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

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Текстовий редактор »

Виконав

студент групи ІА–33

Супик А.О.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Вступ

У даній лабораторній роботі розглядаються основи проєктування програмного забезпечення на прикладі розробки архітектури для багатофункціонального «Текстового редактора».

Метою роботи є опанування ключових інструментів та методологій проєктування, зокрема використання мови UML для візуалізації архітектури, розробки сценаріїв використання для визначення функціональних вимог, а також застосування патернів проєктування для побудови гнучкої та розширюваної системи.

У ході роботи було спроєктовано систему, що надає користувачеві повний набір інструментів для роботи з текстовими файлами: їх відкриття, редагування, збереження та перегляд. Архітектура підтримує розширені функції, такі як використання макросів для автоматизації дій, робота із закладками, пошук по тексту, а також інтеграція із зовнішніми сервісами для перекладу та перевірки синтаксису. Для взаємодії з базою даних реалізовано патерн Repository, що відокремлює бізнес-логіку від доступу до даних.

Результатом роботи є детальний проєкт архітектури текстового редактора, готовий для подальшої реалізації.

Теоретичні відомості

У даній лабораторній роботі для проєктування системи «Текстовий редактор» були застосовані фундаментальні концепції об'єктно-орієнтованого аналізу та проєктування (ООАП). Це дозволило створити надійну, гнучку та підтримувану систему. Ключовими інструментами, які було використано, є уніфікована мова моделювання (UML) та принципи проєктування баз даних.

UML — це загальновизнаний стандарт для візуального моделювання програмних систем. За допомогою UML ми можемо специфікувати, візуалізувати та документувати архітектуру програмного забезпечення на різних рівнях абстракції:

Концептуальний рівень: На цьому рівні система розглядається з точки зору користувача та бізнес-логіки. Для цього була розроблена діаграма варіантів використання (Use Case Diagram). Вона визначає основні функціональні вимоги системи та її взаємодію з акторами: Користувач (User) та SOA (Service-Oriented Architecture) як представник зовнішніх сервісів.

Логічний рівень: Цей рівень описує статичну структуру та динамічну поведінку системи. Для опису статичної структури була створена діаграма класів (Class Diagram). Вона деталізує класи (User, Document, TextEditor, Macro, Bookmark) та їх взаємозв'язки, що є основою для подальшої реалізації. Діаграма також включає сутності-репозиторії (UserRepository, DocumentRepository), які відповідають за взаємодію з базою даних.

Фізичний рівень: На цьому рівні описується фізичне розгортання компонентів системи. У нашому випадку це включає логічну структуру бази даних, спроєктовану на основі логічної моделі. Діаграма сутностей-зв'язків показує таблиці (User, Document, Macro тощо), їх первинні (PK) та зовнішні (FK) ключі, що забезпечує цілісність даних.

Діаграма варіантів використання (Use Case Diagram)

Діаграма варіантів використання є відправною точкою в аналізі вимог. Вона фіксує функціональність системи та її межі. Основними елементами діаграми є:

Актор (Actor): Будь-яка зовнішня сутність, що взаємодіє із системою. У нашому випадку це Користувач та SOA (зовнішній сервіс).

Варіант використання (Use Case): Послідовність дій, яку система виконує для досягнення певної мети для актора. Наприклад, Відкрити файл, Редагування тексту або Інтеграція із зовнішніми сервісами.

Зв'язки (Relationships): Описують взаємодію між акторами та варіантами використання. У роботі використано <<extend>> (наприклад, функціонал «Редагування тексту» розширюється «Автоперекладом») та <<include>> (наприклад, «Перевірка синтаксису» включає «Інтеграцію із зовнішніми сервісами»).

Для усунення неоднозначності кожен варіант використання деталізується за допомогою сценарію використання — текстового, покрокового опису взаємодії, що включає передумови, постумови, основний потік подій та винятки.

Діаграма класів (Class Diagram)

Діаграма класів є основним інструментом для моделювання статичної структури системи. Вона візуалізує класи, їхні атрибути та операції, а також зв'язки між ними. Для зберігання даних системи була спроєктована логічна модель бази даних у вигляді таблиць, полів та ключів. Проєктування виконано з урахуванням перших трьох нормальних форм (1НФ, 2НФ, 3НФ) для уникнення дублювання даних.

Хід роботи

На початковому етапі проєктування було проведено аналіз функціональних вимог до системи. Визначено, що розроблюваний текстовий редактор має надавати користувачеві повний набір інструментів для роботи з текстовими файлами: їх відкриття, редагування, збереження, перегляду та інтеграції з зовнішніми сервісами.

