



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №9
Технологія розробки програмного забезпечення
*«РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER,
SERVICE-ORIENTED ARCHITECTURE»*
Варіант 28

Виконала:
студентка групи ІА–13
Хижняк Валерія Валеріївна

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Різні види взаємодії додатків: client-server, peer-to-peer, service-oriented architecture.

Мета: ознайомитись з наступними видами взаємодії додатків: client-server, peer-to-peer, service-oriented architecture, реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувати одну з архітектур у власному проєкті.

Хід роботи:

Варіант:

..28 JSON Tool (ENG) (strategy, command, observer, template method, flyweight)

Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Auto save\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

Оскільки у варіанті не вказано яку саме архітектуру потрібно реалізовувати я обрала архітектуру клієнт сервер.

Архітектура клієнт-сервер — це модель розподіленої обчислювальної системи, в якій функції системи розділені між клієнтом (користувачем або клієнтським додатком) і сервером.

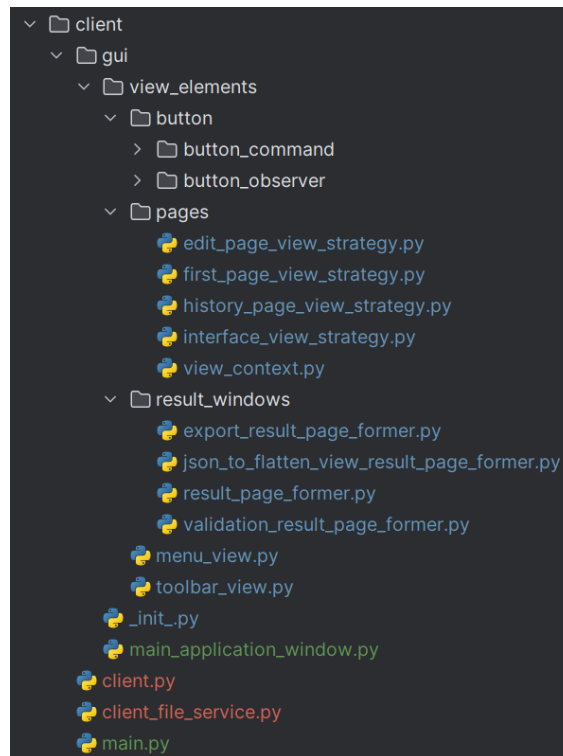
Клієнт: Це додаток або пристрій, який взаємодіє з користувачем і відправляє запити серверу для отримання ресурсів або послуг.

Сервер: Це програмне забезпечення або пристрій, який надає послуги або ресурси клієнтам, обробляючи їх запити.

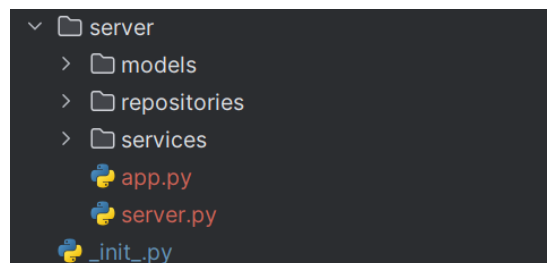
Проект я реалізувала так, що клієнт відповідає за графічне відображення та взаємодію з користувачем, а сервер відповідає за всю бізнес логіку(валідацію, перетворення у flat view, експорт і т.д.)

Структура проекту має наступний вигляд:

Клієнська частина має таку структуру:



Серверна частина виглядає наступним чином:



Взаємодія між клієнтом і сервером відбувається через сокети. Сервер, ініціалізований за певним хостом та портом, створює сокет, прив'язаний до його адреси та прослуховує вхідні з'єднання.

Запуск серверу відбувається в файлі app.py

```
1 from application.server.server import Server
2
3
4 if __name__ == "__main__":
5     server = Server(host="localhost", port=12345)
6     server.start()
```

Створюється об'єкт класу Server та для нього викликається метод start()

```
7  class Server:
8      # Створюємо об'єкт класу JSONSchemaService()
9      json_schema_service = JSONSchemaService()
10
11  def __init__(self, host, port):
12      self.host = host
13      self.port = port
14      self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15
```

В методі start() відбувається створення сервера використовуючи модуль socket та потоки (threading) для обробки одночасно багатьох клієнтських з'єднань. Спочатку відбувається спроба зв'язати сокет з конкретною IP-адресою і портом які ініціалізуються в конструкторі, потім сервер починає слухати вхідні з'єднання. Сервер очікує і приймає нове з'єднання. accept() блокує виконання коду, поки не буде отримано нове з'єднання, створюється новий потік для обробки клієнта.

```
16  def start(self):
17      try:
18          with self.socket as s:
19              s.bind((self.host, self.port))
20              s.listen(5)
21              print(f"Server listening on {self.host}:{self.port}")
22              while True:
23                  client_socket, addr = s.accept()
24                  print(f"Accepted connection from {addr}")
25                  client_handler = threading.Thread(target=self.handle_client, args=(client_socket,))
26                  client_handler.start()
27      except KeyboardInterrupt:
28          print("Server terminated by user.")
29      except Exception as e:
30          print(f"Error in start method: {str(e)}")
31          traceback.print_exc()
```

Метод handle_client() отримує дані від клієнта, декодує їх з формату UTF-8. Розділяє отримані дані на три частини за допомогою символу |. В залежності від отриманого типу запиту викликається відповідний метод для обробки запиту:

Якщо request_type == "VALIDATE", викликається метод validate_json.

