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 №3

Технологія розробки програмного забезпечення

«ДІАГРАМА РОЗГОРТАННЯ. ДІАГРАМА КОМПОНЕНТІВ. ДІАГРАМА ВЗАЄМОДІЙ ТА ПОСЛІДОВНОСТЕЙ.»

Варіант 28

Виконала:
студентка групи ІА–13
Хижняк Валерія Валеріївна

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Діаграма розгортання. Діаграма компонентів. Діаграма взаємодій та послідовностей.

Мета: навчитися створювати діаграму розгортання, діаграму компонентів та діаграму взаємодій та послідовностей.

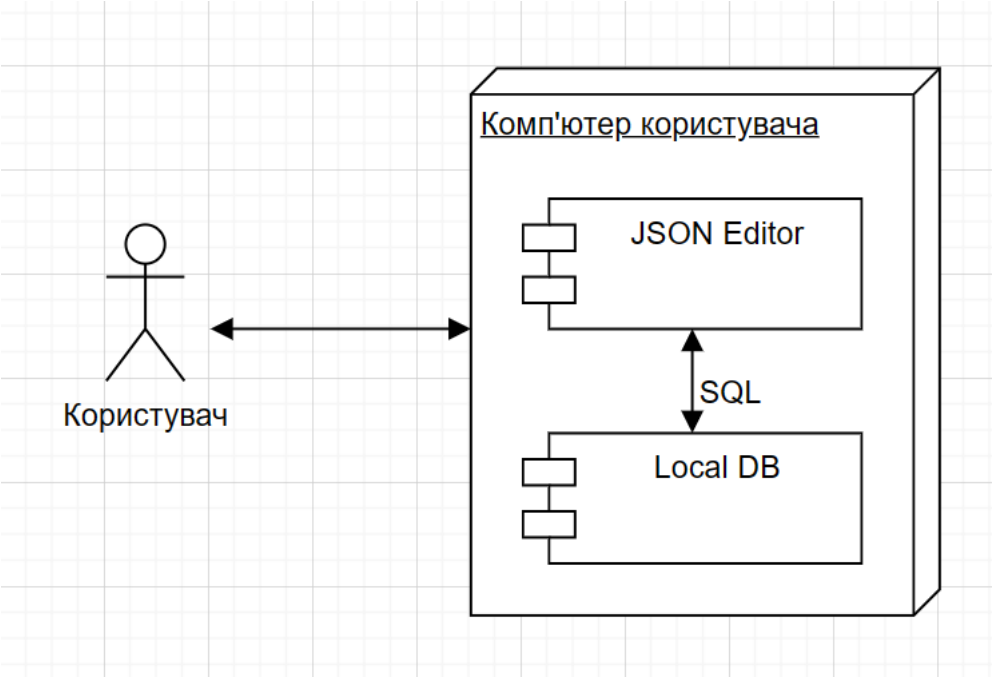
Хід роботи:

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.

1. **Діаграма розгортання:**



Діаграма розгортання представляє фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова

програмного забезпечення. Оскільки за даною предметною областю(JSON Tools) необхідно розробити додаток, який працюватиме на девайсі користувача і не буде задіювати ніякі інші сервери то в діаграмі розгортання з фізичного обладнання є лише комп’ютер користувача.

На даній діаграмі розгортання JSON Editor це наш додаток для роботи з JSON з яким взаємодіє користувач. В свою чергу JSON Editor повинен взаємодіяти з локальною БД, яка буде зберігати історію змін файлів.