На основі аналізу були виділені ключові актори системи:

Користувач (User) – основний актор, який взаємодіє з редактором: відкриває та зберігає файли, редагує текст, читає його, використовує макроси та додаткові функції пошуку.

SOA (Service-Oriented Architecture) – зовнішня система або сервіс, з яким відбувається інтеграція для виконання певних функцій, таких як автоматичний переклад або перевірка синтаксису.

Основні варіанти використання, що описують функціональність системи:

Відкрити файл (Open File) – дозволяє користувачу завантажити файл у систему для подальшої роботи.

Зберегти файл (Save File) – забезпечує збереження зміненого тексту.

Редагування тексту (Edit Text) – головна функція, яка включає такі підфункції:

Заміна слова (Replace Word)

Автопереклад тексту (Auto Translate)

Вставка сніпетів (Insert Snippets)

Автопідказка (Autocomplete)

Перевірка синтаксису (Syntax Check)

Використання макросів (Use Macros) – автоматизація повторюваних дій.

Читання (Reading) – перегляд тексту та його елементів, включаючи:

Пошук слова (Search Word)

Перехід до рядка/сторінки (Go to Line/Page)

Закладки (Bookmarks)

Інтеграція із зовнішніми сервісами (Integration with External Services) – дозволяє виконувати автоматичний переклад, підсвічування синтаксису чи інші функції за допомогою зовнішніх рішень.

Для відображення логічних зв’язків між варіантами використання було застосовано відношення <<include>> та <<extend>>. Наприклад, функція
«Редагування тексту» розширюється можливостями автоперекладу, вставки сніпетів або перевірки синтаксису. У свою чергу, перевірка синтаксису включає інтеграцію із зовнішніми сервісами.

Результатом цього етапу є діаграма варіантів використання, яка візуалізує взаємодію користувача із системою текстового редактора та зовнішніми сервісами. Діаграма чітко демонструє основні сценарії використання та зв’язки між ними.

