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**»

Основи проектування.

Виконав:

Студент групи ІА-31

Шереметьєв Дмитро

Київ – 2025

Тема: Основи проектування.

Мета: Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.

• Спроєктувати діаграму класів предметної області.

• Вибрати 3 варіанти використання та написати за ними сценарії використання.

• На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.

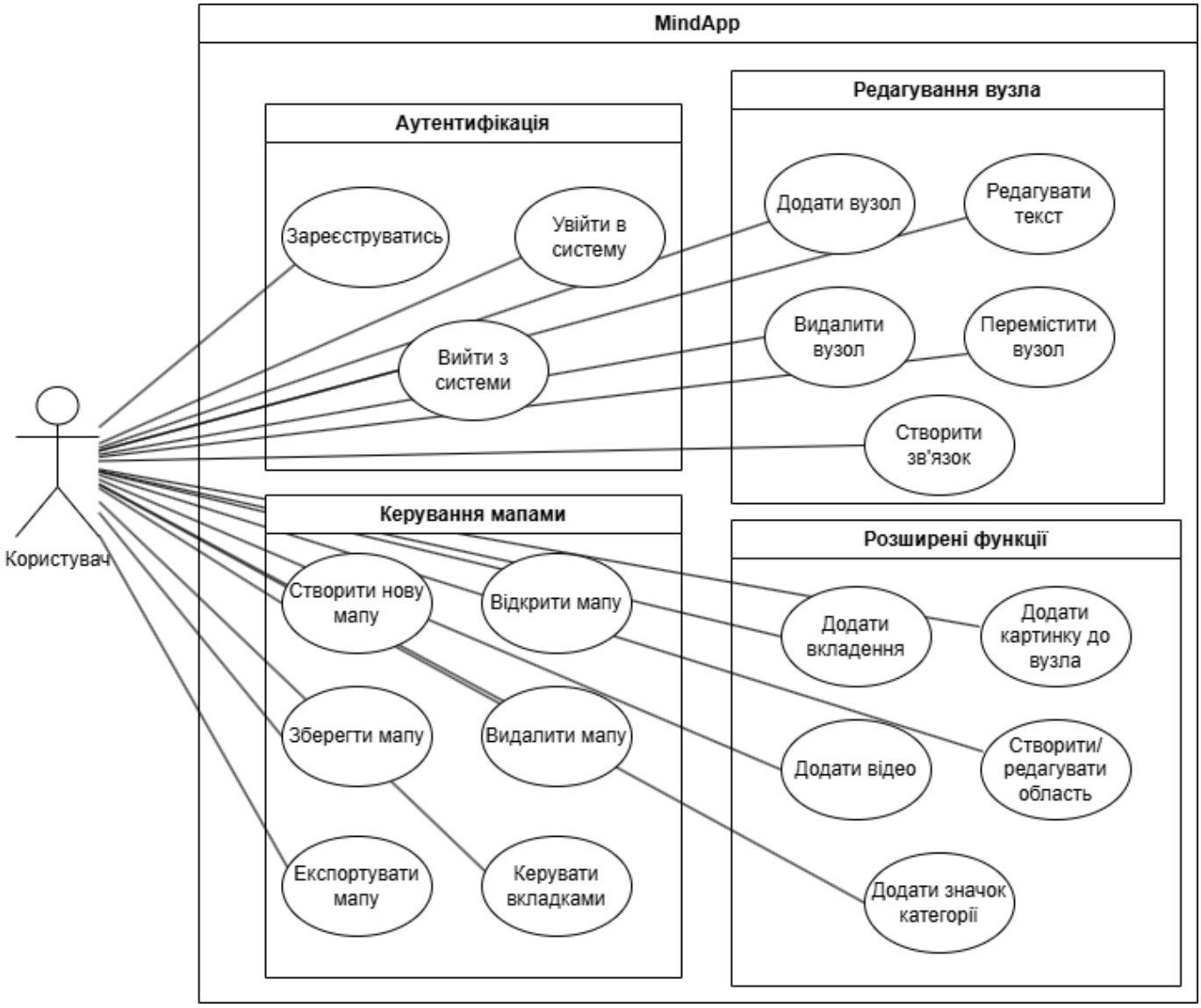
• Нарисувати діаграму класів для реалізованої частини системи.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних

