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

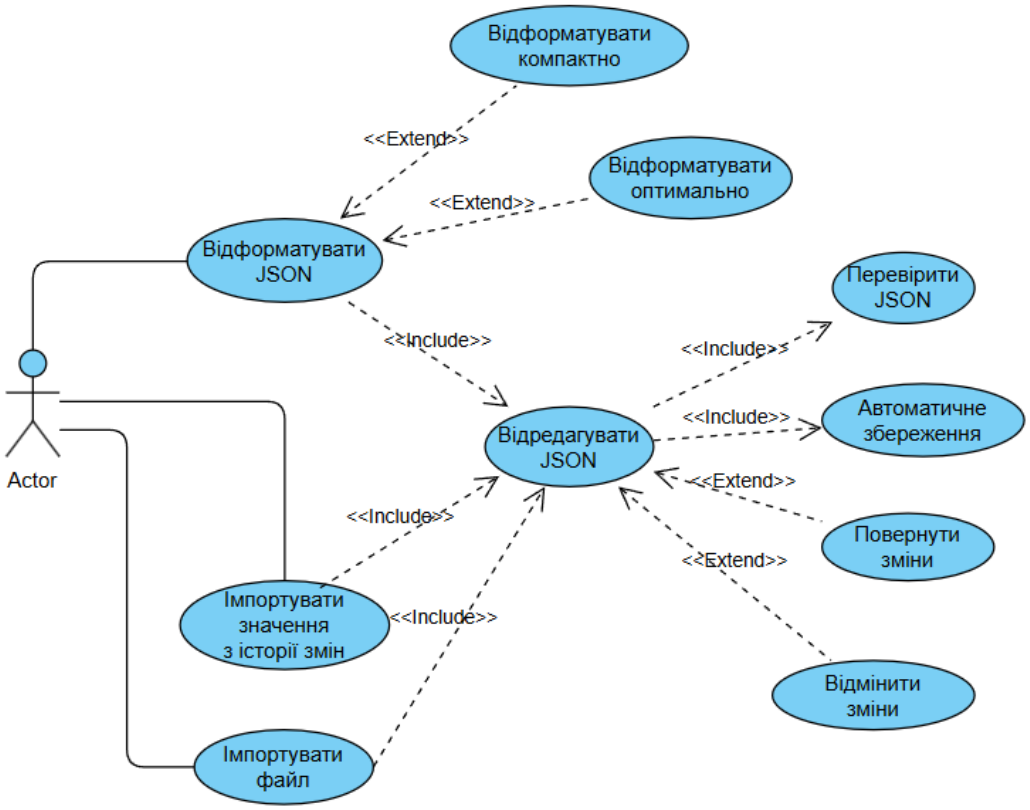

Рисунок 1. – діаграма прецендентів

Сценарії використання для 3 прецендентів

Таблиця 1.1 – Сценарій використання «**Перевірити JSON**»

Назва	Перевірити JSON
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, вводячи 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.
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 1.3 – Сценарій використання «**Імпорт значення з історії змін**»

Назва	Імпорт значення з історії змін
Передумови	Користувач має доступ до системи, та має історію змін

Післяумови	Попередня версія JSON імпортована в редактор.
Сторони, що взаємодіють	Користувач, система
Опис	Користувач обирає збережену версію JSON зі списку історії.
Основний потік подій	1. Користувач відкриває історію змін. 2. Вибирає потрібну версію. 3. Система імпортує обране значення у редактор.
Виняткові ситуації	Якщо користувач, попередньо, не увійшов в акаунт, то система не дасть змогу переглянути історію змін
Примітки	Перед імпортом система виконує валідацію файлу, щоб уникнути помилок у редакторі.

Таблиця 1.3 – Сценарій використання «**Імпорт файлу**»

Назва	Імпорт файлу
Передумови	Користувач має доступ до системи.

Післяумови	1. Дані з імпортованого файлу відображаються у редакторі. 2. Якщо файл не відповідає вимогам, користувач отримує повідомлення про помилку.
Сторони, що взаємодіють	Користувач, система
Опис	Користувач імпортує 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 та імпортує 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 імпортом система виконує валідацію файлу, щоб уникнути помилок у редакторі.