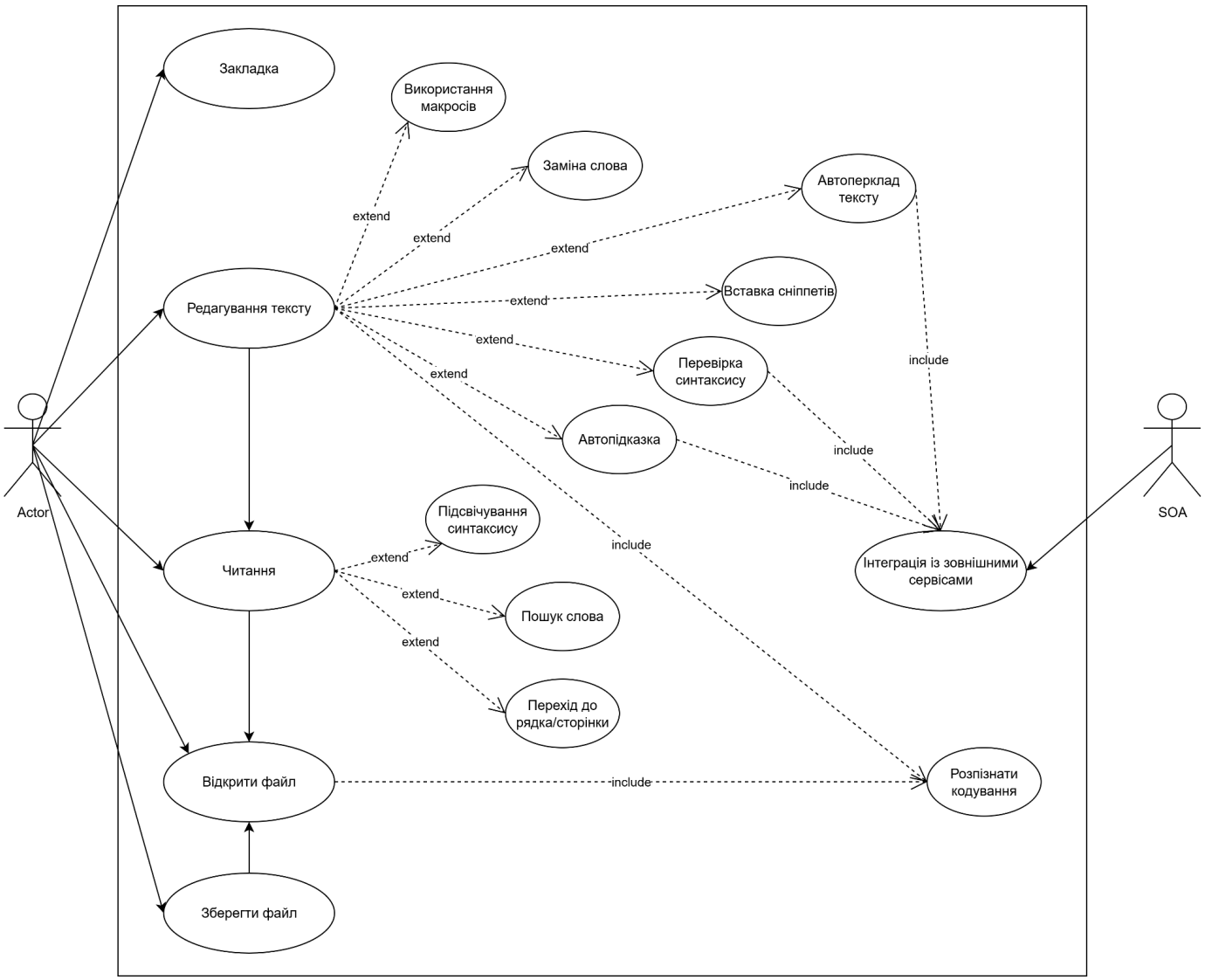

Рис. 1. Діаграма варіантів використання системи

1) Розробка сценаріїв варіантів використання

Сценарій №1: Відкриття та редагування текстового файлу
Передумови: Користувач запустив систему текстового редактора. Постумови:
Успіх: Файл відкрито, відредагований текст збережено.
Провал: Файл не відкрито або зміни не збережено.

Взаємодіючі сторони:
Користувач, Система «Текстовий редактор».

Короткий опис:
Користувач відкриває текстовий файл, вносить зміни до тексту, виконує перевірку синтаксису, заміну слів або використовує автопереклад.

Основний перебіг подій:

1)Користувач натискає «Відкрити файл».
2) Система відкриває діалогове вікно для вибору
файлу. 3)Користувач обирає файл та підтверджує
відкриття.
4)Система завантажує вміст файлу у редактор.
5) Користувач виконує редагування тексту: замінює слова, вставляє сніпети, користується автопідказками або автоперекладом.
6)Користувач запускає перевірку синтаксису.
7) Система виконує перевірку та відображає
результати. 8)Користувач натискає «Зберегти файл».
9)Система зберігає змін

Винятки:
Виняток №1: На кроці 2 користувач натискає «Скасувати» – процес відкриття перервано.
Виняток №2: На кроці 3 обрано файл непідтримуваного формату – система виводить повідомлення: «Неприпустимий формат файлу».
Виняток №3: На кроці 7 система виявила помилки – виводиться звіт із описом помилок.
Виняток №4: На кроці 9 виникла помилка запису – зміни не збережено, система повідомляє: «Помилка збереження».

Сценарій №2: Пошук і навігація по тексту
Передумови: Файл відкрито, користувач у режимі читання або редагування.
Постумови:
Успіх: Користувач знайде потрібне слово або перейде до потрібної сторінки.
Провал: Пошук не дав результатів або перехід не виконано.

Взаємодіючі сторони:
Користувач, Система «Текстовий редактор».

Короткий опис:

Користувач здійснює пошук слова або переходить до потрібного рядка чи сторінки.

Основний перебіг подій:

1)Користувач натискає «Пошук слова».

2)Система відкриває поле пошуку.

3)Користувач вводить слово і підтверджує пошук.

4)Система підсвічує знайдені збіги.

5)Користувач переходить до потрібного рядка або сторінки.

6)Система відображає обрану частину тексту.

Винятки:

Виняток №1: На кроці 3 користувач не ввів слово – система виводить повідомлення: «Введіть пошуковий запит».

Виняток №2: На кроці 4 система не знайшла збігів – повідомлення: «Результатів не знайдено».

Виняток №3: На кроці 5 користувач вказав неіснуючий рядок – повідомлення: «Рядок не знайдено»

Сценарій №3: Інтеграція із зовнішніми сервісами

Передумови:Користувач має активне підключення до зовнішніх сервісів (наприклад, сервіс перекладу або перевірки коду).

Постумови:
Успіх: Текст успішно оброблено зовнішнім сервісом і результати відображено.
Провал: Зовнішній сервіс недоступний або повернув помилку.

Взаємодіючі сторони:
Користувач, Система «Текстовий редактор», Зовнішній сервіс (SOA).

Короткий опис:
Користувач активує функцію автоматичного перекладу тексту або інтегрованої перевірки синтаксису за допомогою зовнішнього сервісу.

Основний перебіг подій:

1) Користувач натискає «Автопереклад тексту» або «Перевірити синтаксис через сервіс».

2)Система надсилає запит до зовнішнього сервісу.

3)Сервіс обробляє текст і надсилає результати назад.

4)Система відображає результати у редакторі.

