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 №2

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ. СЦЕНАРІЇ ВАРІАНТІВ ВИКОРИСТАННЯ. ДІАГРАМИ UML. ДІАГРАМИ КЛАСІВ. КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ JSON Tool

Виконав
студент групи ІА–22:
Щаблевський Д. Е.
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи

Мета: Проаналізувати тему. Розробити діаграму розгортання, діаграму компонентів та діаграму послідовностей для проектованої системи.

Хід роботи

..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.

Теоретичні відомості

У 2 лабораторній роботі розглядається використання діаграм UML для моделювання різних аспектів системи. Ключовими елементами є діаграми варіантів використання, які допомагають визначити основні сценарії взаємодії користувачів із системою.

1. **UML (Unified Modeling Language)** — це універсальна мова для візуального моделювання, яка використовується для опису структури та поведінки програмних систем. Вона надає можливість створювати діаграми для різних аспектів системи, забезпечуючи чітке та зрозуміле уявлення про її роботу. UML застосовується на всіх етапах розробки, від аналізу вимог до етапу тестування.
2. **Діаграми варіантів використання (Use Case Diagrams)** — це засіб для моделювання функціональності системи з точки зору її користувачів. Вони

відображають, як зовнішні актори (користувачі або інші системи) взаємодіють із системою для досягнення певних цілей. Основними елементами таких діаграм є актори, варіанти використання (use cases) та зв’язки між ними.

3. **Діаграма класів (Class Diagrams)** — це важливий інструмент UML, що відображає структуру системи на рівні класів та їх взаємодії. Вона показує класи, їх атрибути, методи, а також різні типи зв’язків між ними, такі як асоціації, агрегації та композиції. Ця діаграма є базою для реалізації об’єктно-орієнтованого програмування, оскільки дозволяє чітко визначити структуру об’єктів у системі.
4. **Концептуальна модель системи** — це початковий етап моделювання, що дозволяє визначити ключові компоненти системи, їх ролі та взаємодії. У рамках цієї моделі можуть бути створені діаграми варіантів використання та діаграми класів, які стають основою для подальшого аналізу та реалізації.
5. **Репозиторій** — це шаблон проєктування, який створює абстракцію над операціями з базою даних. Завдяки цьому шаблону взаємодія з базою даних здійснюється через інтерфейси, що спрощує підтримку та тестування коду. Репозиторій приховує технічні деталі доступу до даних, що робить код більш структурованим і легшим для підтримки.