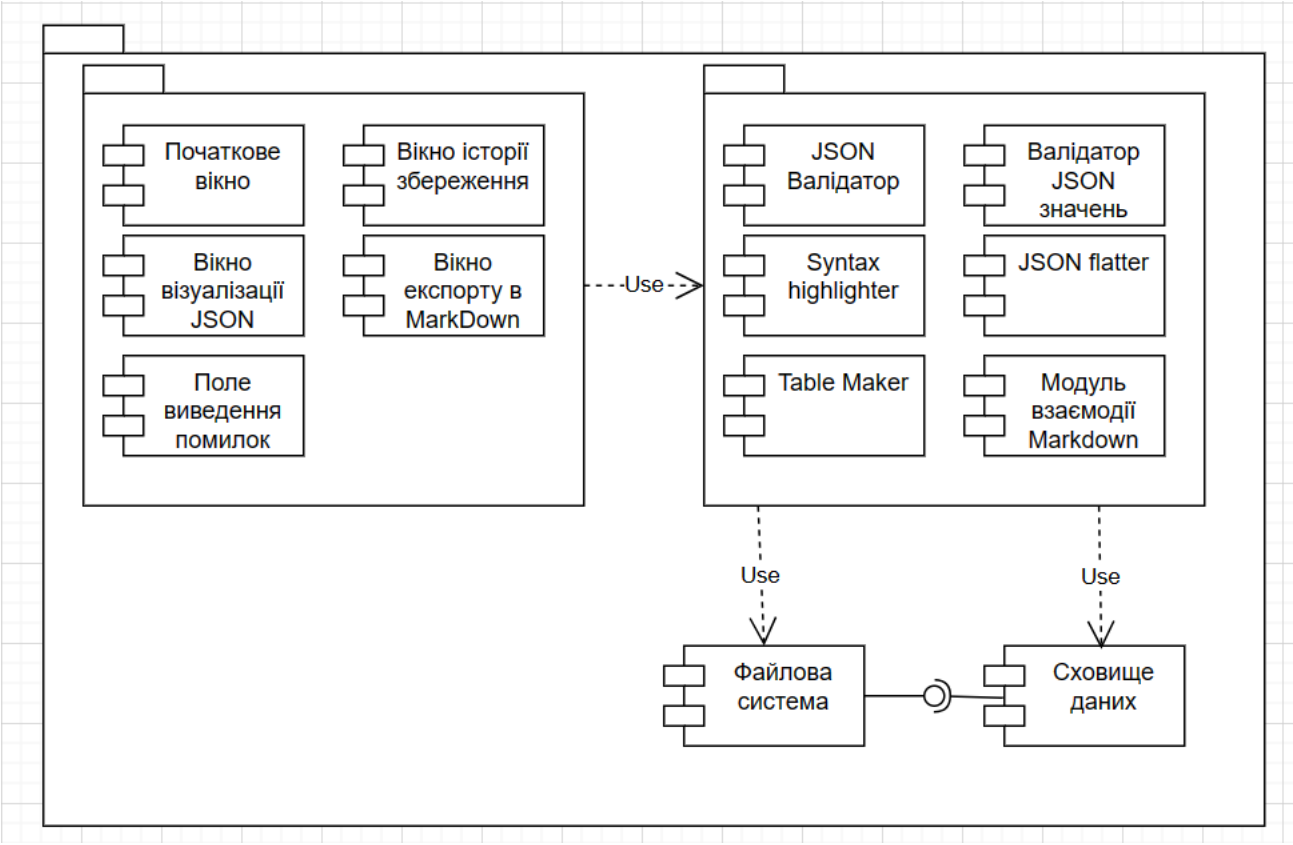
В якості БД можна обрати локальну БД, оскільки це мініатюрні бази даних, які зазвичай використовуються для локального збереження даних на пристроях користувачів або в програмах, що працюють на одному комп'ютері і це буде найзручнішим способом для збереження історії(на мою думку).

Серед таких БД можна виділити наступні:

- **CouchDB**: це документо-орієнтована база даних, яка забезпечує можливість зберігати дані у форматі JSON
- **MongoDB:** це документо-орієнтована база даних, яка спеціалізується на роботі з JSON-подібними документами. Вона дає змогу зберігати кожну зміну файлу у вигляді документа та легко взаємодіяти з ними, використовуючи MongoDB.
- **SQLite**: це легка, безсерверна, автономна та відкрита реляційна система управління базами даних (СУБД). Вона широко використовується в різних додатках, включаючи мобільні додатки, настільні програми та вбудовані системи, де потрібна невелика, ефективна та вбудована база даних. Двигун бази даних вбудований безпосередньо в додаток, а база даних зберігається в одному файлі на локальній файловій системі.