Винятки:

Виняток №1: На кроці 2 з’єднання з сервісом неможливе – система повідомляє: «Зовнішній сервіс недоступний».

Виняток №2: На кроці 3 сервіс повернув помилку – повідомлення: «Помилка під час обробки запиту».

Виняток №3: На кроці 4 результати не можуть бути застосовані – система виводить повідомлення: «Не вдалося оновити текст».

2) Проектування діаграми класів предметної області

На наступному етапі проєктування була створена деталізована діаграма класів, що відображає не лише основні сутності системи, а і їхні атрибути та поведінку через визначені методи. Ця модель слугує логічним каркасом для майбутньої реалізації.

User (Користувач) — центральна сутність, що ініціює всі основні взаємодії. Клас містить методи для керування життєвим циклом документа: openDocument(), saveDocument() та createDocument(). Крім того, він виступає точкою входу для використання розширених функцій, таких як запуск макросів (runMacro()), створення закладок (createBookmark()) та пошук у документі (searchInDocument()).

Document (Документ) — основний об'єкт, з яким працює користувач. Він інкапсулює дані файлу: заголовок (title), вміст (content) та кодування (encoding). Для маніпуляції вмістом передбачені методи getContent() та setContent(), а також changeEncoding() для зміни кодування. Документ може містити колекцію закладок, якими керує метод addBookmark().

TextEditor (Текстовий редактор) — клас, що відповідає безпосередньо за логіку редагування тексту. Він надає операції, які викликаються для об'єкта Document: replaceWord() (заміна слова), insertCharacter() (вставка символу), autoComplete() (автодоповнення) та checkSyntax() (перевірка синтаксису). Для реалізації пошуку TextEditor використовує функціональність класу Search.

Допоміжні класи:

Search (Пошук): Інкапсулює логіку пошуку за ключовим словом (keyword), надаючи методи findNext() для знаходження наступного входження та findAll() для отримання всіх збігів.

Macro (Макрос): Дозволяє автоматизувати дії. Кожен об'єкт цього класу містить код (code), який виконується за допомогою методу execute().

Bookmark (Закладка): Представляє навігаційний маркер у документі. Містить координати (сторінку page та рядок line) і дозволяє швидко перейти до них за допомогою методу goTo().

Таким чином, розроблена діаграма класів чітко розподіляє обов'язки між об'єктами. Визначення методів дозволяє детально описати динамічну взаємодію між сутностями, що створює міцний фундамент для подальшого проєктування та імплементації архітектури текстового редактора.