Схема прецендентів

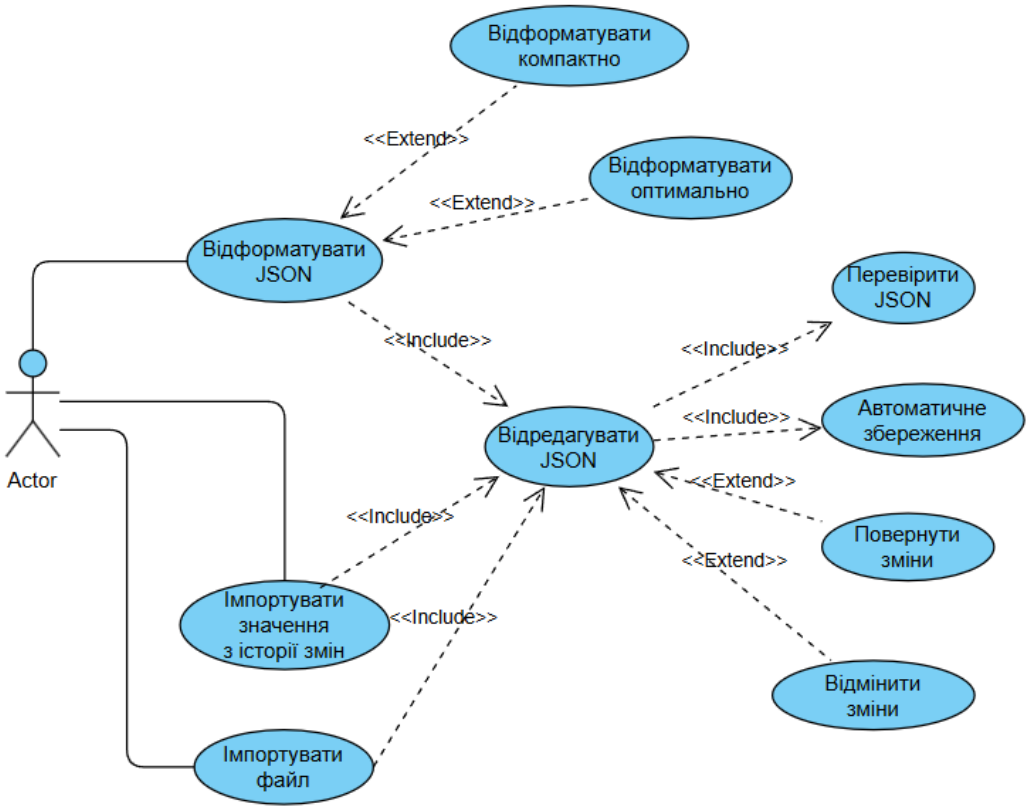

Рисунок 1. – діаграма прецендентів

Сценарії використання для 3 прецендентів

Таблиця 1.1 – Сценарій використання «**Перевірка JSON відповідно до схеми»**

Назва	Перевірка JSON відповідно до схеми
Передумови	Користувач має доступ до системи.
Післяумови	• Користувач отримує підтвердження, чи відповідає JSON заданій схемі. • У разі виявлення помилок — користувач отримує їхній список.

Сторони, що взаємодіють	Користувач, система
Опис	Користувач перевіряє, чи відповідає JSON-об'єкт заданій JSON-схемі, завантажуючи їх у систему для валідації.
Основний потік подій	1. Користувач вводить JSON-об'єкт і JSON-схему в систему. 2. Система виконує перевірку JSON відповідно до схеми. 3. Система повідомляє користувача про результат: • Якщо JSON відповідає схемі — повідомлення про валідність. • Якщо JSON не відповідає схемі — список помилок.
Виняткові ситуації	Користувач ввів некоректний JSON-об'єкт або JSON-схему, тоді система відображає повідомлення про помилку: "Файл JSON некоректний або схема має помилки".
Примітки	У разі перевірки великих JSON-об'єктів може збільшитися час обробки.

Таблиця 1.2 – Сценарій використання «**Редагування метаданих властивостей JSON-схеми**»

Назва	Редагування метаданих властивостей JSON-схеми

Передумови	1. Користувач має доступ до інтерфейсу редагування JSON-схеми. 2. У системі завантажена JSON-схема для редагування.
Післяумови	Зміни до метаданих властивостей JSON-схеми збережено.
Сторони, що взаємодіють	Користувач, система
Опис	Користувач редагує метадані властивостей JSON-схеми, зокрема опис, приклади, типи даних та інші характеристики, через інтерфейс системи.
Основний потік подій	1. Користувач відкриває інтерфейс редагування властивостей JSON-схеми. 2. У інтерфейсі вибирає потрібну властивість для редагування. 3. Вносить зміни до таких метаданих, як: • Опис властивості; • Приклади значень; • Типи даних або інші характеристики. 4. Система автоматично зберігає внесені зміни в JSON-схемі. 5. Користувач за потреби обирає функцію відновлення попередньої версії метаданих.

Виняткові ситуації	Користувач залишає обов'язкове поле пустим, тоді система виводить повідомлення.
Примітки	Інтерфейс підтримує валідацію введених даних у реальному часі.

Таблиця 1.3 – Сценарій використання «**Завантаження JSON-файлу**»

Назва	Завантаження JSON-файлу
Передумови	Користувач має доступ до редактора з функцією імпорту файлів.
Післяумови	1. Дані з завантаженого файлу відображаються у редакторі. 2. Якщо файл не відповідає вимогам, користувач отримує повідомлення про помилку.
Сторони, що взаємодіють	Користувач, система
Опис	Користувач завантажує JSON або текстовий файл до редактора, щоб працювати з його вмістом.

Основний потік подій	1. Користувач відкриває інтерфейс для імпорту файлів у редактор. 2. У вікні вибору файлів користувач обирає файл у форматі .json або .txt. 3. Користувач підтверджує свій вибір. 4. Система перевіряє формат файлу: Якщо формат коректний, система обробляє файл і завантажує дані у редактор. 5. Редактор відображає вміст файлу для подальшого редагування.
Виняткові ситуації	Користувач обирає файл у невідповідному форматі, система блокує підтвердження вибору і виводить повідомлення.
Примітки	Перед завантаженням система виконує валідацію файлу, щоб уникнути помилок у редакторі.