Хід роботи:

Тема проекту: Mind-mapping software (strategy, prototype, abstract factory, bridge, composite, SOA)

Функціональність: Розробка клієнт-серверної системи для візуалізації ідей у вигляді ментальних карт. Створення зручного графічного інтерфейсу для додавання, редагування та видалення ієрархічних вузлів діаграми. Забезпечення хмарної синхронізації: збереження та завантаження карт з віддаленого сервера . Реалізація аутентифікації користувачів для розмежування доступу до особистих даних. Підтримка роботи з декількома картами одночасно через систему вкладок.



Діаграма варіантів використання

Актор - User (Користувач) — взаємодіє з системою та має доступ до таких функцій:

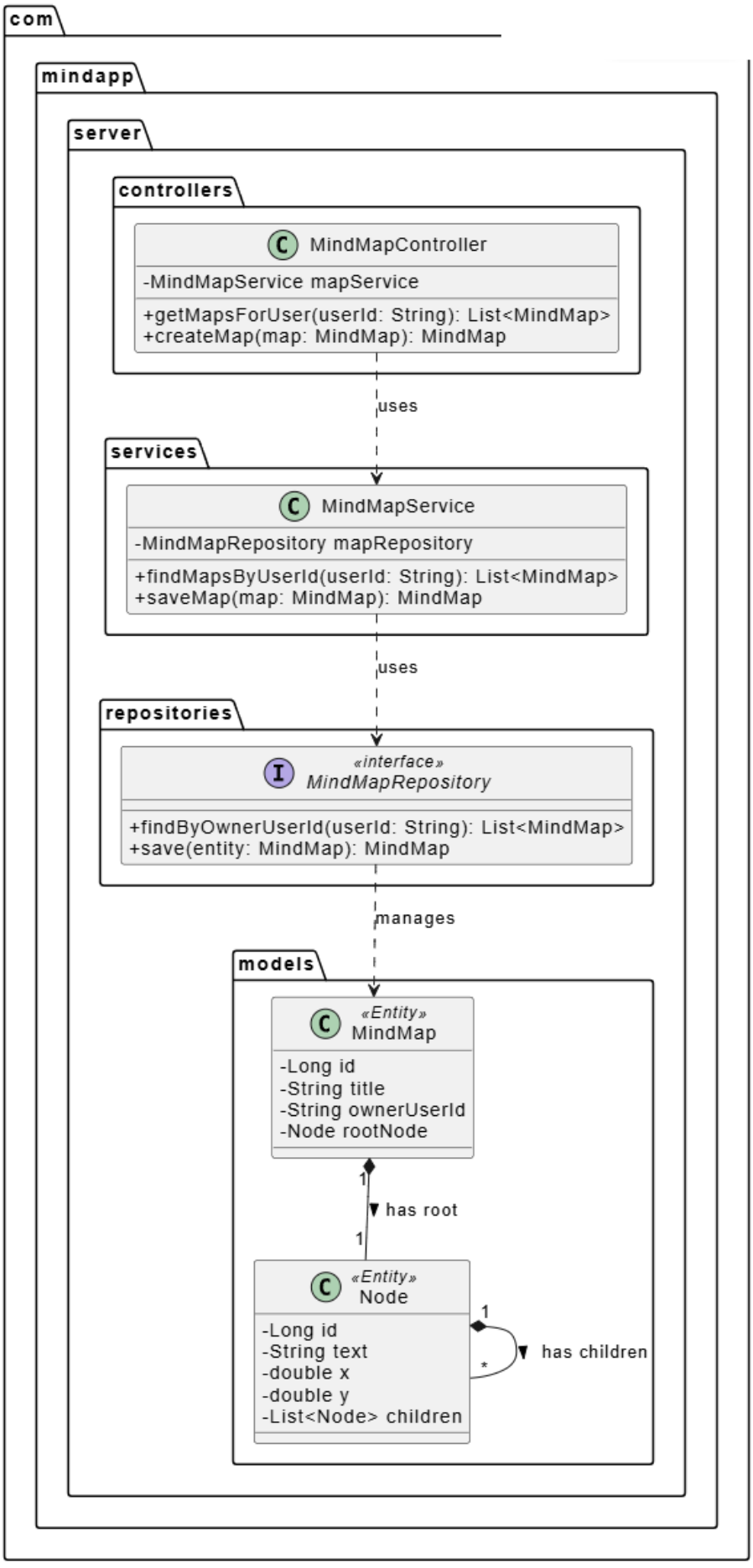
- Зареєструватись / Увійти в систему
- Створити нову мапу
- Відкрити існуючу мапу (з хмари)
- Зберегти мапу
- Додати / Видалити вузол
- Редагувати текст вузла
- Додати вкладення (картинку, відео)
- Вийти з системи