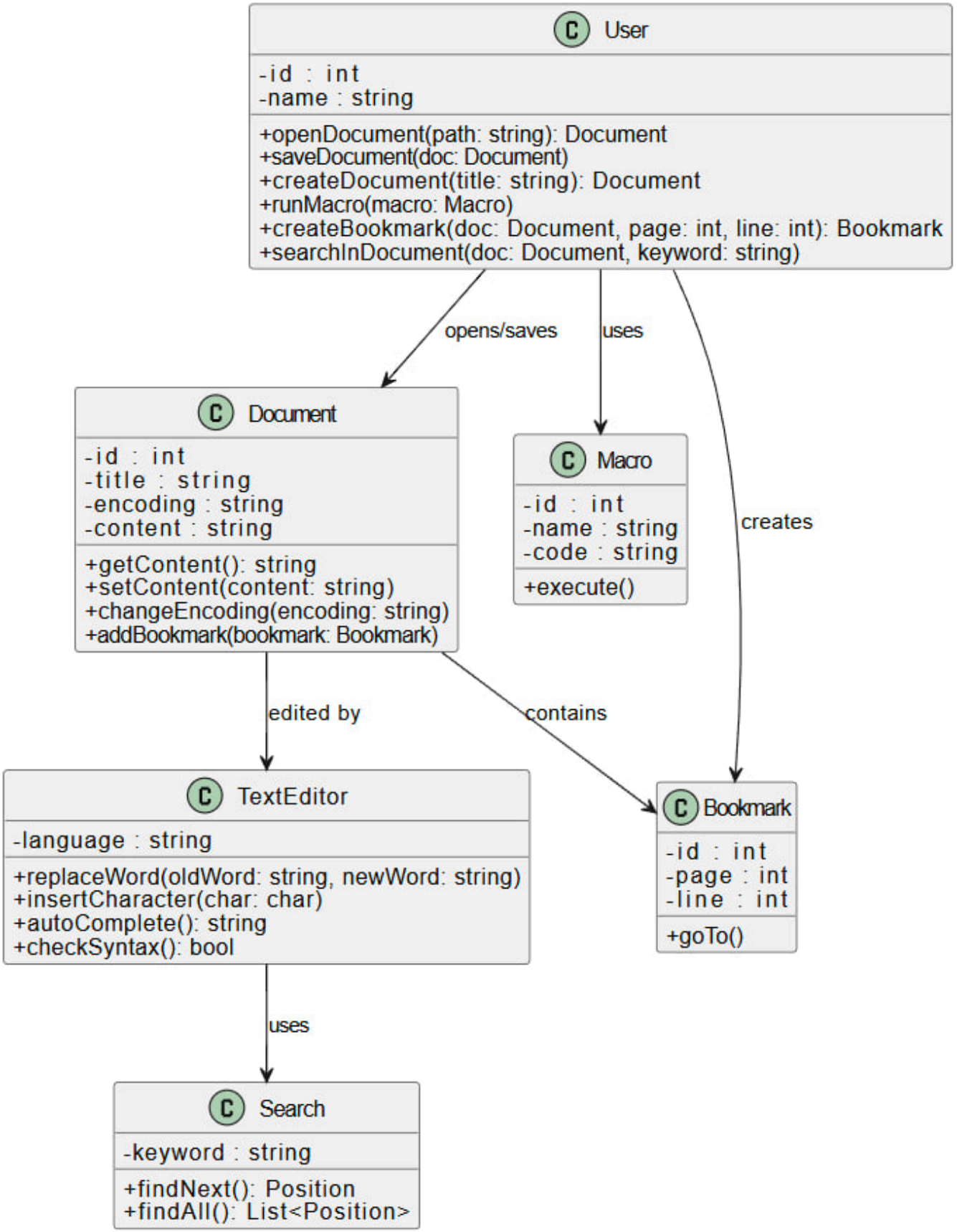

Рис. 2. Діаграма класів предметної області

3) Структура бази даних системи

Для зберігання та управління інформацією про користувачів, документи, макроси, сніппети, історію змін та закладки була розроблена реляційна база даних, структура якої представлена на ER-діаграмі (рис. X.X). Модель бази даних відображає основні сутності предметної області та взаємозв’язки між ними.

User — зберігає відомості про всіх користувачів системи: ім’я, електронну пошту, пароль та дату створення облікового запису.

Document — містить інформацію про документи, що належать користувачам, включаючи заголовок, кодування, вміст, дати створення та оновлення.
Macro — описує макроси, створені користувачами для автоматизації дій у текстовому редакторі.

Snippet — зберігає короткі фрагменти коду чи тексту, які можуть бути повторно використані під час редагування документів.
Bookmark — представляє закладки, створені користувачами для швидкої навігації в документі, містить сторінку, рядок та опис.

History — фіксує історію змін документів, у тому числі тип виконаної дії, її зміст і користувача, який її здійснив.

Між таблицями реалізовані зв’язки типу «один-до-багатьох». Наприклад, один користувач може створити багато документів, макросів або сніппетів, а один документ може мати багато закладок і записів історії. Така структура забезпечує логічну цілісність даних, а також гнучкість у подальшому масштабуванні системи.

Для взаємодії з базою даних у проєкті реалізовано патерн Repository, який забезпечує відокремлення бізнес-логіки застосунку від деталей доступу до даних. Кожна сутність має власний репозиторій (наприклад, UserRepository, DocumentRepository, SnippetRepository тощо), що спрощує роботу з даними через уніфіковані інтерфейси. Це підвищує масштабованість, підтримуваність та зручність подальшої модифікації системи.CR