Діаграма класів

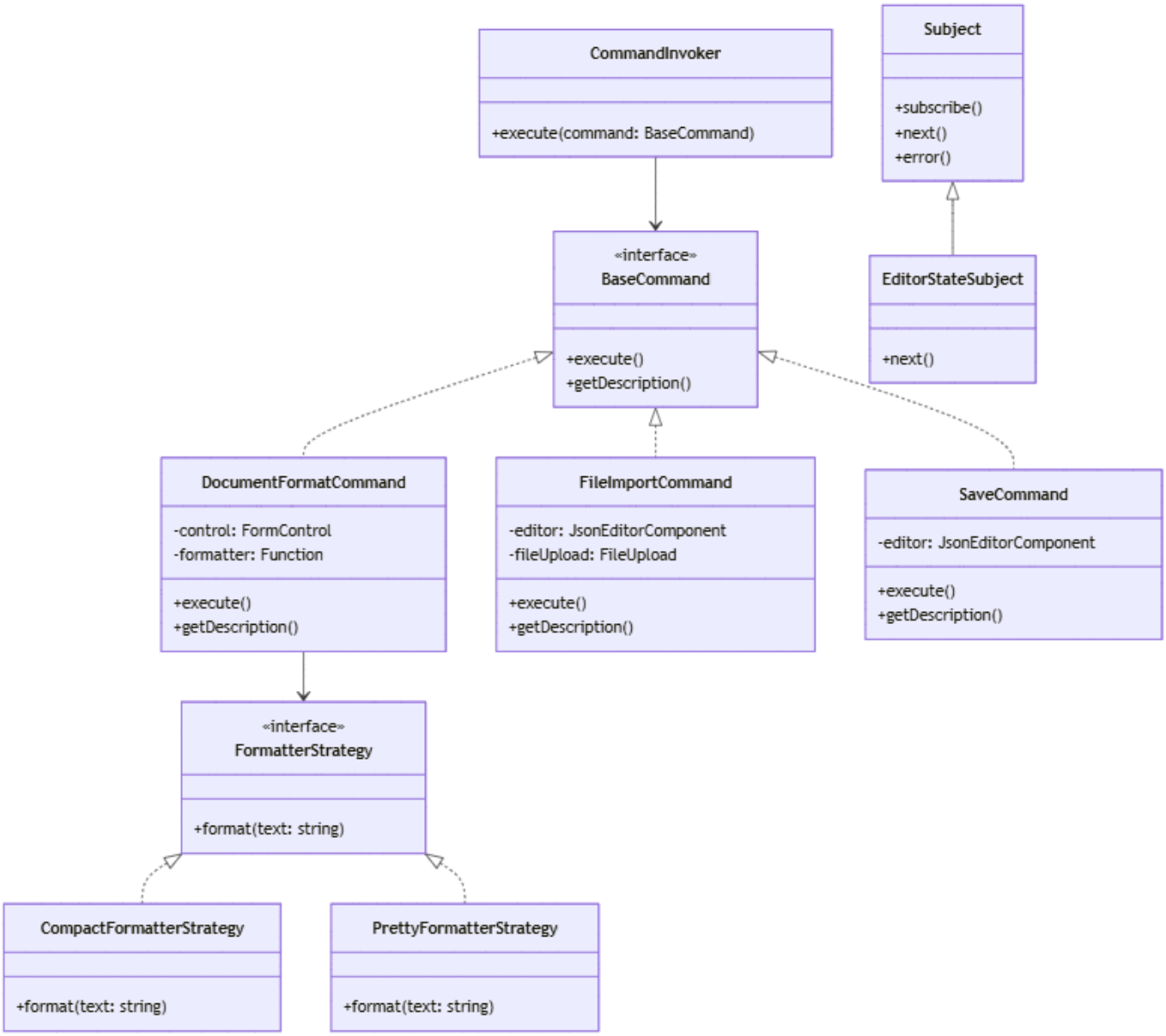

Рисунок 2.1 - Діаграма класів

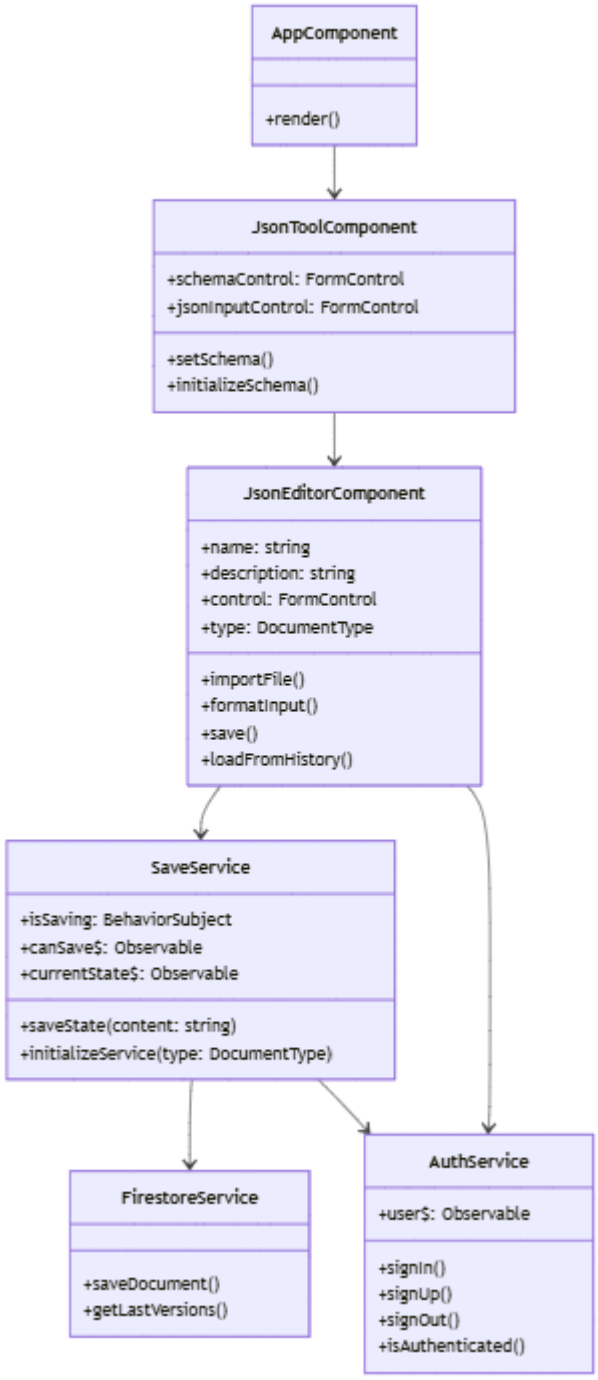

Рисунок 2.2 – діаграма класів

Опис основних класів JSON редактора:

1. Компоненти застосунку:

- **AppComponent**: Кореневий компонент, що ініціалізує основні модулі та сервіси, налаштовує маршрутизацію та глобальні стилі
- **JsonToolComponent**: Контейнер, що координує роботу редактора та панелі інструментів, керує станом JSON схеми та валідацією введених даних
- **JsonEditorComponent**: Основний редактор з Monaco Editor, забезпечує форматування, підсвічування синтаксису та валідацію JSON в реальному часі
- **JsonToolbarComponent**: Інтерфейс для файлових операцій, містить кнопки форматування, завантаження/збереження файлів та налаштування редактора

2. Сервіси:

- **SaveService**: Реалізує логіку автозбереження з таймером відліку, зберігає історію змін та забезпечує відновлення даних
- **UndoRedoService**: Підтримує стек змін для можливості скасування/повторення операцій, синхронізується з локальним сховищем
- **DBService**: Абстракція над IndexedDB, забезпечує типізоване збереження та отримання даних
- **LoggerService**: Системний журнал подій, допомагає відстежувати роботу патернів та налагоджувати застосунок

Структура бази даних (Firebase Datastore)

Бази даних дає можливість зберігати дані для роботи з користувачами та їхніми документами, розділеними на формати JSON і Schema. База даних складається з трьох таблиць:

1. **Users** – зберігає інформацію про користувачів:
    - id (string): Унікальний ідентифікатор користувача.
    - email (string): Електронна адреса користувача.
    - createdAt (datetime): Дата створення профілю.
    - signedIn (datetime): Дата останнього входу в систему.
2. **JSON** – зберігає документи у форматі JSON:
    - id (string): Унікальний ідентифікатор документа.