Якщо request_type == "FLATTEN_VIEW", викликається метод json_to_flatten_view.

Якщо request_type == "OPEN_FILE", викликається метод open_json_file.

Якщо `request_type == "SAVE_FILE"`, викликається метод `save_json_file`.

Якщо `request_type == "EXPORT_MARKDOWN"`, викликається метод `export_json_as_markdown_table`.

Відправляє оброблені дані або відповідь клієнту, попередньо закодовані у форматі UTF-8.

```
def handle_client(self, client_socket):
    try:
        data = client_socket.recv(1024).decode("utf-8")
        # Отримання типу запиту та даних від клієнта
        request_type, data, path = data.split("|", 2)
        # Логіка обробки запиту в залежності від типу
        if request_type == "VALIDATE":
            state, response = self.validate_json(data)
        elif request_type == "FLATTEN_VIEW":
            response = self.json_to_flatten_view(data)
        elif request_type == "OPEN_FILE":
            response = self.open_json_file()
        elif request_type == "SAVE_FILE":
            response = self.save_json_file(data, path)
        elif request_type == "EXPORT_MARKDOWN":
            response = self.export_json_as_markdown_table(data)
        # Відправка відповіді клієнту
        client_socket.sendall(response.encode("utf-8"))
    except Exception as e:
        # Зареєструвати деталі винятку
        print(f"Помилка у функції handle_client: {str(e)}")
        traceback.print_exc() # Вивести повний стек винятку для отримання повної
інформації

    # Відправити відповідь із помилкою клієнту
    error_response = "Помилка: щось пішло не так на сервері"
    client_socket.sendall(error_response.encode('utf-8'))
    finally:
        # Закриття сокету у будь-якому випадку
        client_socket.close()
```

Сторона клієнта має наступний вигляд:

В класі `Client` в методі `send_request()` відбувається з'єднання з хостом через сокет. Потім після перевірки з'єднання дані відправляються на сервер та отримується відповідь від серверу.