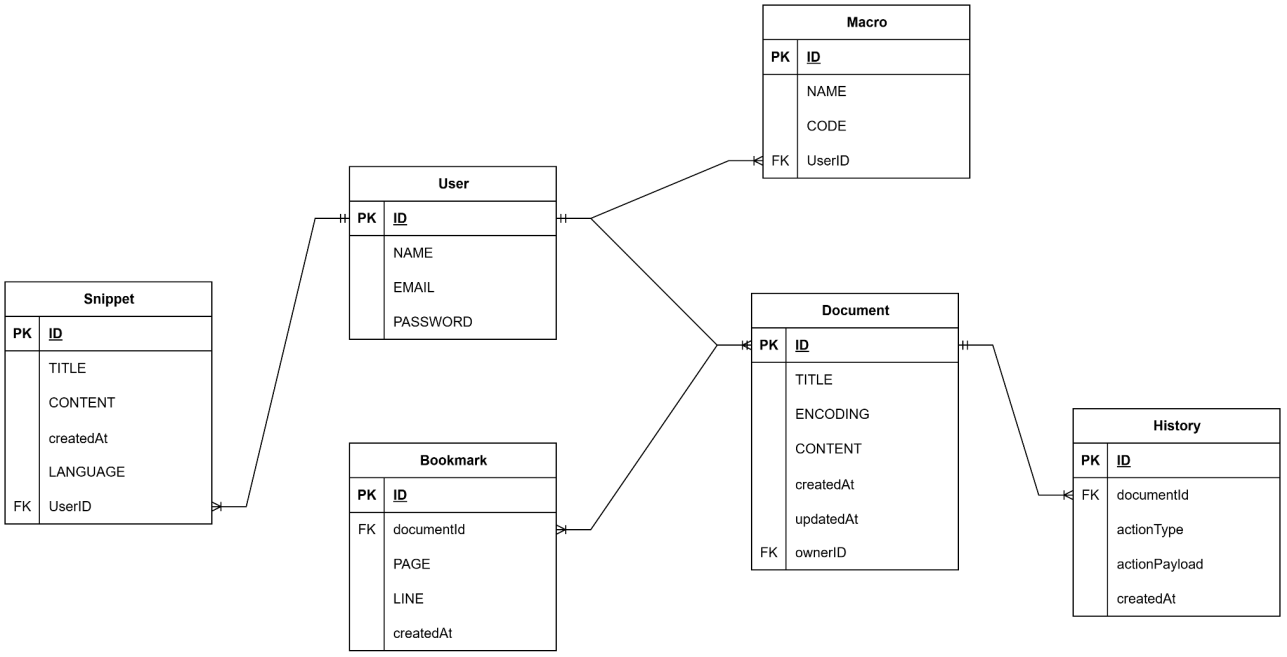

Рис. 3. Схема даних

4) Проектування діаграми класів реалізованої системи

Фінальна діаграма класів відображає архітектуру реалізованої частини системи, яка поєднує сутності предметної області, моделі даних, ключові методи та зв’язки між об’єктами. Вона показує структуру програмного забезпечення з точки зору об’єктно-орієнтованого проєктування, демонструючи, як реалізована бізнес-логіка системи. Побудована модель забезпечує гнучкість, масштабованість, можливість повторного використання компонентів та розширення функціональності.

Основні елементи діаграми:

User – клас, що зберігає інформацію про користувача (ідентифікатор, ім’я, email, пароль, дату створення) та реалізує ключові функції роботи з документами: відкриття, збереження, редагування тексту, пошук, додавання закладок і виконання макросів.

Document – головний елемент системи, який представляє документ з його властивостями (заголовок, кодування, вміст, дата створення та оновлення). Має методи для роботи з текстом (getContent(), setContent(), changeEncoding()), що забезпечують управління вмістом.

Macro – клас, який описує макроси користувача. Містить назву, код, дату створення та метод execute(), що реалізує виконання автоматизованих дій.

Snippet – забезпечує збереження та повторне використання коротких фрагментів коду або тексту з прив’язкою до мови програмування та вмісту.

Bookmark – реалізує механізм створення закладок у документі, зберігаючи сторінку, рядок та опис для швидкої навігації.
History – відповідає за фіксацію історії змін документа: зберігає тип дії, її зміст та час виконання.

TextEditor – клас текстового редактора, який підтримує такі функції, як автодоповнення, перевірка синтаксису, вставка символів і заміна слів.

Search – клас, що реалізує пошук слів або виразів у документі за допомогою методів findNext() та findAll().

Зв’язки між класами:

Користувач створює документи, макроси, сніппети та закладки.

Документ зберігає історію змін, редагується за допомогою текстового редактора, може містити закладки та фрагменти коду.

Макроси можуть змінювати вміст документів.

Редактор використовує функціонал пошуку для роботи з текстом.

Таким чином, фінальна діаграма демонструє узгоджену систему взаємодії класів, де User виступає центральним елементом, а інші класи реалізують окремі аспекти роботи з документами: їх створення, редагування, автоматизацію дій, пошук, історію змін та збереження корисних фрагментів. Така архітектура забезпечує логічну структурованість і підвищує ефективність подальшого розвитку програмного продукту.

