

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 № 3

Технології розроблення програмного забезпечення
«Основи проектування розгортання»

Виконала:

студентка групи ІА-32

Чайковська С. В.

Перевірив:

Мягкий М. Ю.

Київ 2025

Зміст

1 Теоретичні відомості.....	3
2 Тема.....	4
3 Діаграма розгортання.....	5
4 Діаграма компонентів.....	7
5 Діаграма послідовностей.....	10
6 Структура проєкту та візуальні форми.....	13
7 Висновки.....	19
8 Відповіді на питання.....	20

Тема: Основи проектування розгортання.

Мета: Навчитися проектувати діаграми розгортання та компонентів для системи що проектується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

1 Теоретичні відомості

Діаграма розгортання

Діаграми розгортання (Deployment Diagram) відображають фізичне розташування компонент системи та показують, на якому обладнанні запускається програмне забезпечення.

Основними елементами є вузли, що можуть бути:

- Пристроями (фізичне обладнання, наприклад, комп'ютери);
- Середовищами виконання (програмне забезпечення, яке запускає інші компоненти, наприклад, операційна система).

Вузли можуть бути з'єднані шляхами, що показують взаємодію між ними, з використанням різних протоколів чи технологій. Усередині вузлів можуть бути артефакти — файли, які реалізують компоненти системи (виконувані файли, дані тощо).

Діаграми розгортання можуть бути:

- Описовими, які показують загальну структуру без деталей обладнання.
- Екземплярними, що включають конкретні приклади обладнання та програмних компонентів, і використовуються на пізніх етапах розробки для планування розгортання системи.

Діаграма компонентів

Діаграма компонентів UML відображає систему, розбиту на окремі модулі.

Виділяють три типи діаграм:

- Логічні
- Фізичні
- Виконувані

Найчастіше використовують логічне розбиття — система представлена як набір незалежних компонентів, що взаємодіють між собою. Наприклад, система продажу продуктів може мати такі компоненти: каса, сервер продаж, система обліку тощо. Компоненти можуть бути розміщені на різних фізичних пристроях або в різних процесах.

Фізичне розбиття фокусується на розподілі компонент між серверами та комп'ютерами, що частіше відображається на діаграмі розгортання.

Виконване розбиття показує компоненти як файли (наприклад, .exe, html, бази даних), даючи інший ракурс системи. Однак цей підхід не має широкого використання через наявність діаграм розгортання.

Діаграми послідовностей

Діаграми послідовностей в UML використовуються для моделювання виконання операцій і взаємодії між об'єктами в системі. Вони відображають послідовність повідомлень між об'єктами у процесі виконання операцій, деталізуючи алгоритми і логіку.

Основні елементи діаграми послідовностей:

- Актори або об'єкти, які взаємодіють між собою.
- Повідомлення, що передаються між ними.
- Лінії життя, які показують час існування об'єкта або актора під час взаємодії.

Діаграми послідовностей використовуються для візуалізації порядку виконання дій і детальної реалізації алгоритмів, відображаючи, як операції переходять із одного стану в інший.

2 Тема

3. Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

3 Діаграма розгортання

Діаграма розгортання зображена на Рисунку 1:

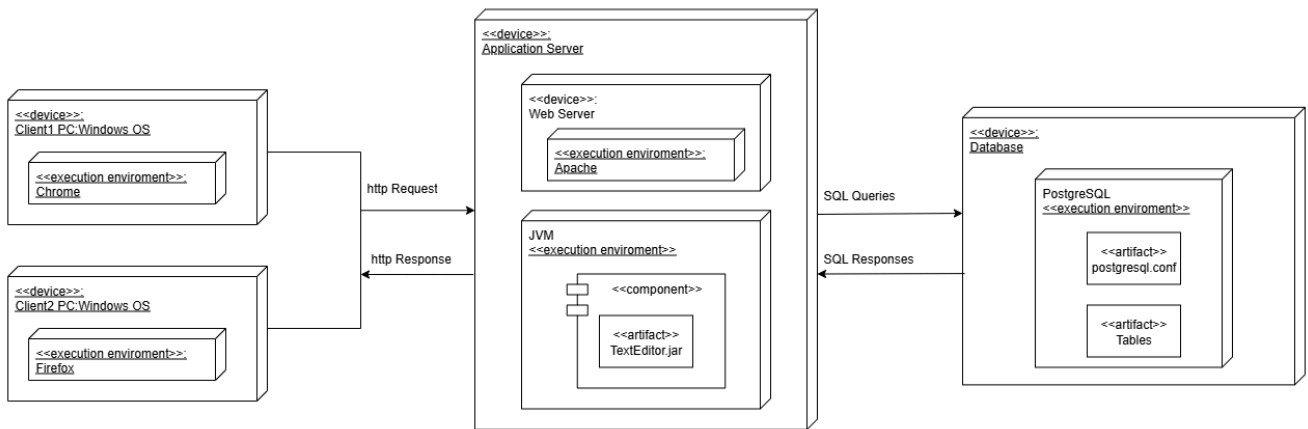


Рис. 1 – Діаграма розгортання

Ця діаграма розгортання описує архітектуру системи, що складається з трьох основних частин: пристрої користувачів, серверу додатків та бази даних.

