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

Технології розроблення програмного забезпечення

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

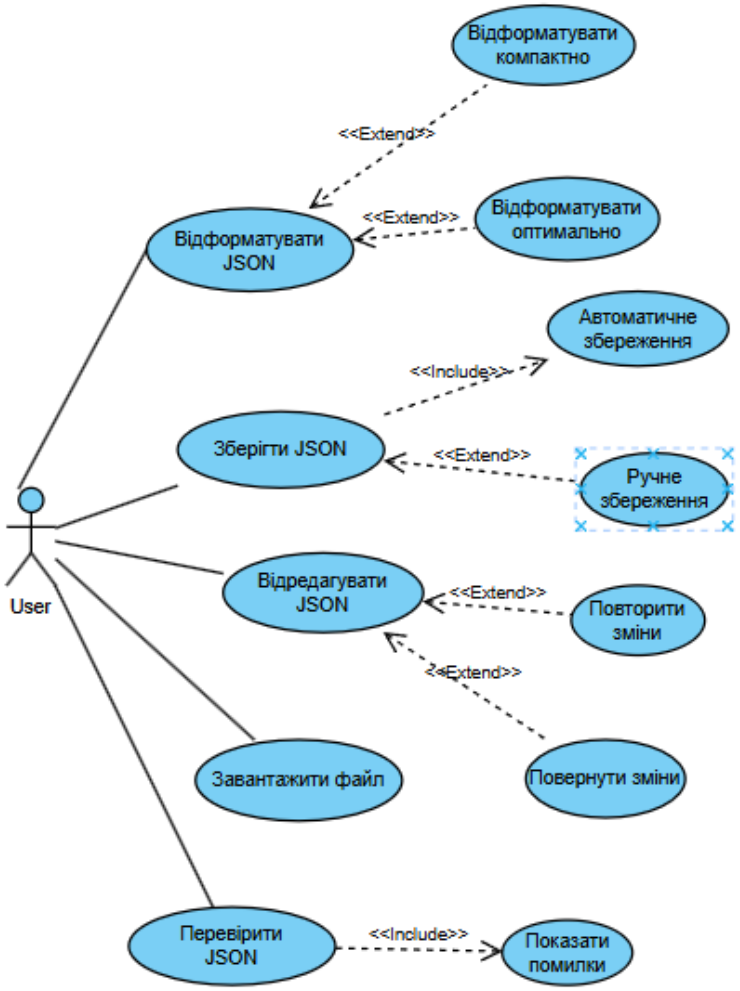

Рисунок 1. – Схема прецденту

Сценарії використання для 3 прецендентів

Сценарій використання 1: Перевірка JSON відповідно до схеми

Актори: Користувач.

Опис: Користувач перевіряє, чи відповідає JSON-об'єкт заданій JSON-схемі.

Основний хід подій:

1. Користувач завантажує JSON-об'єкт і відповідну JSON-схему.
2. Система порівнює JSON із заданою схемою.
3. У разі успішної перевірки система повідомляє, що JSON валідний.
4. Якщо в JSON є помилки, система відображає список помилок.

   Виключення: Якщо користувач завантажив неповний або некоректний JSON-об'єкт чи схему, система виведе повідомлення про помилку.

Сценарій використання 2: Редагування метаданих властивостей схеми

Актори: Користувач.

Опис: Користувач редагує метадані властивостей JSON-схеми, такі як опис, приклади, типи даних тощо.

Основний хід подій:

1. Користувач відкриває інтерфейс редагування властивостей JSON-схеми.
2. Вибирає властивість для редагування.
3. Вносить зміни до опису, прикладів або інших метаданих.
4. Система автоматично зберігає зміни в JSON-схемі.
5. За необхідності користувач може відновити попередню версію метаданих.

   Виключення: Якщо користувач залишає поле пустим або вводить некоректні дані, система попереджає про помилку.

Сценарій використання 3: Завантажити JSON файл

Актори: Користувач.

Опис: Користувач завантажує JSON файл у редактор

Основний хід подій:

1. Користувач відкриває функцію імпорту файлу.
2. Користувач обирає файл відповідного формату (.json, або .txt)
3. Користувач підтверджує вибір файлу
4. Система обробляє обраний файл та вводить дані у відповідний редактор.

Виключення: Якщо обрано файл невірного формату, то система не дозволить підтвердити вибір.