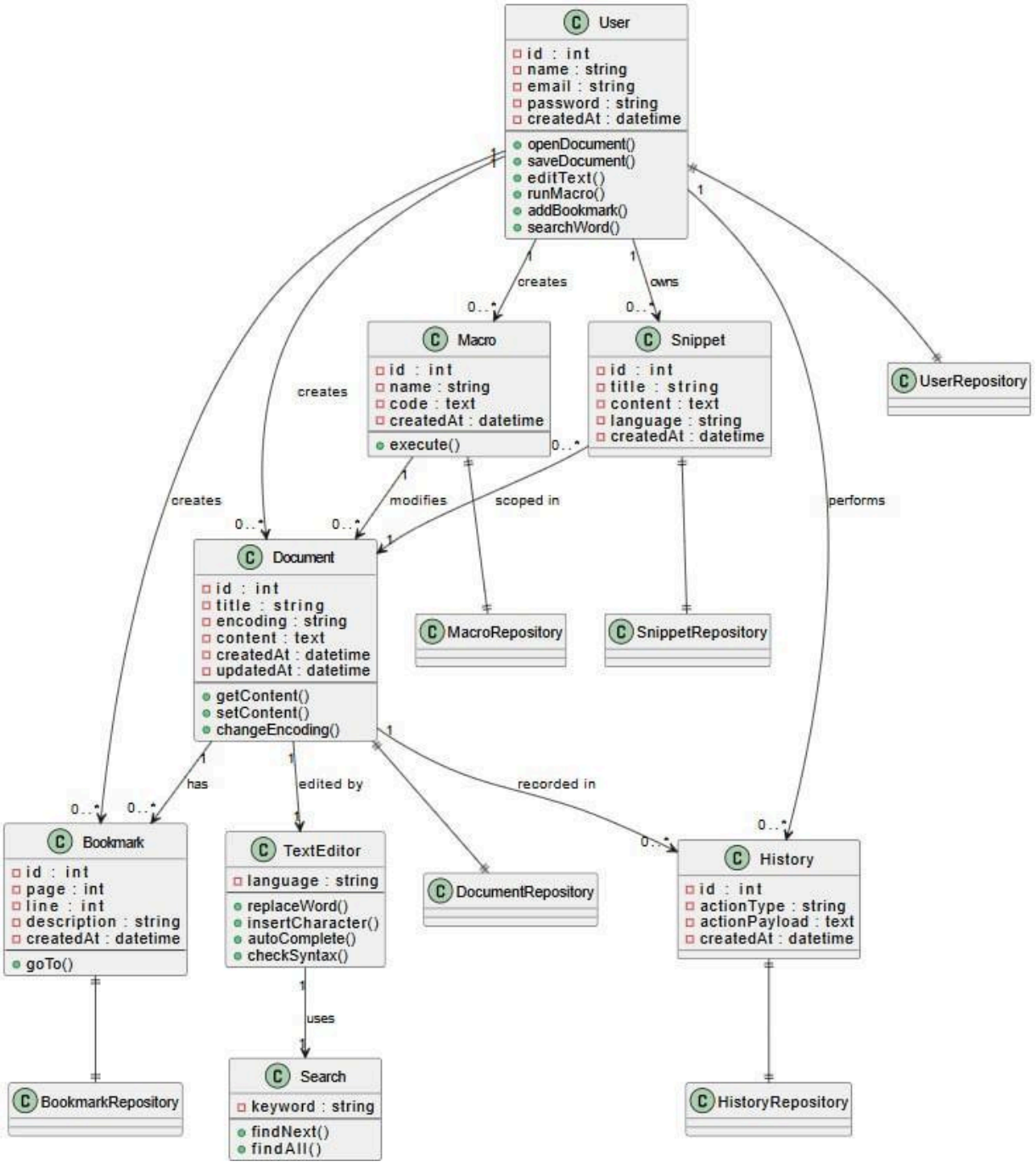

Рис. 4. Діаграма класів реалізованої системи

5)Лістинг коду реалізованих класів системи [Репозиторій]

Model

```java
package model;

import java.util.Date;

public class Bookmark {
    private int id;
    private int page;
    private int line;
    private String description;
    private Date createdAt;

    public void goTo() {}
}
```

Рис. 5. Bookmark.java

```java
package model;

import java.util.Date;

public class Document {
    private int id;
    private String title;
    private String encoding;
    private String content;
    private Date createdAt;
    private Date updatedAt;

    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }
    public void changeEncoding(String encoding) { this.encoding = encoding; }
}
```

Рис. 6. Document.java

```java
package model;

import java.util.Date;

public class History {
    private int id;
    private String actionType;
    private String actionPayload;
    private Date createdAt;
}
```

Рис. 7. History.java

```java
package model;

import java.util.Date;

public class Macro {
    private int id;
    private String name;
    private String code;
    private Date createdAt;

    public void execute() {}
}
```

Рис. 8. Macro.java

```java
package model;

public class Search {
    private String keyword;

    public void findNext() {}
    public void findAll() {}
}
```

Рис. 9. Search.java

```java
package model;

import java.util.Date;

public class Snippet {
    private int id;
    private String title;
    private String content;
    private String language;
    private Date createdAt;
}
```

Рис. 10. Snippet.java

```java
package model;

public class TextEditor {
    private String language;

    public void replaceWord() {}
    public void insertCharacter() {}
    public void autoComplete() {}
    public void checkSyntax() {}
}
```