- content (object): Вміст документа у форматі JSON.
- timestamp (datetime): Час останнього оновлення документа.
- userId (string): Ідентифікатор користувача, якому належить документ.

3. **Schema** – зберігає документи у форматі Schema:
   - id (string): Унікальний ідентифікатор документа.
   - content (object): Вміст документа у форматі Schema.
   - timestamp (datetime): Час останнього оновлення документа.
   - userId (string): Ідентифікатор користувача, якому належить документ.

Відношення між таблицею **Users** і таблицями **JSON** та **Schema** є "один до багатьох", що означає, що кожен користувач може мати кілька документів у форматах JSON або Schema.

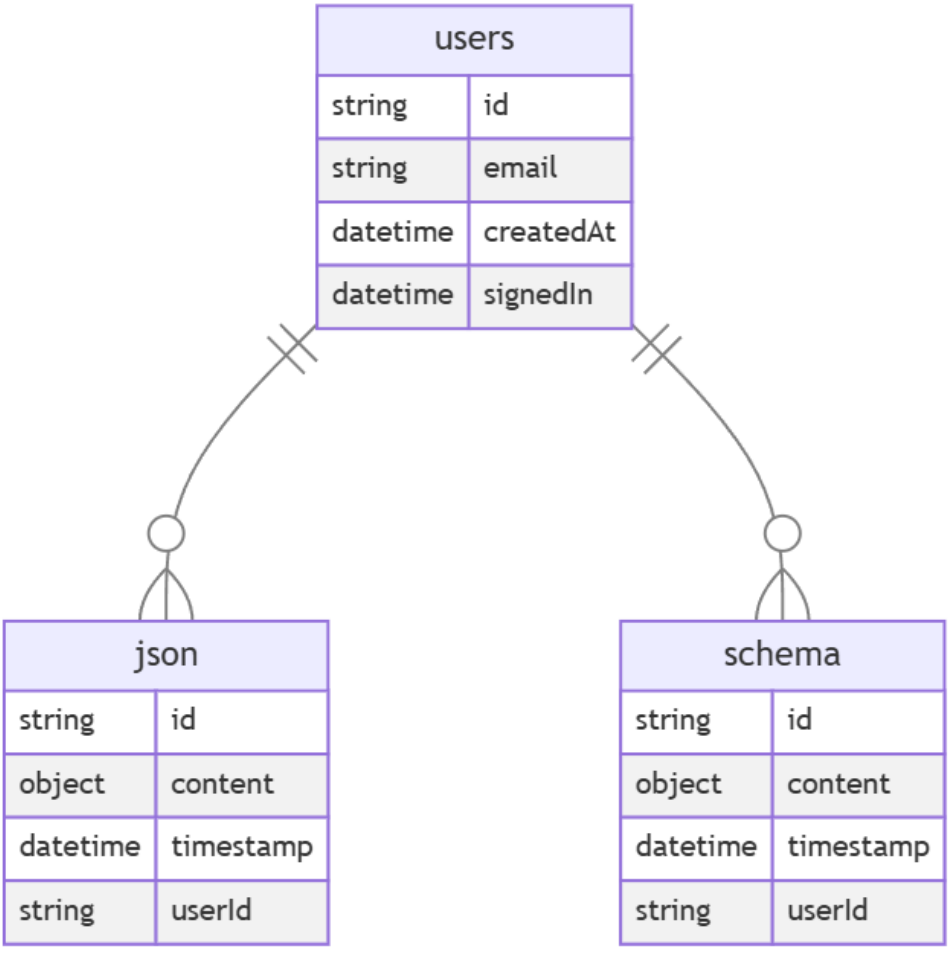

Рисунок 3 - Структура бази даних

Посилання на проект: https://github.com/neodavis/json-tool

Висновок: В даній лабораторній роботі я проаналізував тему, намалював схему прецедентів, діаграму класів, розробив основні класи і структуру бази даних.