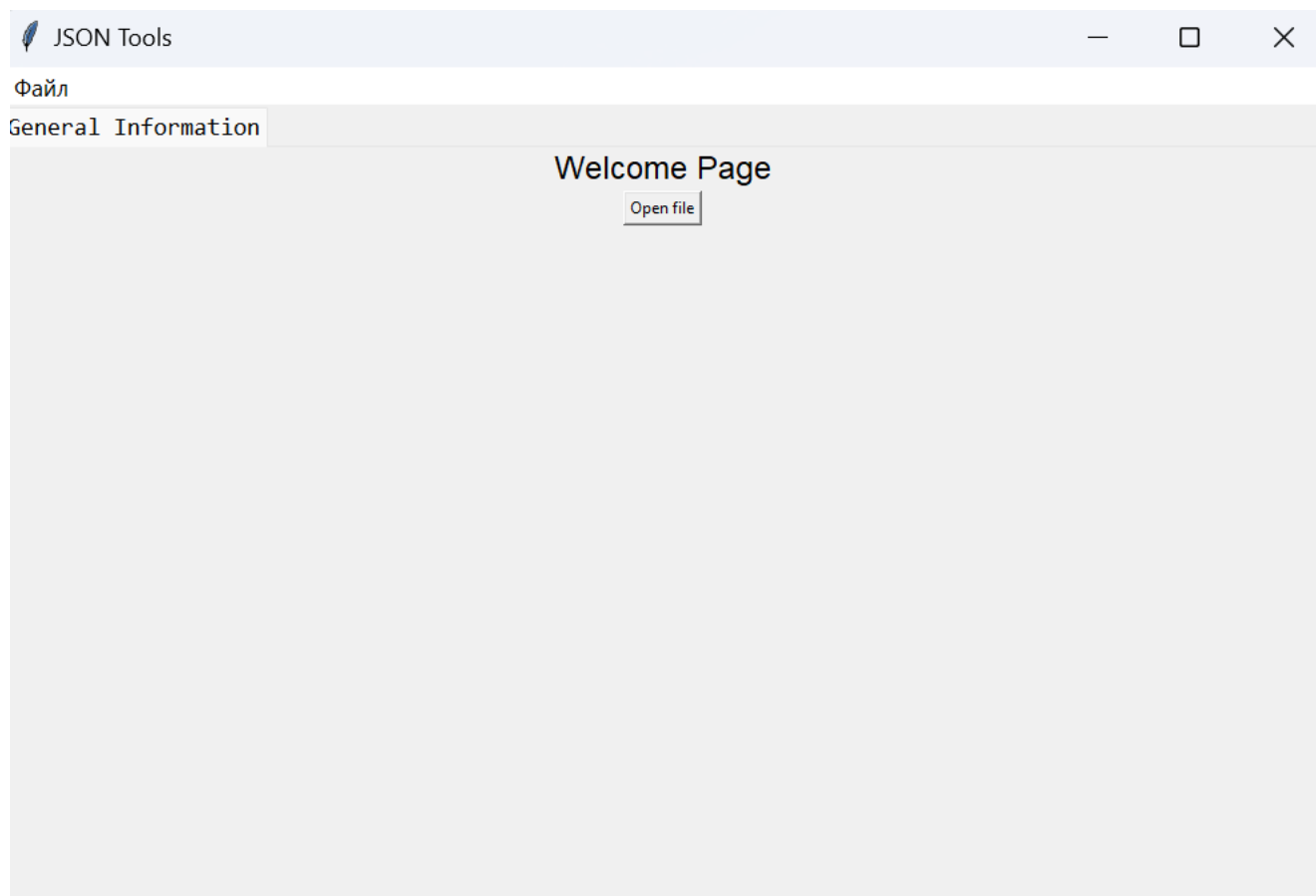
```

22     def send_request(self, data):
23         try:
24             with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
25                 # Встановлення з'єднання
26                 client_socket.connect(("localhost", 12345))
27                 # Перевірка з'єднання перед відправкою даних
28                 if client_socket.fileno() == -1:
29                     raise ConnectionError("Connection failed")
30                 # Відправлення даних
31                 client_socket.send(data.encode('utf-8'))
32                 # Отримання відповіді
33                 response = client_socket.recv(1024).decode('utf-8')
34                 return response
35         except Exception as e:
36             return f"Error: {str(e)}"

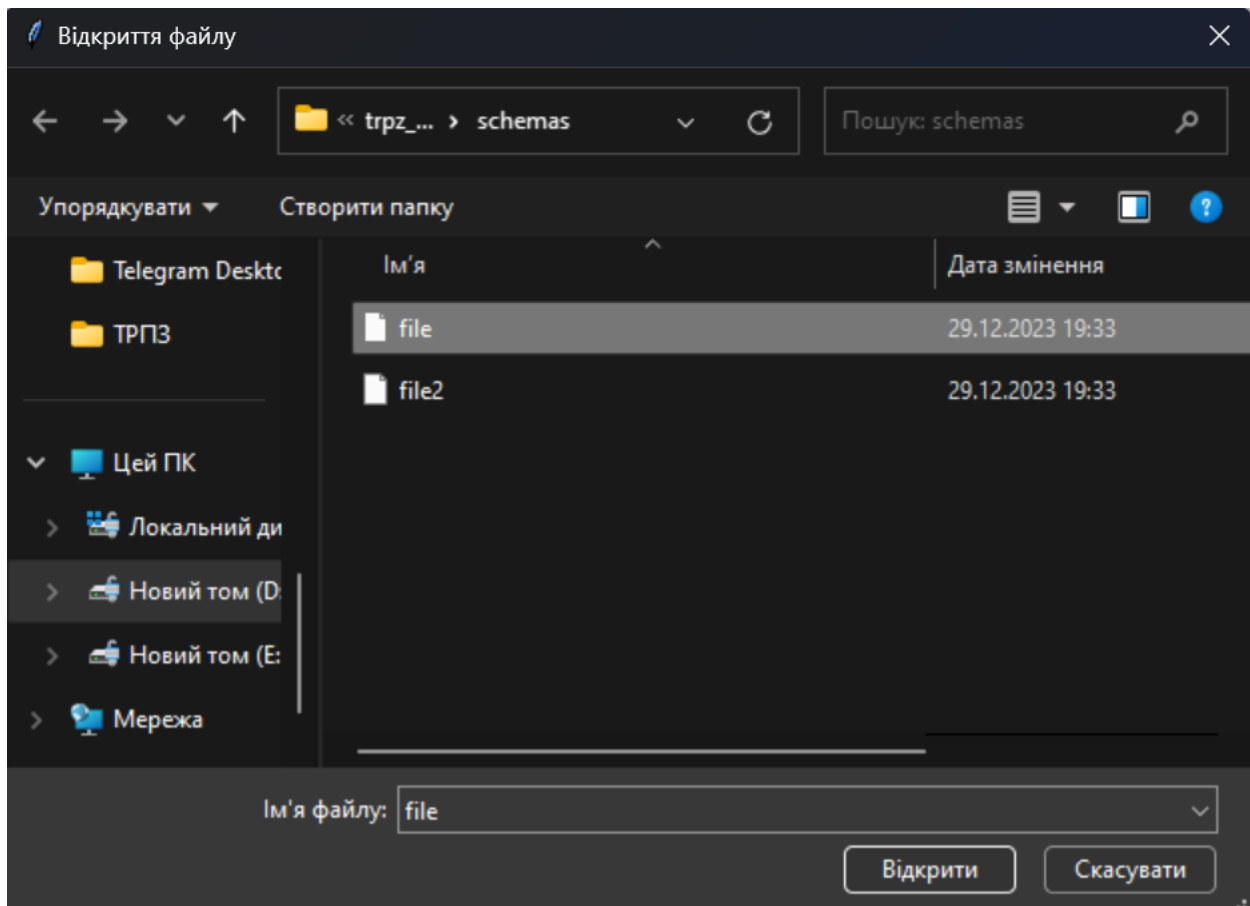
```

Результати роботи коду:

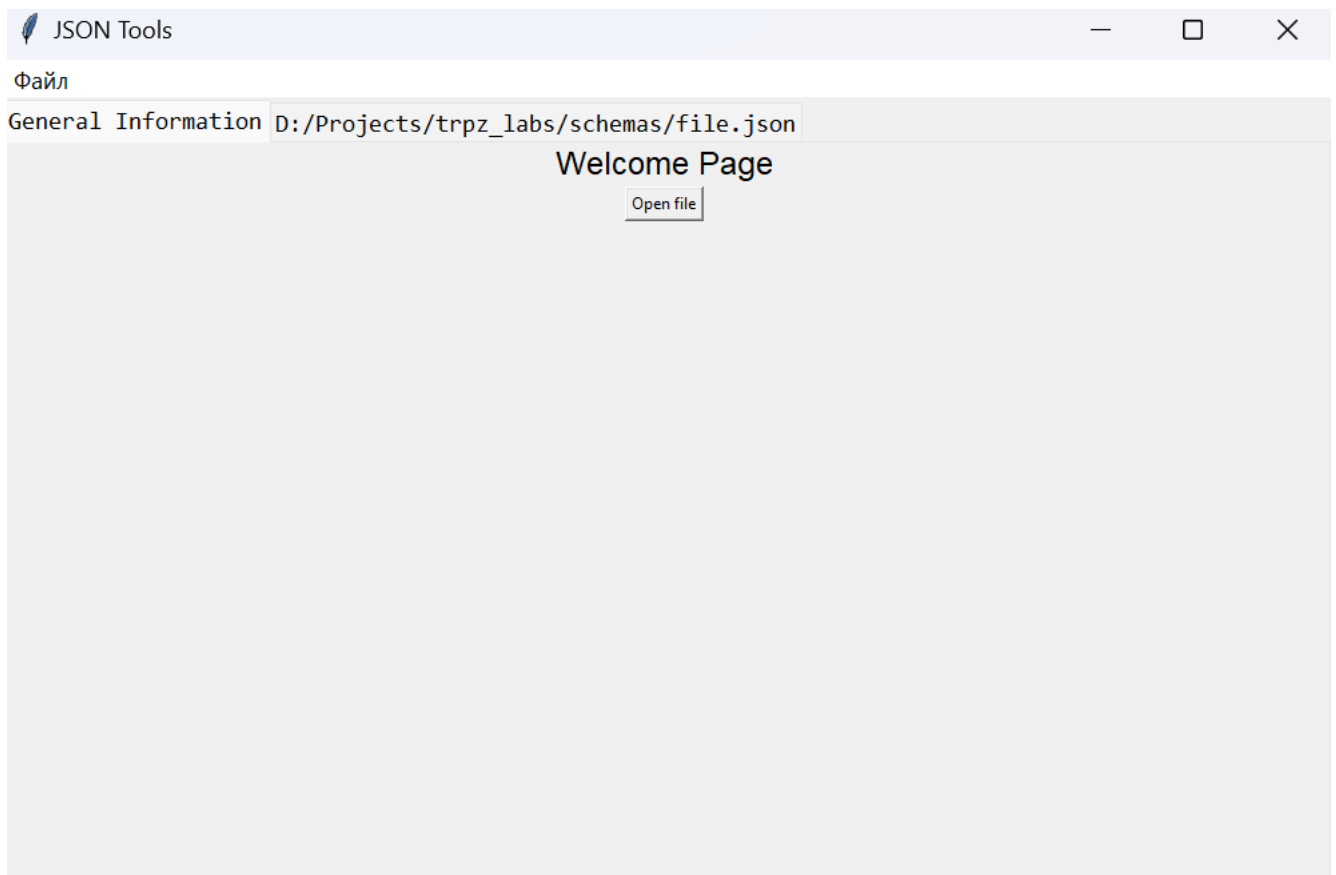
При запуску застосунку користувач бачить початкову сторінку програми:



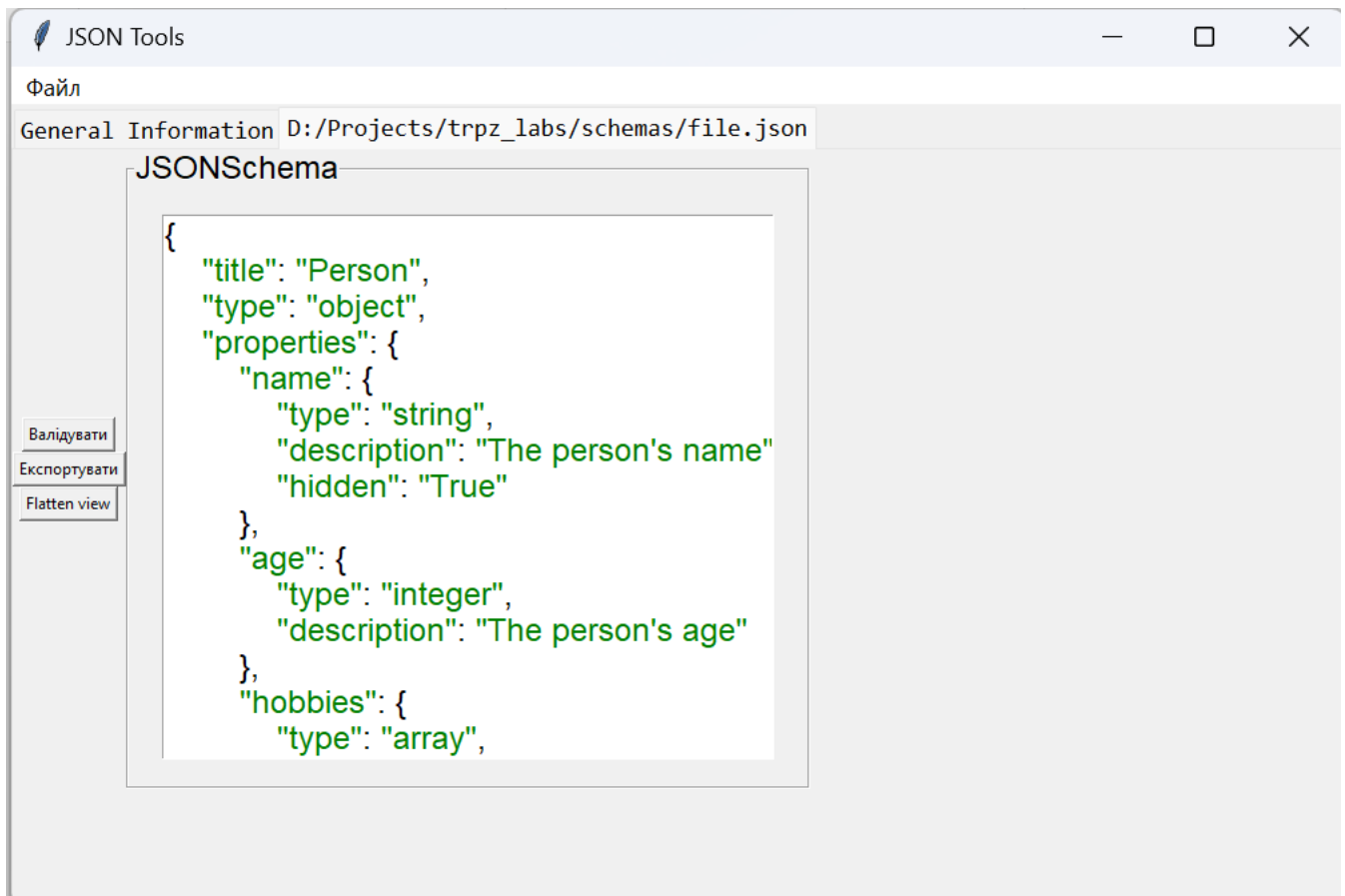
Для початку роботи необхідно відкрити файл типу JSON за допомогою кнопки Open file або через меню – файл – Відкрити. При натисненні на один із цих варіантів відкриття JSON виводиться вікно, де користувач може обрати необхідний файл за необхідним шляхом:



Після відкриття з'явиться вкладка цього файлу, куди можна перейти:



Вкладка має наступний вигляд:



Як можна побачити, виводиться JSON у форматуванні та з підсвіткою, при цьому файл можна редагувати та в реальному часі користуватись панеллю інструментів, яка знаходиться зліва та має 3 кнопки: Валідувати, Експортувати, Flatten view.

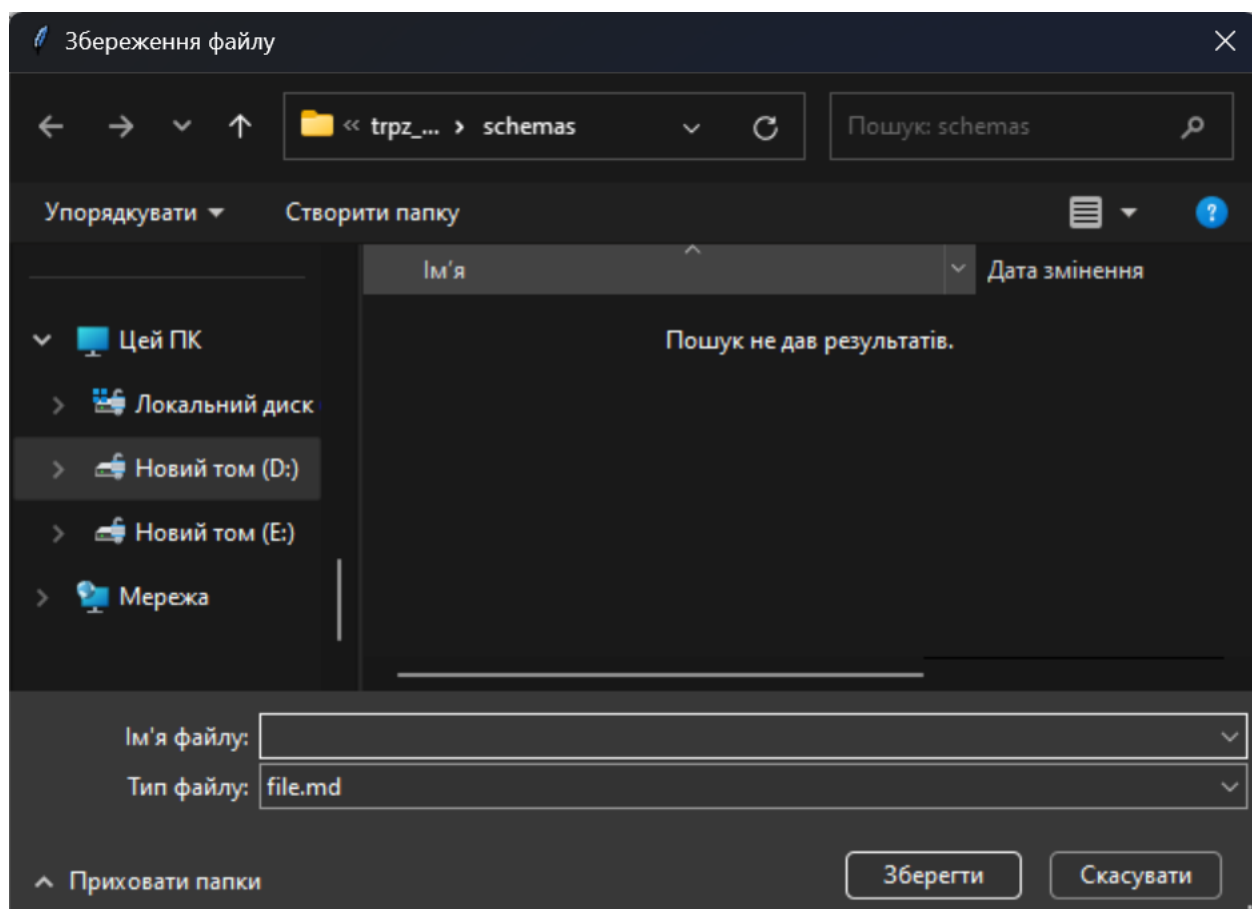
Для перевірки функціональності натиснемо відповідні кнопки, при цьому файл, для якого будемо їх натискати на скріншотах.