Діаграма класів

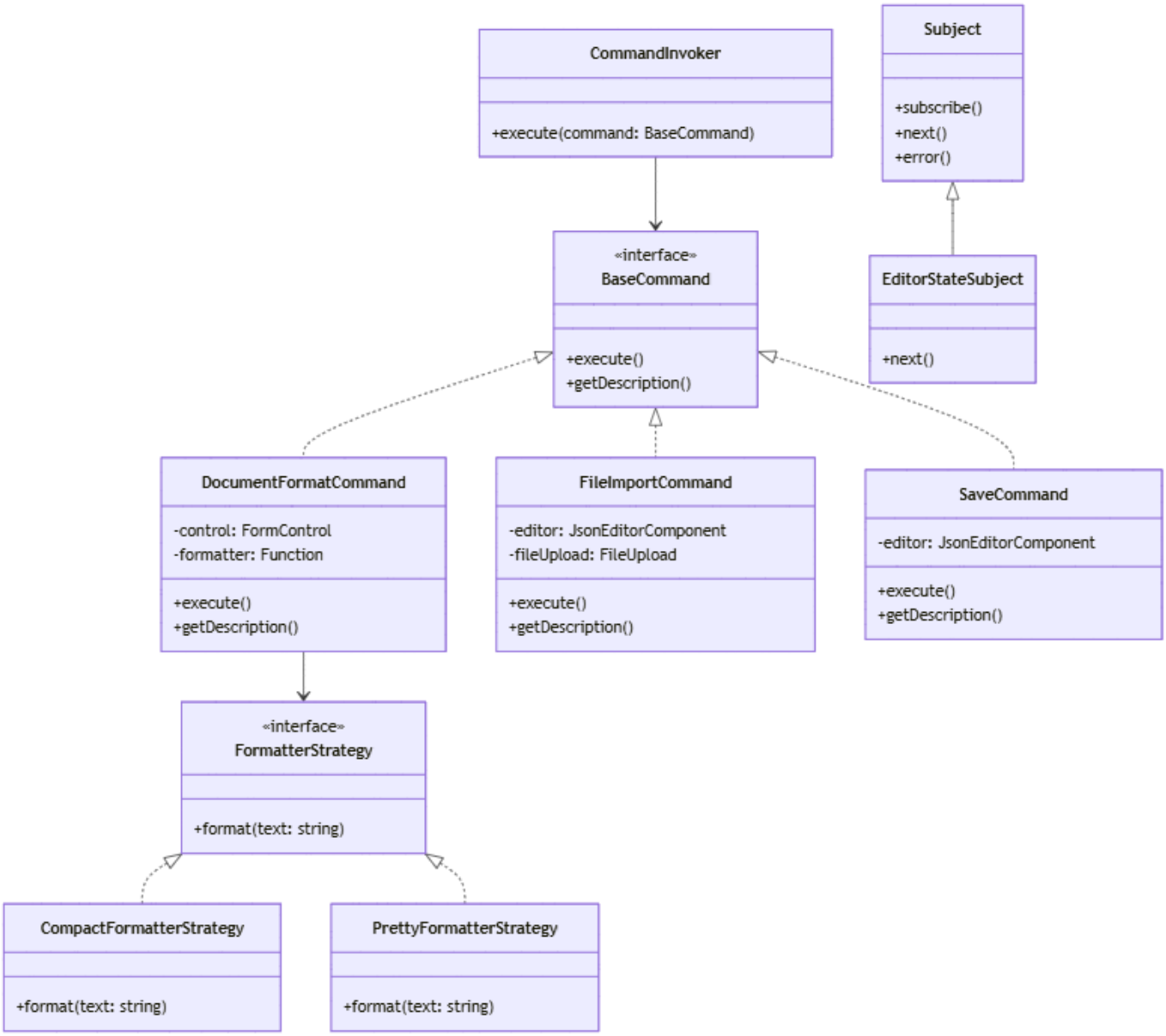

Рисунок 2.1 - Діаграма класів

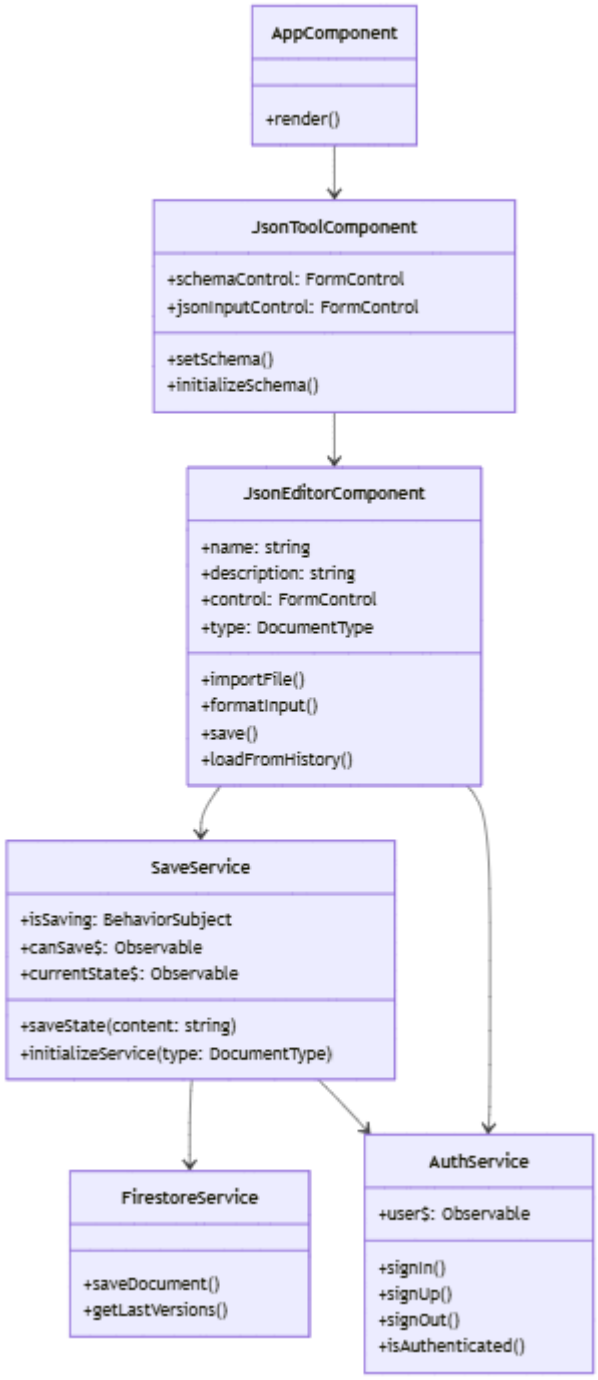

Рисунок 2.2 – діаграма класів

Опис основних класів JSON редактора:

Класи компонентів:

- AppComponent: Відповідає за рендеринг основного компонента застосунку.

- JsonToolComponent: Керує введенням JSON-схеми та даних, ініціалізує схему, перевіряє її валідність.
- JsonEditorComponent: Основний редактор JSON із функціями імпорту, форматування, збереження та завантаження історії.

Класи сервісів:

- SaveService: Реалізує збереження стану редактора, відповідає за логіку автозбереження.
- FirestoreService: Сервіс для роботи з Firebase, зберігає та отримує документи.
- AuthService: Відповідає за автентифікацію користувачів, забезпечує функції входу, реєстрації та перевірки автентичності.

Класи команд:

- CommandInvoker: Виконує команди, використовуючи об'єкти BaseCommand.
- DocumentFormatCommand: Команда для форматування JSON-документів.
- FileImportCommand: Команда для імпорту файлів у редактор.
- SaveCommand: Команда для збереження змін у редакторі.

Реалізації стратегій форматування:

- CompactFormatterStrategy: Стратегія форматування JSON у компактному вигляді.
- PrettyFormatterStrategy: Стратегія форматування JSON у читабельному вигляді з відступами.

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. **Document** – зберігає введені значення з інтерфейсу користувача:
   - 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.

Відношення між таблицею **Users** і **Document** є "один до багатьох", що означає, що кожен користувач може мати кілька збережених документів.

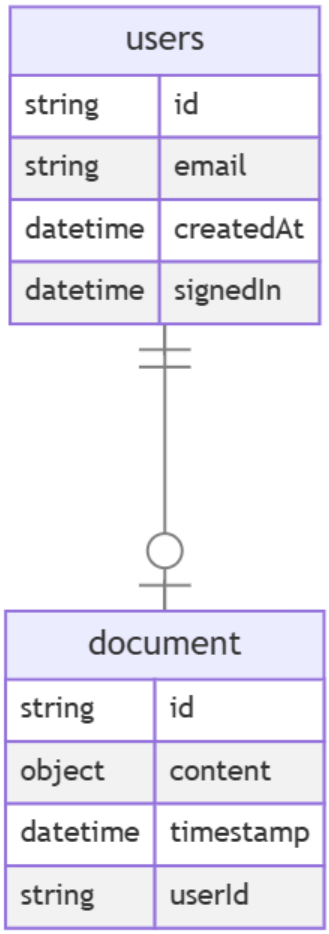

Рисунок 3 - Структура бази даних

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

Висновок:
У ході виконання лабораторної роботи я ознайомився з основними аспектами моделювання програмних систем за допомогою UML-діаграм. Зокрема, я створив діаграму варіантів використання, яка описує основні сценарії взаємодії користувачів із системою JSON Tool, а також розробив сценарії використання для ключових прецедентів.

Діаграма класів допомогла візуалізувати структуру системи, визначивши основні компоненти та взаємодії між ними. Завдяки цьому було сформовано чітке уявлення про функціональність системи, що є основою для подальшої реалізації. Крім того, я створив концептуальну модель системи, яка описує ключові компоненти та їх взаємодії, включаючи реалізацію шаблону репозиторію для роботи з базою даних.

У процесі роботи було визначено, що використання UML-діаграм дозволяє ефективно моделювати складні системи, забезпечуючи структурований підхід до розробки.

Отже, виконання цієї лабораторної роботи сприяло закріпленню знань з UML-моделювання та розробки концептуальних моделей програмних систем, що є важливим етапом у проектуванні сучасних застосунків.