Рис. 11. TextEditor.java

```java
package model;

import java.util.Date;

public class User {
    private int id;
    private String name;
    private String email;
    private String password;
    private Date createdAt;

    public void openDocument() {}
    public void saveDocument() {}
    public void editText() {}
    public void runMacro() {}
    public void addBookmark() {}
    public void searchWord() {}
}
```

Рис. 12. User.java

Repository

```
1  package repository;
2
3  import model.Bookmark;
4
5  public interface BookmarkRepository extends Repository<Bookmark> {}
```

Рис. 13. BookmarkRepository.java

```
1  package repository;
2
3  import model.Document;
4
5  public interface DocumentRepository extends Repository<Document> {}
```

Рис. 14. DocumentRepository. java

```
1  package repository;
2
3  import model.History;
4
5  public interface HistoryRepository extends Repository<History> {}
```

Рис. 15. HistoryRepository.java

```
1  package repository;
2
3  import model.Macro;
4
5  public interface MacroRepository extends Repository<Macro> {}
```

Рис. 16. MacroRepository.java

```java
package repository;

import model.Snippet;

public interface SnippetRepository extends Repository<Snippet> {}
```

Рис. 17. SnippetRepository.java

```java
package repository;

import model.User;

public interface UserRepository extends Repository<User> {}
```

Рис. 18. UserRepository.java

```java
package repository;

import java.util.List;
import java.util.Optional;

public interface Repository<T> {
    T save(T entity);
    Optional<T> findById(int id);
    List<T> findAll();
    void deleteById(int id);
}
```

Рис. 19. Repository.java

6)Питання до лабораторної роботи

1. Що таке UML? UML (Unified Modeling Language) — це стандартизована мова для візуального моделювання програмного забезпечення.

2. Що таке діаграма класів UML? Діаграма класів — це тип діаграми UML, що моделює статичну структуру системи, показуючи класи, їхні атрибути, операції та зв'язки.

3. Які діаграми UML називають канонічними? Зазвичай так називають 5 основних діаграм: варіантів використання, класів, послідовності, станів та діяльності.

4. Що таке діаграма варіантів використання? Діаграма варіантів використання фіксує функціональність системи та її межі, показуючи взаємодію акторів із системою.

5. Що таке варіант використання? Варіант використання — це послідовність дій, яку система виконує для досягнення мети актора.

6. Які відношення можуть бути відображені на діаграмі використання? На діаграмі використання відображають відношення асоціації, <<include>> (включення) та <<extend>> (розширення).

7. Що таке сценарій? Сценарій — це покроковий текстовий опис варіанту використання, що включає передумови, потік подій та винятки.

8. Що таке діаграма класів? Діаграма класів моделює статичну структуру системи, візуалізуючи класи, їхні атрибути, операції та зв'язки між ними.

9. Які зв’язки між класами ви знаєте? Основні типи зв'язків між класами: асоціація, агрегація, композиція, залежність та успадкування.

10. Чим відрізняється композиція від агрегації? Відмінність між композицією та агрегацією полягає в управлінні життєвим циклом. При агрегації "частини" можуть існувати окремо від "цілого", а при композиції — ні.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів? На діаграмах класів агрегація позначається незафарбованим ромбом , а композиція — зафарбованим з боку класу-"цілого".

12. Що являють собою нормальні форми баз даних? Нормальні форми — це правила проєктування баз даних для усунення надлишковості та забезпечення цілісності даних.

13. Що таке фізична модель бази даних? Логічна? Логічна модель бази даних описує, що за дані зберігаються (таблиці, зв'язки). Фізична модель описує, як вони зберігаються (типи даних, індекси).

14. Який взаємозв'язок між таблицями БД та програмними класами? Класи є програмним представленням сутностей, а таблиці БД зберігають їхні дані. Патерн Repository використовується для зв'язку між ними, абстрагуючи доступ до даних від бізнес-логіки.

Висновок

У результаті виконання цієї лабораторної роботи були засвоєні та застосовані основні принципи об'єктно-орієнтованого аналізу та проєктування на практичному прикладі створення архітектури багатофункціонального «Текстового редактора».

На початковому етапі було виконано аналіз вимог та побудовано діаграму варіантів використання, що дозволило чітко визначити функціональні можливості системи та ролі акторів. На основі цього було розроблено детальні сценарії, які лягли в основу подальшого проєктування.

Було створено логічну модель предметної області та деталізовану діаграму класів, що визначили ключові сутності системи та їхню взаємодію. Для забезпечення надійного зберігання даних було спроєктовано схему бази даних та впроваджено патерн Repository, який відокремлює бізнес-логіку від механізмів доступу до даних.

Таким чином, у ході роботи було створено повний архітектурний проєкт текстового редактора, який є добре структурованим, гнучким та готовим до подальшої розробки.

.