Пристрої користувачів:

- Пристрої користувачів <<device>> представлені двома комп'ютерами з операційною системою Windows.
- На цих пристроях є середовища виконання <<execution environment>>:
 - Chrome — браузер, через який користувач надсилає HTTP-запити.
 - Firefox — альтернативний браузер для доступу до системи.

Обидва пристрої взаємодіють із сервером додатків через веб-браузери.

Сервер додатків:

- Сервер додатків <<device>> Application Server включає два основні середовища виконання:
 - Web Server (Apache) <<execution environment>> — відповідає за обробку HTTP-запитів та передачу їх у середовище JVM.
 - JVM (Java Virtual Machine) <<execution environment>>, в якому розміщено:
 - Компонент <<component>> — логіка веб-додатку.

- Артефакт <<artifact>> TextEditor.jar — згенерований файл програми (Spring Boot-додаток).

Сервер одночасно містить веб-сервер для обробки запитів і JVM для виконання бізнес-логіки додатку.

База даних:

- База даних <<device>> Database включає середовище виконання PostgreSQL.

- PostgreSQL містить:

- Артефакт <<artifact>> postgresql.conf — конфігураційний файл бази даних.
- Артефакт <<artifact>> Tables — структуру таблиць для зберігання інформації системи.

Взаємодія між компонентами:

- HTTP Request — запит, що надсилається з браузера клієнта до сервера додатків.

- HTTP Response — відповідь сервера, що повертається на клієнтський пристрій.

- SQL Queries — запити, які JVM надсилає до бази даних PostgreSQL.

- SQL Responses — результати виконання запитів, які база даних повертає серверу додатків.

4 Діаграма компонентів

Діаграма компонентів зображена на Рисунку 2:

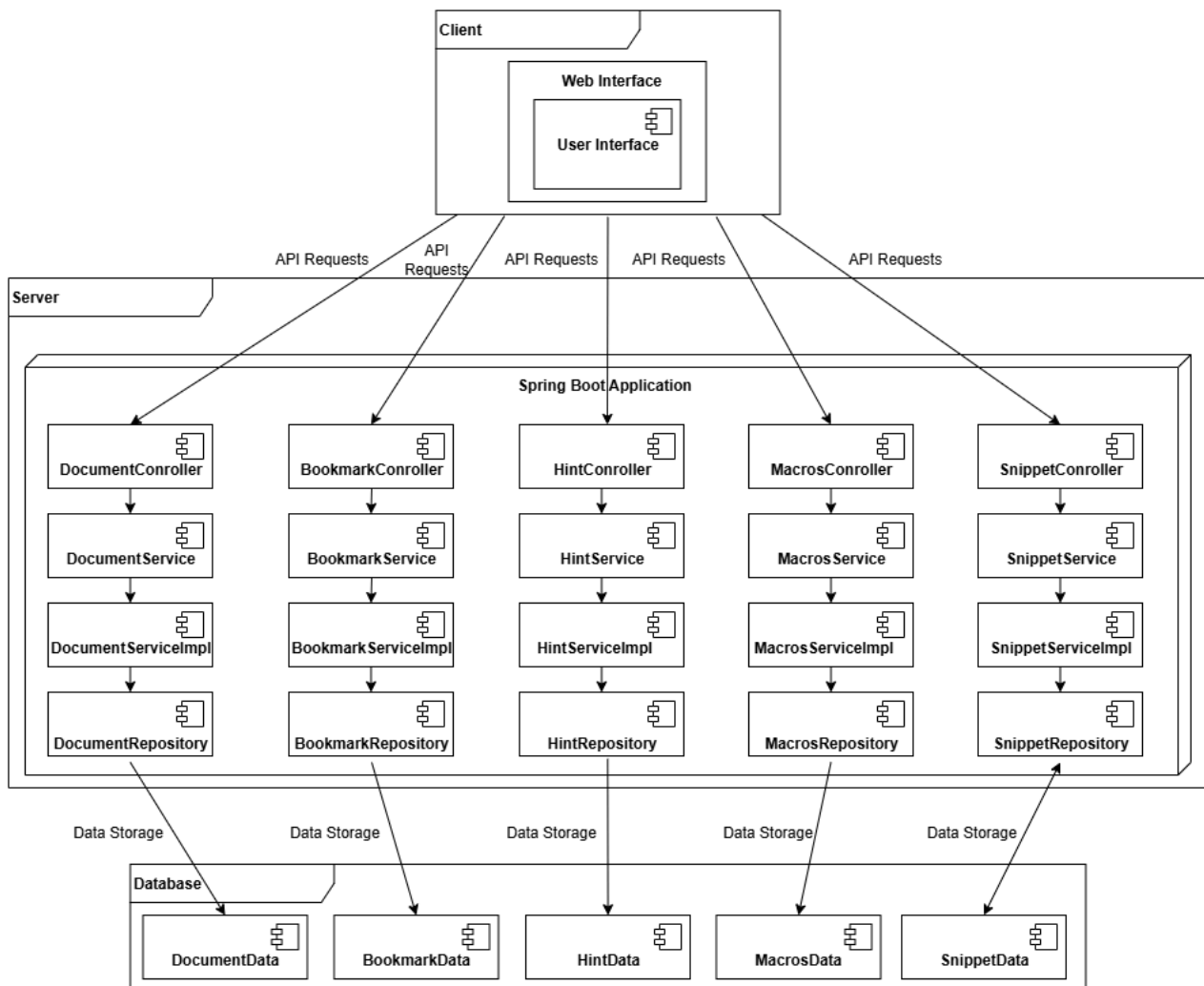


Рис. 2 – Діаграма компонентів

Ця діаграма компонентів, яку ми бачимо на рисунку, зображує архітектуру текстового редактора, що складається з різних компонентів, організованих за функціональними рівнями на стороні клієнта, сервері застосунків та сервері бази даних.

1. Клієнт (Client):

Користувач взаємодіє з текстовим редактором через Web Interface та User Interface, надсилаючи API-запити для виконання різних функцій, таких як управління документами, закладками, підказками, макросами та фрагментами коду.

2. Сервер застосунків (Application Server):

Містить логіку обробки запитів клієнта та складається з трьох рівнів:

- Контролери (Controllers):

Приймають запити від клієнта та перенаправляють їх на відповідні сервіси для обробки:

- *DocumentController*
- *BookmarkController*
- *HintController*
- *MacrosController*
- *SnippetController*

- Сервіси (Services):

Реалізують бізнес-логіку для кожного типу операцій у текстовому редакторі:

- *DocumentService*
- *BookmarkService*
- *HintService*
- *MacrosService*
- *SnippetService*

- Реалізації сервісів (ServiceImpl):

Містять фактичну реалізацію методів сервісів і взаємодіють з репозиторіями для доступу до даних:

- *DocumentServiceImpl*
- *BookmarkServiceImpl*
- *HintServiceImpl*
- *MacrosServiceImpl*
- *SnippetServiceImpl*

- Репозиторії (Repositories):

Кожен сервіс використовує відповідний репозиторій для роботи з даними. Репозиторії є інтерфейсом для взаємодії з базою даних:

- *DocumentServiceImpl*

- *BookmarkServiceImpl*
- *HintServiceImpl*
- *MacrosServiceImpl*
- *SnippetServiceImpl*

3. Сервер бази даних (Database Server):

Зберігає всі дані, що необхідні для роботи текстового редактора:

- *DocumentData* — дані, пов'язані з документами.
- *BookmarkData* — дані закладок.
- *HintData* — дані підказок.
- *MacrosData* — дані макросів.
- *SnippetData* — дані фрагментів коду.

Взаємодії:

- Клієнт надсилає API-запити до контролерів.
- Контролери викликають відповідні сервіси.
- Сервіси взаємодіють зі своїми реалізаціями, які звертаються до репозиторіїв.
- Репозиторії виконують запити до бази даних для збереження та отримання потрібних даних.

Ця діаграма демонструє чітку модульну структуру застосунку, де кожен компонент має конкретну роль, що забезпечує легке розширення, модифікацію, тестування та підтримку системи текстового редактора.

5 Діаграма послідовностей

Діаграма послідовності (1) зображена на рисунку 3:

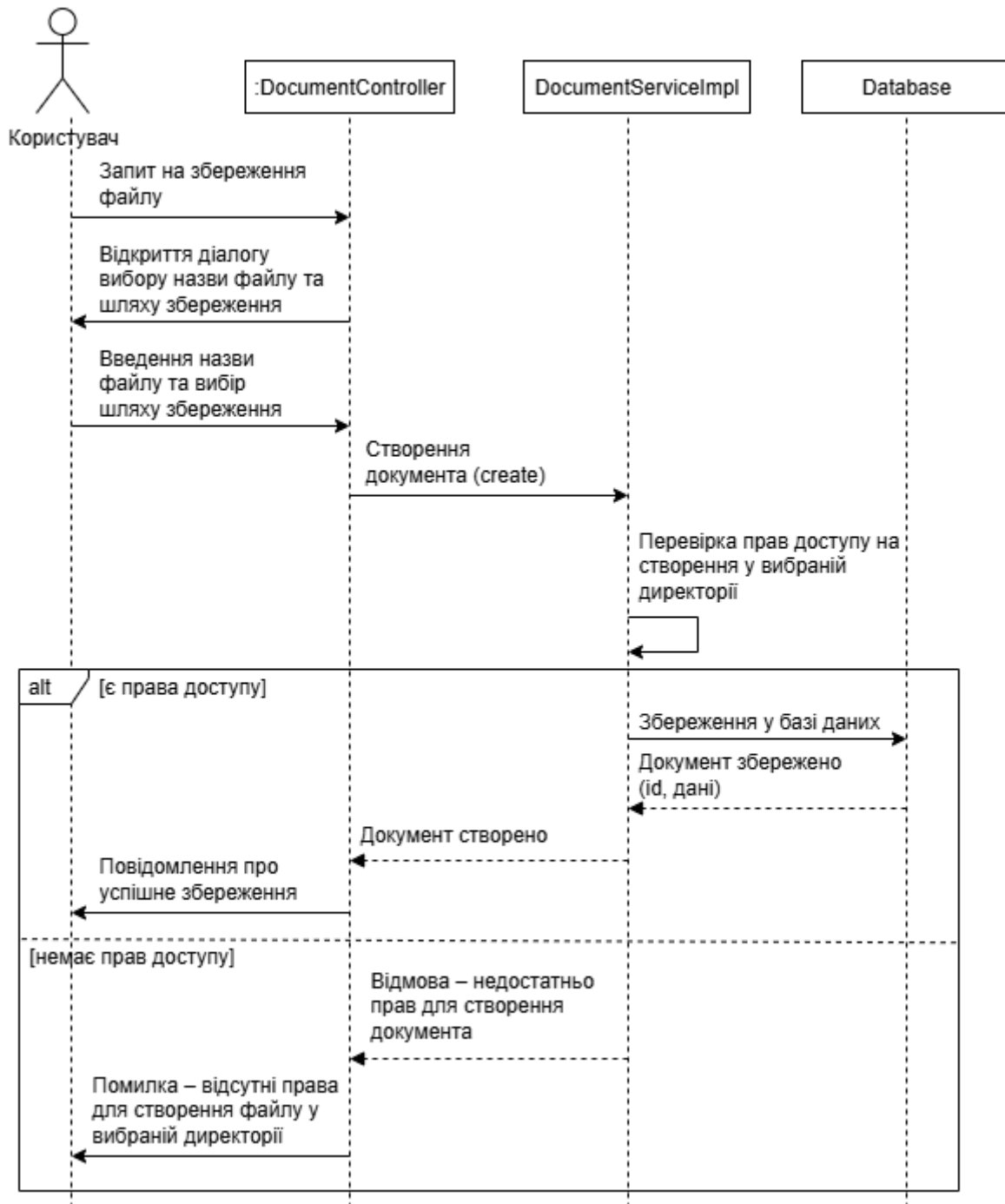


Рис. 3 – Діаграма послідовності «Збереження текстового файлу»

Ця діаграма ілюструє послідовність взаємодій між компонентами системи під час збереження текстового файлу. Вона демонструє, як користувач,

контролер, сервіс та база даних взаємодіють для створення документа, враховуючи перевірку прав доступу.

Збереження текстового файлу:

- Користувач ініціює процес збереження, надсилаючи запит до *DocumentController*.
- Контролер відкриває діалог, у якому користувач вводить назву файлу та обирає шлях збереження.
- Користувач вводить дані й передає їх назад у *DocumentController*.
- Контролер викликає метод створення документа у *DocumentServiceImpl*, передаючи введену інформацію.
- Сервіс перевіряє права доступу користувача на створення файлу у вибраній директорії.

Успішний сценарій (є права доступу):

- Сервіс надсилає запит до бази даних для збереження документа.
- База даних повертає підтвердження разом з даними збереженого документа.
- Сервіс передає створений документ назад у *DocumentController*.
- Контролер відправляє користувачу повідомлення про успішне збереження.

Альтернативний сценарій (немає прав доступу):

- Сервіс повертає контролеру повідомлення про відмову через недостатність прав.
- Контролер повідомляє користувача про помилку – відсутні права для створення файлу у вибраній директорії.

Діаграма послідовності (2) зображена на рисунку 4:

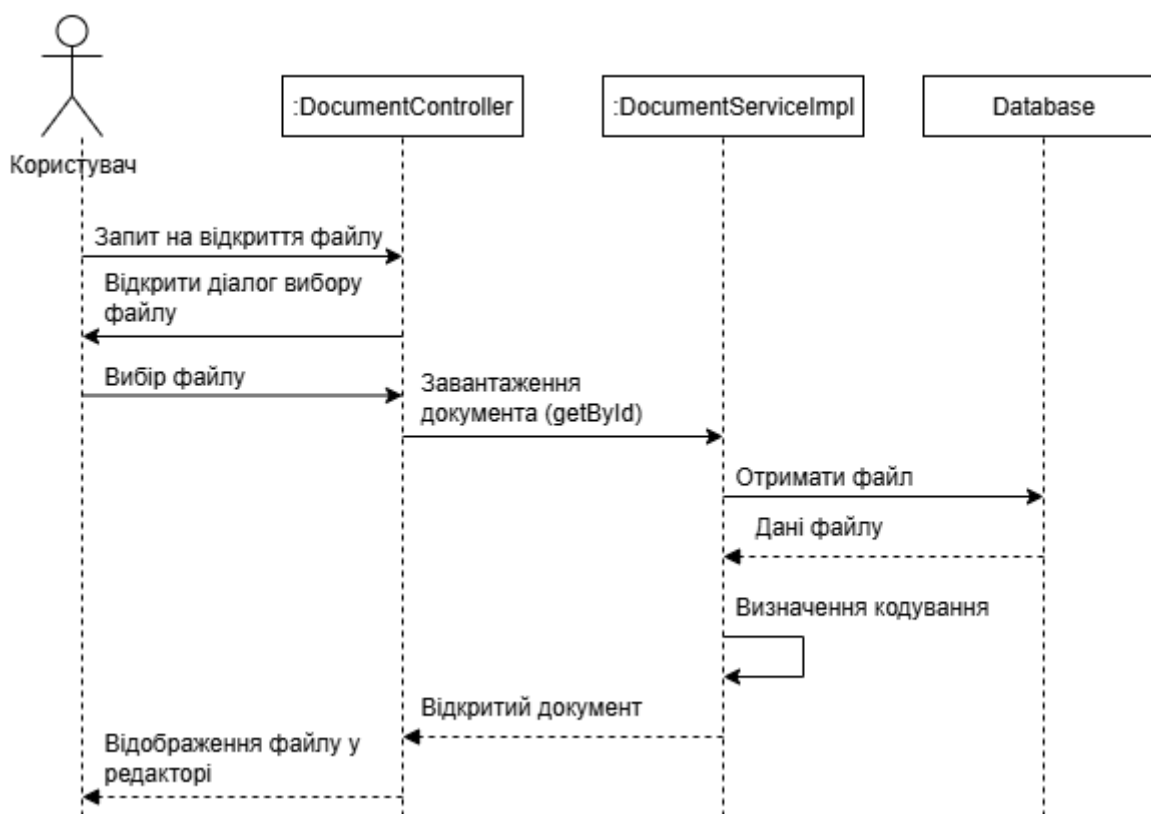


Рис. 4 – Діаграма послідовності «Відкриття текстового файлу»

Ця діаграма ілюструє послідовність взаємодій між компонентами текстового редактора під час відкриття існуючого текстового файлу. Вона демонструє, як користувач, контролер, сервіс та база даних взаємодіють для отримання та відображення документа.

Відкриття текстового файлу:

Користувач ініціює процес відкриття файлу, відправляючи запит до DocumentController.

DocumentController відкриває діалог вибору файлу для користувача.

Користувач обирає файл і надсилає вибір назад у контролер.

Контролер викликає метод getById у DocumentServiceImpl для завантаження документа.

Сервіс надсилає запит у базу даних для отримання даних файлу.

База даних повертає дані файлу у сервіс.

Сервіс визначає кодування документа та підготовлює його до

відображення.

Підготовлений документ повертається у контролер.

Контролер відображає файл у редакторі для користувача.

6 Структура проєкту та візуальні форми

Структура проєкту:

Структура проєкту зображена на Рисунку 5:

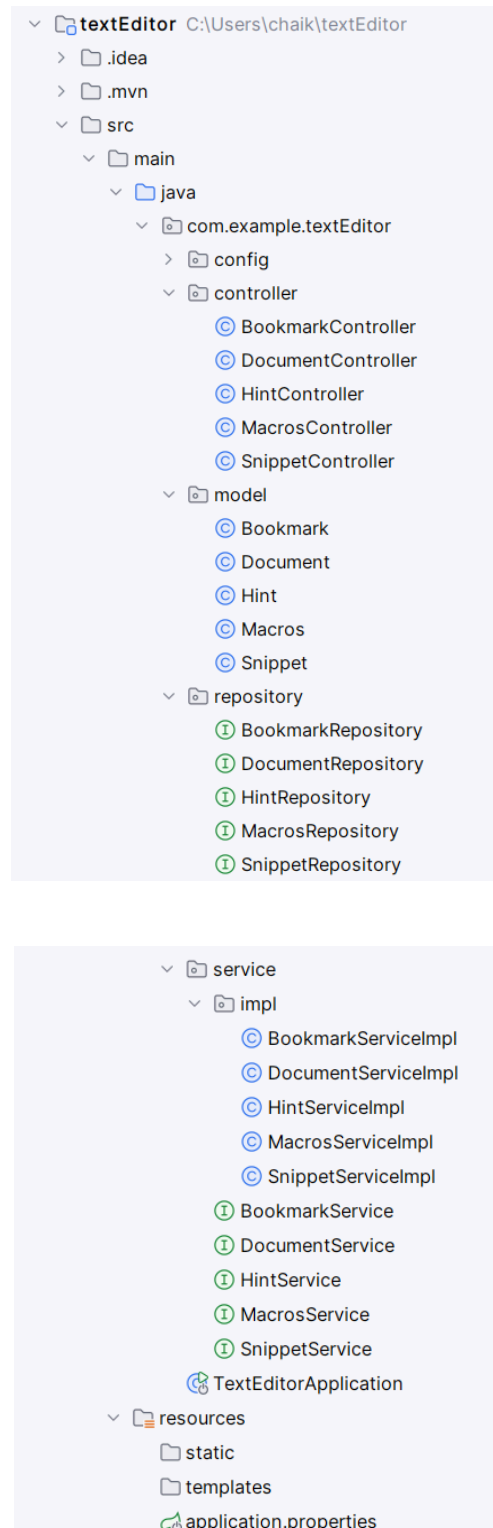


Рис. 5 – Структура проєкту

Структура проєкту TextEditor:

Архітектура побудована за принципом багаторівневого розподілу (MVC + Service + Repository), що забезпечує чітке розділення відповідальностей:

- *controller* – містить контролери, які приймають запити від користувача, викликають бізнес-логіку та повертають відповіді на UI:

DocumentController, BookmarkController, HintController, MacrosController, SnippetController – керують операціями з відповідними сутностями (документи, закладки, підказки, макроси, фрагменти коду).

- *model* – включає моделі даних (сутності), що відображають таблиці бази даних:

Document, Bookmark, Hint, Macros, Snippet. Кожна з моделей описує структуру даних, які зберігаються у БД.

- *repository* – шар доступу до даних. Містить інтерфейси для роботи з БД:

DocumentRepository, BookmarkRepository, HintRepository, MacrosRepository, SnippetRepository. Завдяки Spring Data JPA забезпечується збереження, пошук та вибірка даних без написання SQL-запитів вручну.

- *service* – шар бізнес-логіки. Інтерфейси:

DocumentService, BookmarkService, HintService, MacrosService, SnippetService.

- *Реалізації (напка impl):*

DocumentServiceImpl, BookmarkServiceImpl, HintServiceImpl, MacrosServiceImpl, SnippetServiceImpl. У цьому шарі виконується логіка обробки даних, яка відділяє контролери від репозиторіїв.

- *TextEditorApplication* – головний клас, що запускає Spring Boot застосунок.

Така структура проєкту дає змогу легко підтримувати й розширювати систему, оскільки контролери, бізнес-логіка та доступ до даних розділені по різних шарах. Це відповідає стандартам побудови сучасних корпоративних застосунків.

Візуальні форми системи

У поточній реалізації програмної системи TextEditor є дві основні візуальні форми:

1. Форма налаштування нового документа, зображена на Рисунку 6:

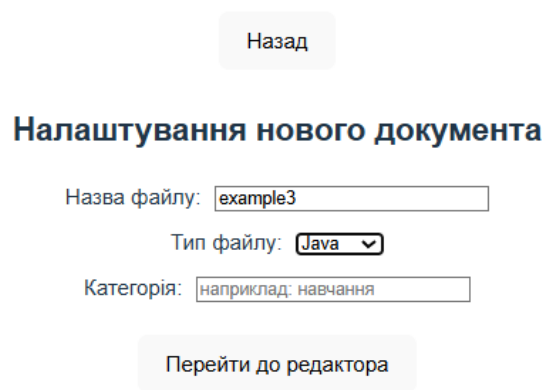


Рис. 6 – Форма налаштування нового документа

Ця форма відповідає за створення нового документа і містить такі поля:

- Назва файлу – обов’язкове поле для введення імені документа.
- Тип файлу (розширення) – випадаючий список (txt, py, java, html, css, xml, json), що визначає формат документа.
- Категорія – текстове поле для вказівки категорії (наприклад, навчання, робота).

Також передбачена валідація: якщо назва файлу порожня, користувач бачить повідомлення “Назва файлу обов’язкова”.

Після успішного заповнення полів і натискання кнопки «Перейти до редактора»:

1. Виконується POST-запит на бекенд (http://localhost:8081/api/documents) для створення нового запису в БД.

2. У базі даних зберігається документ із введеними параметрами.
3. Користувача автоматично переадресовує на форму Редактора документа з передачею ID документа у запиті.

2. Форма редагування документа (Редактор), зображена на Рисунку 7:

Назад

Редактор документа

Назва файлу:

Розширення:

Вміст:

```
package com.example.textEditor.repository;

import com.example.textEditor.model.Document;
import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DocumentRepository extends JpaRepository<Document, Integer> {

    @EntityGraph(attributePaths = {"bookmarks", "snippets", "hints", "macros"})
    Document findWithAllDependenciesById(int id);
}
```

Зберегти

Рис. 7 – Форма редагування документа

Форма відкривається після створення нового документа або після вибору локального файлу.

Вона містить:

- Поле Назва файлу (можна змінювати).
- Поле Розширення (список підтримуваних розширень).
- Велике текстове поле для редагування вмісту документа.
- Кнопку «Зберегти» для збереження файлу.

Механізм збереження документа:

1. Виконується перевірка: чи не є вміст файлу порожнім.
 - Якщо поле порожнє → повідомлення “Вміст файлу обов’язковий”.

2. Застосовується функція `validateContent`, яка перевіряє, чи відповідає вміст формату вибраного файлу (наприклад, чи JSON є валідним, чи HTML містить `<html>`).