Результат валідації:

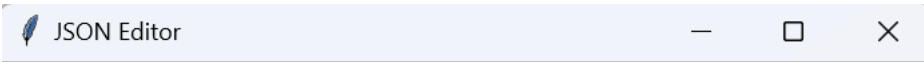


Результат експорту:

Спочатку вибір, куди зберегти таблицю Markdown:






Вікно сповіщення про успішний експорт:



Export markdown_table completed

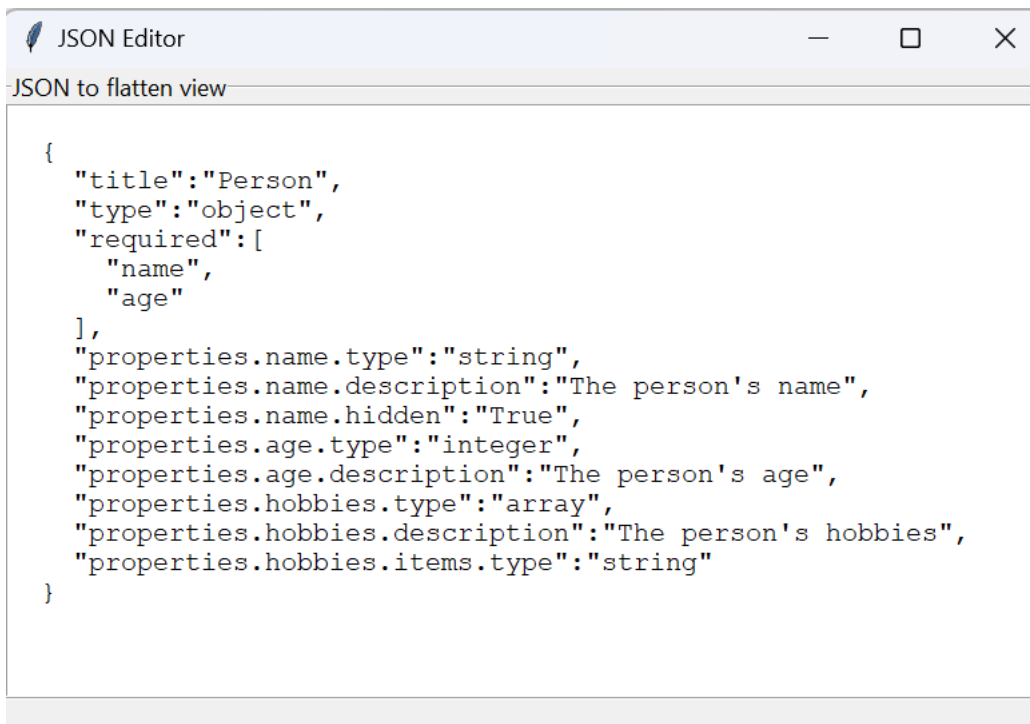
Директорія, де було збережено таблицю:

Ім'я	Дата змінення	Тип	Розмір
 file	29.12.2023 19:33	JSON Source File	1 K
 file2	29.12.2023 19:33	JSON Source File	1 K
 test	30.12.2023 7:37	Markdown Source File	2 K

Перегляд збереженого файлу:

Key	Value
title	Person
type	object
properties	{'name': {'type': 'string', 'description': 'The person's name', 'hidden': 'True'}, 'age': {'type': 'integer', 'description': 'The person's age'}, 'hobbies': {'type': 'array', 'description': 'The person's hobbies', 'items': {'type': 'string'}}}
required	['name', 'age']