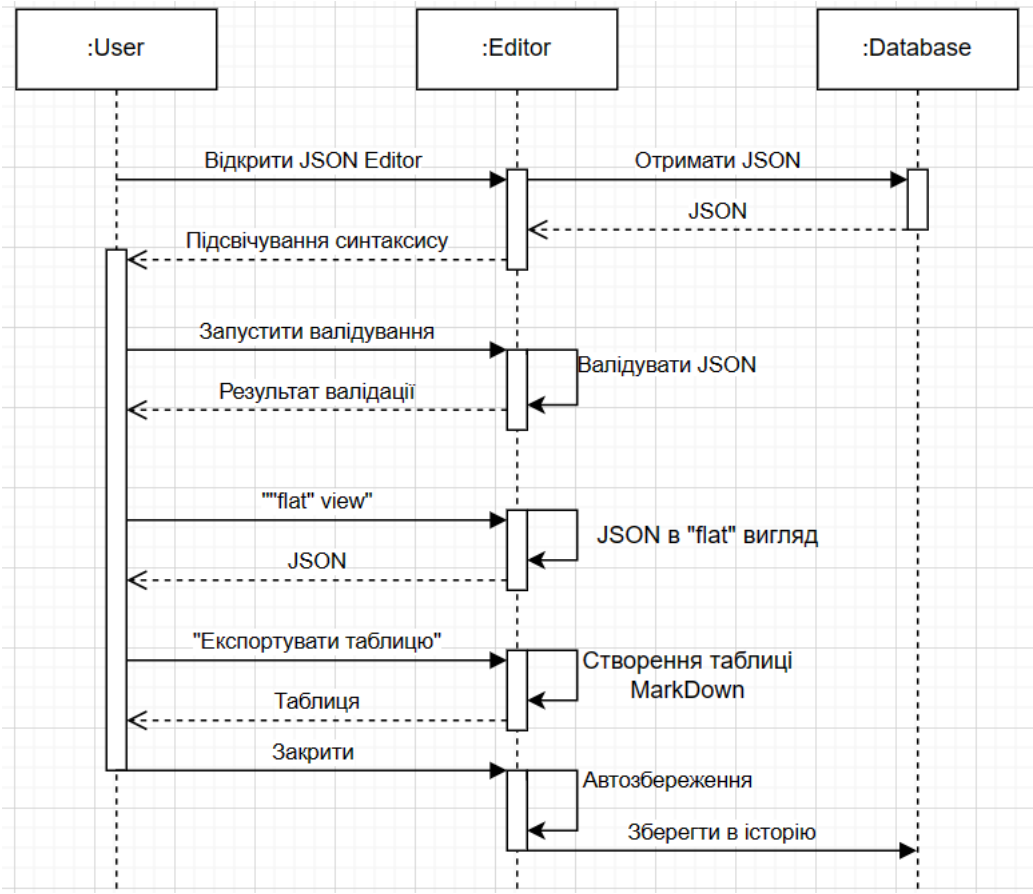
В ході розробки додатку буде обрана БД, яка найкраще підійде для нашої предметної області та зробить роботу додатку найбільш ефективною.

2. Діаграма компонентів



На даній діаграмі компонентів ми розділили компоненти на пакети. У першому пакеті знаходяться компоненти графічного відображення. У другому пакеті знаходиться бізнес логіка, яка виконує функціональність додатку. Перший пакет в свою чергу використовує цей пакет для візуалізації роботи додатку. Другий пакет використовує сховище даних для збереження історії та файлову систему для відкриття та збереження файлів .json.

3. Діаграма послідовності



Коли користувач відкриває JSON Editor програма з бази даних отримує останній json з який працював користувач та повертає його користувачу для роботи, при цьому відбувається підсвітка синтаксису. Коли користувач натискає кнопку «Валідувати» програма проводить валідацію та повертає результат валідування. Коли користувач натискає кнопку «flat view» програма перетворює json в flat вигляд та повертає користувачу. При натисканні на кнопку «Експортувати таблицю» програма створює таблицю MarkDown та повертає її користувачу. При закритті програми користувачем відбувається автозбереження змін у файлі та збереження цих змін до історії.

4. Код програми:

В ході лабораторної роботи було розроблено наступний код:

```python
2 usages  Valeriia Khyzhniak *
class JSONSchemaService(IJSONSchemaService):
    Valeriia Khyzhniak
    def __init__(self, schema_repository: IJSONSchemaRepository):
        super().__init__()
        self.schema_repository = schema_repository

    1 usage  Valeriia Khyzhniak *
    def open_file(self, schema_text):
        # Відкриваємо діалогове вікно, яке дозволяє користувачеві вибрати файл
        # Записуємо шлях до файлу в змінну
        opened_file = askopenfilename()
        # Відкриваємо файл та записуємо вміст в змінну content
        f = open(opened_file, "r", encoding="utf-8")
        content = f.read()
        # Очищуємо простір вікна редактора
        schema_text.delete(1.0, END)
        # Вставляємо в редактор значення змінної content
        schema_text.insert(END, content)
        # Повертаємо шлях до файлу
        return opened_file
```

```python
    # Зберігаємо зміни у поточному відкритому файлі
    2 usages  new *
    def save_file(self, schema_text, file_name):
        # Записуємо вміст редактора в змінну
        content = schema_text.get(1.0, END)
        # Відкриваємо файл та записуємо в нього значення змінної
        f = open(file_name, "w", encoding="utf-8")
        f.write(content)
        # Закриваємо файл
        f.close()
        # Очищаємо вікно редактора
        schema_text.delete(1.0, END)
```

```python
    def save_as_file(self, schema_text):
        # Відкриваємо діалогове вікно, яке дозволяє користувачеві вибрати файл
        # Записуємо шлях до файлу в змінну
        file_to_save = asksaveasfilename()
        # Записуємо вміст редактора в змінну
        content = schema_text.get(1.0, END)
        # Відкриваємо файл та записуємо в нього значення змінної
        f = open(file_to_save, "w", encoding="utf-8")
        f.write(content)
        # Закриваємо файл
        f.close()
        # Очищаємо вікно редактора
        schema_text.delete(1.0, END)

    # Автозбереження при закритті редактора
    1 usage  new *
    def auto_save(self, schema_text, opened_file):
        print("autosave")
        # Викликаємо метод save_file
        self.save_file(schema_text, opened_file)
```

```python
class JSONSchemaView(Tk, IJSONSchemaView):
    # Створюємо об'єкт класу JSONSchemaService()
    json_schema_service = JSONSchemaService()
    # Створюємо змінну для збереження шляху відкритого файлу
    opened_file = ''

    # Ініціалізуємо атрибути класу
    Valeriia Khyzhniak *
    def __init__(self):
        super().__init__()
        # Встановлюємо назву вікна
        self.title("JSON Schema Editor")
        # Встановлюємо розмір вікна
        self.geometry("1200x600")
        # Встановлюємо DPI-свідомість процесу в Windows, для того щоб програма могла змінювати
        # масштаб елементів інтерфейсу користувача, щоб вони виглядали на екрані більш чітко
        ctypes.windll.shcore.SetProcessDpiAwareness(True)
        # Створюємо об'єкт рамки
        self.schema_frame = LabelFrame(self, text='JSONSchema', padx=25, pady=20, font='Consolas 20')
        # Створюємо об'єкт текстового поля для редагування JSON-схеми
        self.schema_text = Text(self.schema_frame, font='Consolas 30', width=40, height=20, wrap=tk.NONE)
```

```
        # Створюємо меню для вікна
        self.create_menu()
        # Створюємо панель інструментів для редагування JSON - схеми
        self.create_toolbar()
        # Створюємо рамку для відображення JSON-схеми
        self.create_schema_frame()

    # Створюємо меню для вікна
    1 usage  Valeriia Khyzhniak *
    def create_menu(self):
        # Створюємо об'єкт меню
        self.menu = tk.Menu(self)
        # Додаємо пункт меню "Файл"
        self.file_menu = tk.Menu(self.menu, tearoff=0)
        self.file_menu.add_command(label="Відкрити", font='10', command=self.open_file)
        self.file_menu.add_command(label="Зберегти", font='10', command=self.save_file)
        self.file_menu.add_command(label="Зберегти як...", font='10', command=self.save_as_file)
        self.menu.add_cascade(label="Файл", menu=self.file_menu)
        self.protocol( name: "WM_DELETE_WINDOW", self.on_closing)
        # Встановлюємо меню для вікна
        self.config(menu=self.menu)
```

```
# Створюємо панель інструментів для редагування JSON-схеми
1 usage  Valeriia Khyzhniak *
def create_schema_frame(self):
    # Розташовуємо текстове поле у рамці
    self.schema_text.pack(fill=BOTH, expand=True)
    # Розташовуємо рамку у лівій частині вікна
    self.schema_frame.pack(side=LEFT, fill=BOTH, expand=True)

1 usage  Valeriia Khyzhniak *
def create_toolbar(self):
    # Створюємо об'єкт панелі інструментів
    self.toolbar = tk.Frame(self, bd=1, relief=tk.RAISED)
    self.toolbar.pack(side=TOP, fill=tk.X)

1 usage  new *
def on_closing(self):
    # Виконайте необхідну дію тут
    self.json_schema_service.auto_save(self.schema_text, self.opened_file)
    self.destroy()
    print("close")
```

```
1 usage  new *
def open_file(self):
    self.opened_file = self.json_schema_service.open_file(self.schema_text)
    print(self.opened_file)

1 usage  new *
def save_file(self):
    print("save file" + self.opened_file)
    self.json_schema_service.save_file(self.schema_text, self.opened_file)
    self.opened_file = ''

1 usage  new *
def save_as_file(self):
    self.json_schema_service.save_as_file(self.schema_text)
    self.opened_file = ''
```

Висновок: на даній лабораторній роботі я побудувала діаграму розгортання, діаграму компонентів та діаграму взаємодій та послідовностей, продовжила розробку коду програми.