3. Якщо вміст не відповідає розширенню:

- з'являється діалогове вікно, зображене на Рисунку 8, з рекомендацією іншого формату (наприклад, `.json` замість `.txt`).
- користувач може:
 1. змінити і зберегти;
 2. залишити як є;
 3. скасувати збереження.

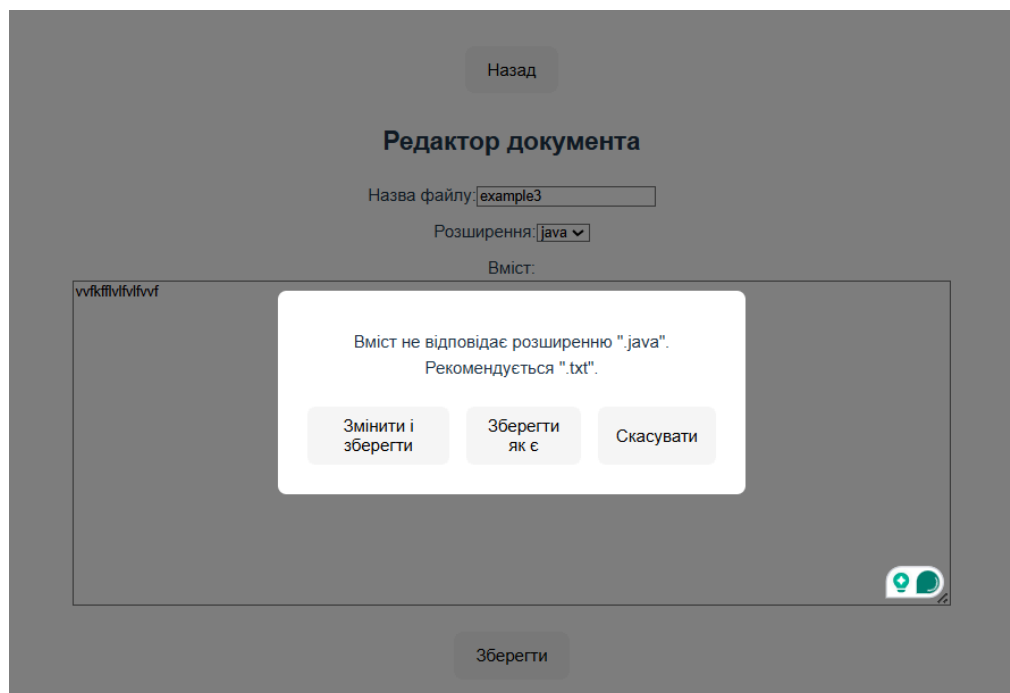


Рис. 8 – діалогове вікно (з'являється, якщо вміст не відповідає розширенню)

4. Після підтвердження користувачем виконується локальне збереження через API браузера `showSaveFilePicker` – створюється файл на комп'ютері користувача з вибраним іменем та розширенням.

5. Паралельно документ відправляється на бекенд:

■ Якщо документ новий → POST-запит (/api/documents).

Якщо документ редагується → PUT-запит (/api/documents/{id}).

6. У результаті файл зберігається як локально на диску користувача, так і в базі даних.

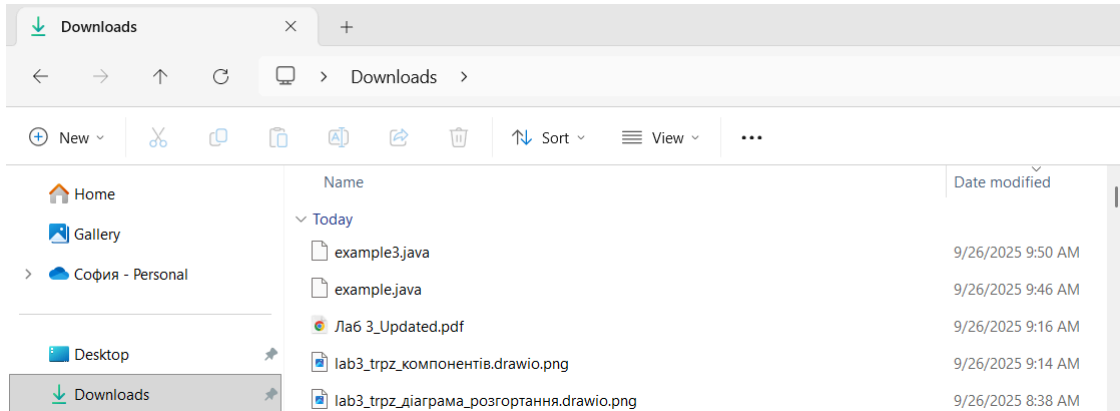


Рис. 9 – Результат збереження файлу

3. Головна форма (HomePage), зображена на Рисунку 10:

Текстовий редактор

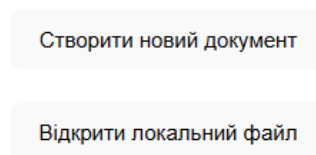


Рис. 10 – Головна форма

Додатково існує стартова форма з меню, яка дає користувачеві можливість:

- Створити новий документ (перехід на форму налаштувань).
- Відкрити локальний файл (через діалог вибору файлу). У такому випадку файл одразу відкривається у редакторі з передачею його вмісту.

