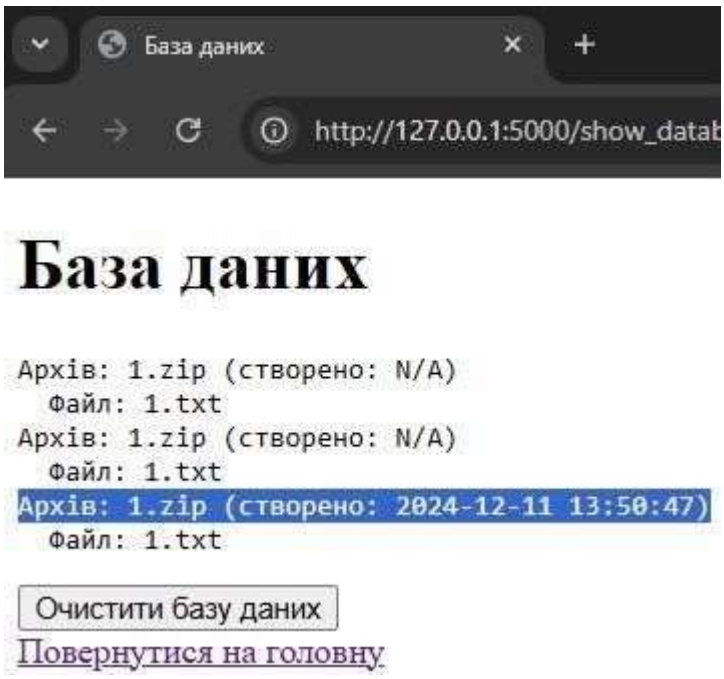

Рис 13. – Оновлення бази даних після створення архіву.

```
 * Debugger is active!
 * Debugger PIN: 291-271-312
127.0.0.1 - - [11/Dec/2024 13:50:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [11/Dec/2024 13:50:23] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [11/Dec/2024 13:50:38] "GET /create_archive HTTP/1.1" 200 -
Creating ZIP archive...
127.0.0.1 - - [11/Dec/2024 13:50:47] "POST /create_archive HTTP/1.1" 200 -
127.0.0.1 - - [11/Dec/2024 13:50:49] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [11/Dec/2024 13:50:53] "GET /show_database HTTP/1.1" 200 -
127.0.0.1 - - [11/Dec/2024 13:51:01] "POST /clear_database HTTP/1.1" 302 -
127.0.0.1 - - [11/Dec/2024 13:51:01] "GET /show_database HTTP/1.1" 200 -
127.0.0.1 - - [11/Dec/2024 13:51:03] "GET / HTTP/1.1" 200 -
```

Рис 14. – Сповіщення на сервері про дії користувача.

Висновок:

У цій лабораторній роботі я реалізував клієнт-серверну архітектуру, використовуючи принцип "тонкий клієнт - товстий сервер". Основна обробка даних та бізнес-логіка виконуються на сервері, а клієнтська частина відповідає за взаємодію з користувачем і передачу даних на сервер.

Принцип взаємодії такий що клієнт-серверна архітектура дозволяє централізувати обробку даних на сервері, забезпечуючи їхню безпеку та контроль. Клієнтська частина, яка реалізована за допомогою HTML і JavaScript, відповідає за збір даних від користувача та їхню передачу на сервер для обробки. Серверна частина, реалізована за допомогою Flask, обробляє запити, виконує бізнес-логіку та взаємодіє з базою даних SQLite для збереження інформації.

Причина вибору саме такої архітектури це централізація даних і логіки забезпечує простоту підтримки та оновлень. Розвантаження клієнтських пристроїв дозволяє використовувати менш потужні пристрої. Масштабованість дозволяє обробляти велику кількість запитів від багатьох клієнтів. Безпека гарантується центральним зберіганням даних на сервері.

Реалізував таким чином що створив кілька основних файлів. Файл app.py обробляє запити від клієнтів, реалізуючи функції створення і розпакування архівів, перегляду і очищення бази даних. Файл database_manager.py відповідає за взаємодію з базою даних, забезпечуючи створення таблиць, додавання і отримання архівів і файлів, а також очищення бази даних. Файл archive_handler.py реалізує бізнес-логіку створення архівів, додаючи файли до архіву і зберігаючи інформацію про архів у базі даних. HTML-шаблони (index.html, create_archive.html, extract_archive.html, show_database.html) забезпечують інтерфейс для взаємодії з користувачем.