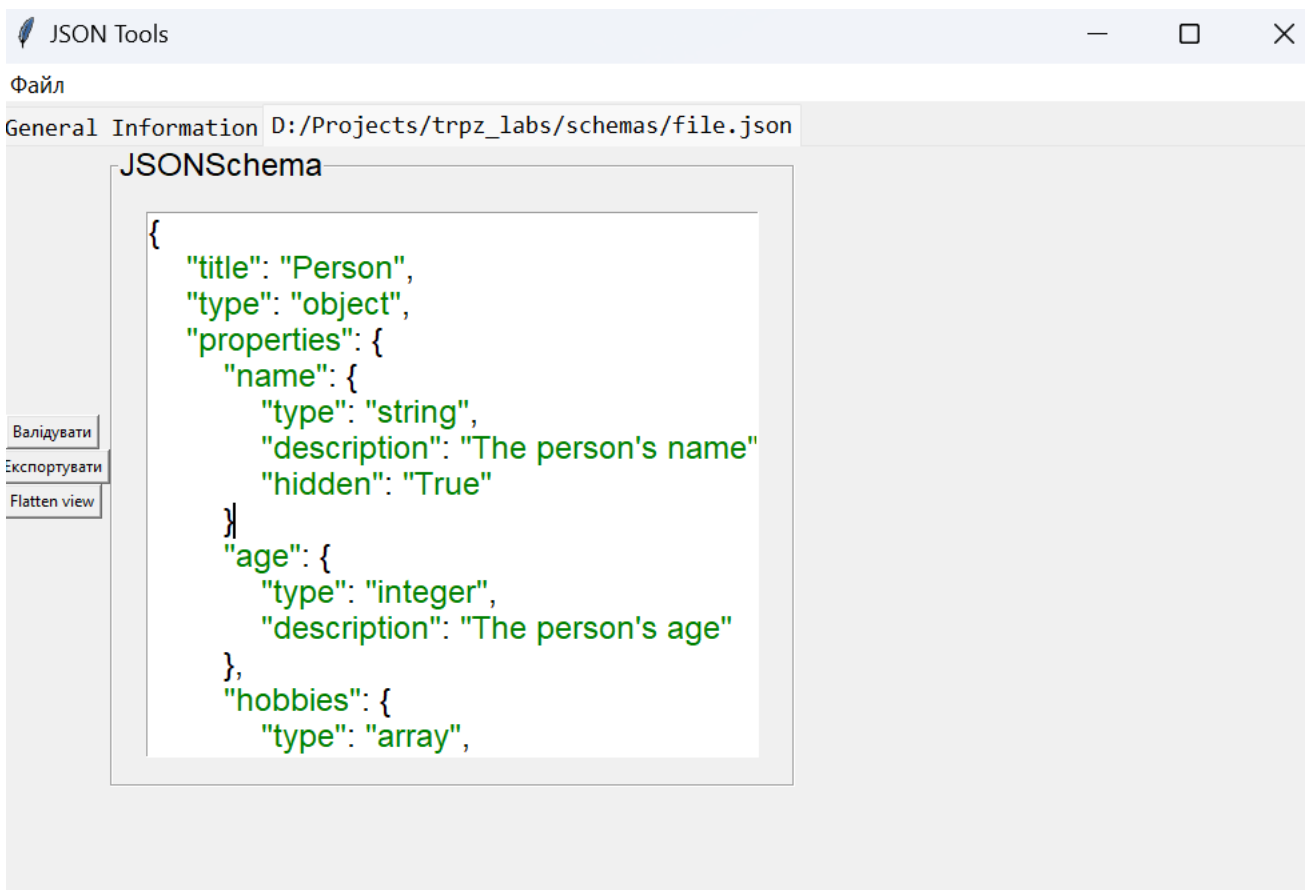
Результат перетворення у плоский вигляд:



The image shows a window titled "JSON Editor" with a standard macOS-style title bar (minimize, maximize, close buttons). Below the title bar is a tab labeled "JSON to flatten view". The main area of the window contains a JSON object representing a schema for a person. The JSON is as follows:

```
{
  "title": "Person",
  "type": "object",
  "required": [
    "name",
    "age"
  ],
  "properties": {
    "name": {
      "type": "string",
      "description": "The person's name",
      "hidden": "True",
      "age": {
        "type": "integer",
        "description": "The person's age",
        "hobbies": {
          "type": "array",
          "description": "The person's hobbies",
          "items": {
            "type": "string"
          }
        }
      }
    }
  }
}
```

Для перевірки на валідацію, змінимо файл:



The image shows a window titled "JSON Tools" with a standard macOS-style title bar. Below the title bar is a menu bar with "Файл" (File). Below the menu bar is a tab labeled "General Information" and a file path "D:/Projects/trpz_labs/schemas/file.json". The main area of the window is titled "JSONSchema" and contains a JSON object representing a schema for a person. The JSON is as follows:

```
{
  "title": "Person",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "The person's name",
      "hidden": "True"
    },
    "age": {
      "type": "integer",
      "description": "The person's age"
    },
    "hobbies": {
      "type": "array",

```

On the left side of the window, there are three buttons: "Валідувати" (Validate), "Експортувати" (Export), and "Flatten view". The "Валідувати" button is highlighted.