Таким чином, система реалізує повний цикл роботи з документами:

- Створення документа через форму налаштування.
- Редагування і збереження документа через форму редактора.
- Вибір локальних файлів і роботу з ними.

Система реалізує архітектуру багаторівневого розподілу (UI → Controller → Service → Repository → Database), що дає змогу організувати повний цикл роботи з даними: від введення їх на формі, передачі на бекенд і збереження у БД — до подальшої вибірки з бази даних і відображення актуальної інформації на інтерфейсі користувача.

Посилання на GitHub: <https://github.com/chaikovsska/textEditor>

7 Висновки

У ході виконання лабораторної роботи було проведено аналіз архітектури системи та створено кілька UML-діаграм, які дозволили детально зрозуміти її структуру та функціонування. Діаграма розгортання допомогла візуалізувати фізичне розташування компонентів системи та їх взаємодію в реальному середовищі, що сприяло визначенню ефективної конфігурації інфраструктури. Діаграма компонентів відобразила логічну структуру програмного забезпечення, зв'язки між його модулями та ключові інтерфейси, що забезпечило розуміння взаємозалежностей між різними частинами системи. Діаграма взаємодій і послідовностей продемонструвала динаміку виконання запитів, порядок обробки операцій та зв'язок між користувачем, системою і базою даних, що дало змогу побудувати чітку модель поведінки програми. Усі ці діаграми стали важливим інструментом для проєктування, аналізу та документування системи, допомогли виявити слабкі місця архітектури й оптимізувати її для подальшої реалізації.

8 Відповіді на питання

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) показує фізичне розташування системи, тобто на якому обладнанні і в якому середовищі виконання працюють складові програмного забезпечення. Основна мета – візуалізація розгортання програмних артефактів на вузлах.

2. Які бувають види вузлів на діаграмі розгортання?

Вузли бувають двох типів:

- Пристрій (device) – фізичне обладнання, наприклад комп'ютер або сервер.
- Середовище виконання (execution environment) – програмне забезпечення, яке може включати інші програми (наприклад, операційна система, вебсервер, контейнер).

3. Які бувають зв'язки на діаграмі розгортання?

Зв'язки між вузлами показують взаємодію та можуть містити:

- Пряму лінію між вузлами.
- Назву або протокол взаємодії (HTTP, IPC, WCF тощо).
- Атрибути множинності (наприклад, кількість клієнтів, підключених до сервера).

4. Які елементи присутні на діаграмі компонентів?

Основні елементи:

- Компоненти – автономні модулі або файли системи (логічні, фізичні або виконувані).
- Файли/артефакти – виконувані файли (.exe, .dll), вихідні файли, HTML-сторінки, таблиці баз даних, документи тощо.
- Залежності між компонентами – показують використання класів одного компонента іншими.

5. Що становлять собою зв'язки на діаграмі компонентів?

Зв'язки показують залежності між компонентами: один компонент використовує класи або ресурси іншого. Це допомагає зрозуміти, як зміни в одному компоненті вплинуть на інші.

6. Які бувають види діаграм взаємодії?

Діаграма послідовностей (Sequence Diagram)

- Показує обмін повідомленнями між об'єктами у часовій послідовності.
- Використовується для моделювання логіки сценаріїв та бізнес-процесів.

Діаграма комунікацій (Communication / Collaboration Diagram)

- Фокус на зв'язках між об'єктами та структурі їх взаємодії.
- Повідомлення пронумеровані для показу послідовності.

Діаграма взаємодії з часовими обмеженнями (Timing Diagram)

- Показує зміну станів об'єктів у часі.
- Використовується для систем, де важлива синхронізація.

Діаграма огляду взаємодії (Interaction Overview Diagram)

- Комбінує елементи діаграми послідовностей та активностей.
- Показує загальну логіку взаємодії системи на високому рівні.

7. Для чого призначена діаграма послідовностей?

Вона моделює взаємодію між об'єктами системи у часовій послідовності, показуючи порядок і логіку виконання операцій. Використовується для аналізу бізнес-процесів, проектування архітектури та тестування.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Актори (Actors) – користувачі або зовнішні системи.
- Об'єкти або класи – з лінією життя (Life Line).

- Повідомлення – стрілки між об'єктами (синхронні або асинхронні).
- Активності (Activations) – періоди виконання дії.
- Контрольні структури – умови, цикли, альтернативи (alt, loop).

9. Як діаграми послідовностей пов'язані з діаграмами варіантів використання?

Діаграми послідовностей деталізують сценарії, які описані на діаграмах варіантів використання. Вони показують, як саме актори і об'єкти взаємодіють для реалізації конкретного випадку використання.

10. Як діаграми послідовностей пов'язані з діаграмами класів?

Об'єкти на діаграмі послідовностей представляють класи та їх екземпляри з діаграм класів. Послідовність викликів методів між об'єктами відповідає логіці взаємодії між класами.