Діаграма класів

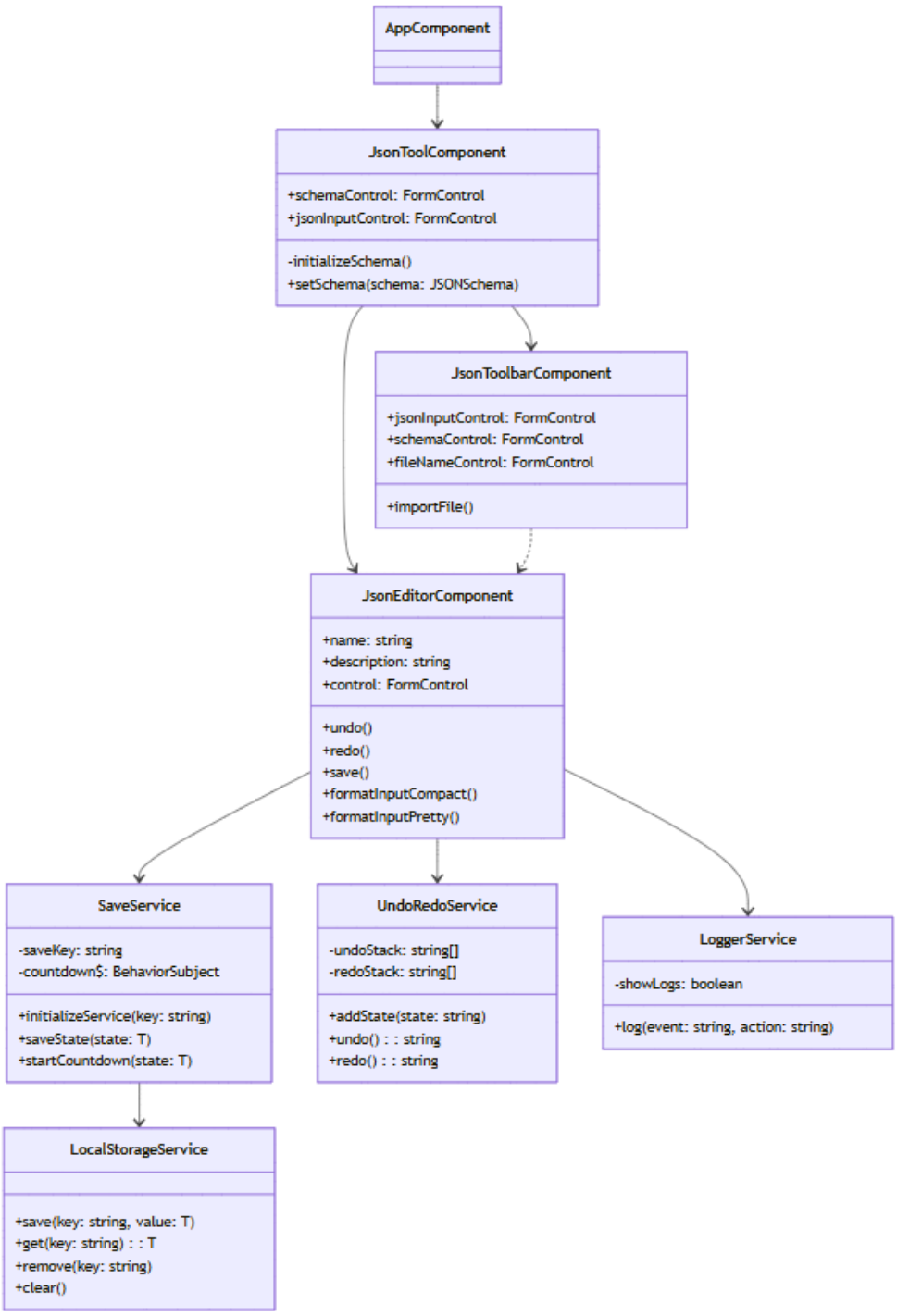

Рисунок 2. - Діаграма класів

Опис основних компонентів 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
- **LocalStorageService**: Абстракція над браузерним localStorage, забезпечує типізоване збереження та отримання даних
- **LoggerService**: Системний журнал подій, допомагає відстежувати роботу патернів та налагоджувати застосунок

3. Основний функціонал:

- **Редагування JSON**: Повнофункціональний редактор з підсвічуванням синтаксису
- **Валідація схеми**: Перевірка відповідності JSON документа заданій схемі
- **Файлові операції**: Завантаження/збереження JSON файлів з підтримкою drag&drop
- **Автозбереження**: Автоматичне збереження змін з налаштовуваним таймером
- **Історія змін**: Повна історія змін з можливістю відміни/повторення

- **Форматування**: Гнучкі налаштування форматування JSON структури

Структура бази даних (LocalStorage)

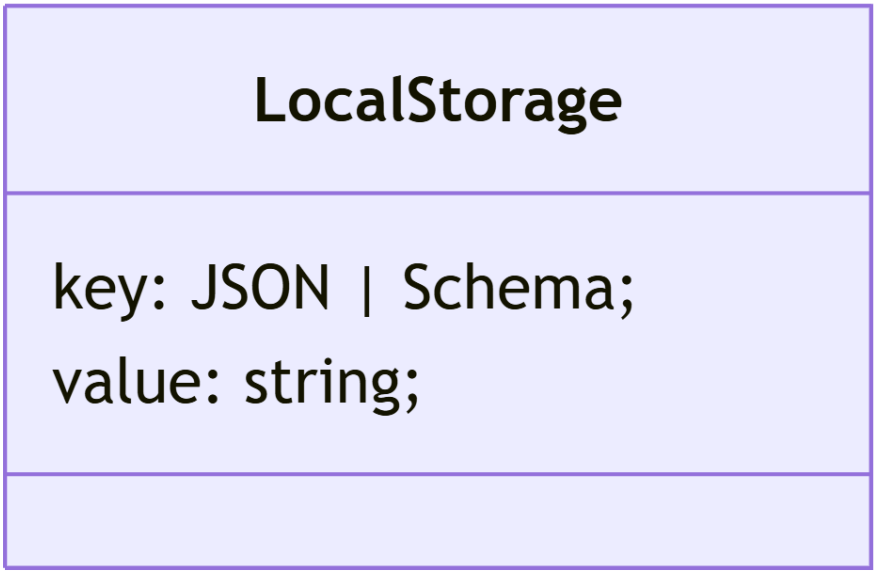

Рисунок 3. - Структура бази даних

Локальне сховище відповідає за збереження і завантаження стану редакторів. Записи містять ключ, який може бути або JSON, або Schema (у залежності від редактора, за стан якого відповідає запис), та значення типу String, яке містить текст, який введено у редактор.

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.