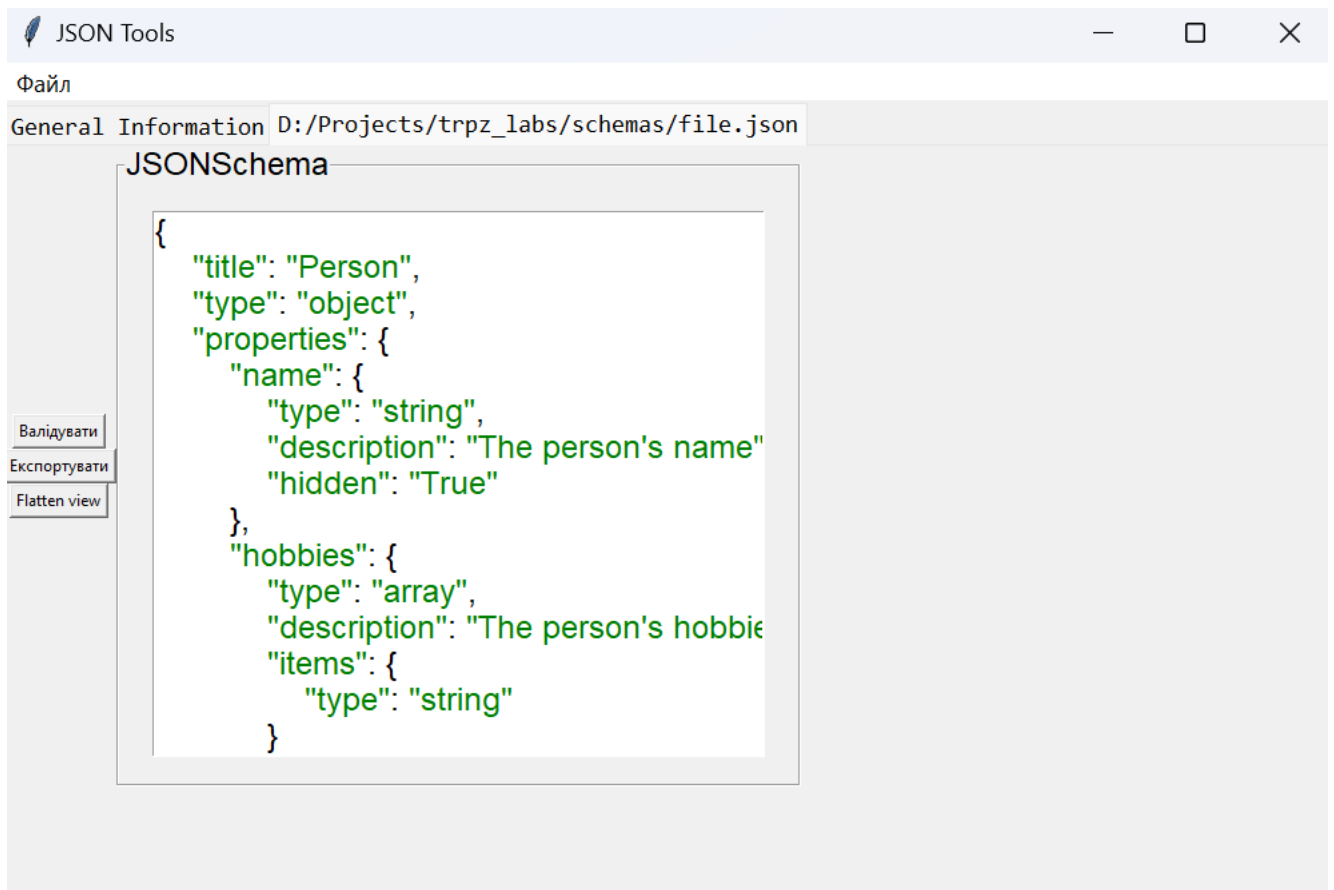
Було видалено кому, тож JSON тепер не повинен бути валідним:

```
JSON Editor
JSON is not valid. Error: Expecting ',' delimiter: line 10
column 9 (char 210)
```

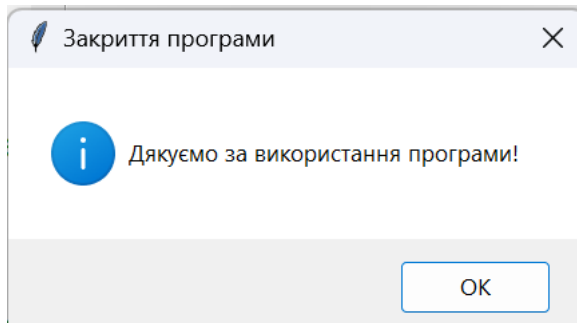
Як бачимо, функціональність працює, вивід місця помилки теж. При спробі зробити плоский вигляд невалідного JSON також буде виведено помилку:

```
JSON Editor
JSON to flatten view
JSON is not valid.: Expecting ',' delimiter: line 9 column 10 (char 201) Can't make JSON to flat view.
```

Для перевірки на автозбереження змінимо файл, видаливши частину JSON:



При закритті програми отримуємо повідомлення:



Тепер перевіримо файл на збереження змін:

```
{
  "title": "Person",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "The person's name",
      "hidden": "True"
    },
    "hobbies": {
      "type": "array",
      "description": "The person's hobbies",
      "items": {
        "type": "string"
      }
    }
  }
},
```

Як бачимо, зміни було збережено. Підтримка збереження всіх відкритих вкладок при закритті також присутня.

Висновок: на даній лабораторній роботі я ознайомила з наступними видами взаємодії додатків: client-server, peer-to-peer, service-oriented architecture, реалізувала частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувала одну з архітектур у власному проекті.