Варіанти використання:

1. **Аутентифікація та безпека**
    - **Реєстрація** — створення нового облікового запису користувача для доступу до системи.
    - **Вхід в систему** — аутентифікація користувача за логіном та паролем для отримання доступу до особистих мап.
    - **Вийти з системи** — завершення сеансу роботи з додатком.
2. **Керування ментальними мапами**
    - **Створення мапи** — ініціалізація нової порожньої ментальної карти з кореневим вузлом.
    - **Відкриття мапи** — завантаження раніше збереженої структури мапи з хмарної бази даних.
    - **Збереження мапи** — синхронізація поточного стану мапи з сервером для надійного зберігання.

- **Видалення мапи** — остаточне вилучення мапи з хмарного сховища.
- **Експортувати мапу** — збереження діаграми у зовнішній файл (наприклад, PNG або PDF).
- **Керувати вкладками** — навігація між декількома відкритими мапами.

3. **Редагування структури та контенту**
   - **Додавання вузла** — створення нового дочірнього елемента на діаграмі.
   - **Редагування тексту** — зміна текстового вмісту обраного вузла.
   - **Видалення вузла** — вилучення вузла разом з усіма його нащадками з діаграми.
   - **Переміщення вузла** — зміна координат розташування вузла на полотні.
   - **Створити зв'язок** — встановлення логічної лінії між двома вузлами.
4. **Розширені функції**
   - **Додавання вкладень** — прикріплення до вузла зовнішніх файлів, зображень або відео.
   - **Додати відео — прикріплення відеофайлу з можливістю перегляду.**
   - **Створити/редагувати область** — виділення групи вузлів візуальною рамкою.
   - **Додати значок категорії** — присвоєння вузлу графічної іконки для категоризації.



Діаграма класів предметної області

Клас **MindMap** моделює ментальну карту в системі "MindApp". Він відповідає за зберігання загальної інформації про мапу та є коренем її структури. **Атрибути:**

- id: Long - унікальний ідентифікатор мапи в базі даних
- title: String - назва мапи
- ownerUserId: String - ідентифікатор користувача-власника
- rootNode: Node - посилання на кореневий вузол мапи

Клас **Node** моделює окремий вузол ментальної карти. Він утворює ієрархічну структуру даних. **Атрибути:**

- id: Long - унікальний ідентифікатор вузла
- text: String - текстовий вміст вузла
- x: double - координата X позиції вузла на полотні
- y: double - координата Y позиції вузла на полотні
- children: List<Node> - список дочірніх вузлів

Клас **MindMapRepository** (інтерфейс) реалізує патерн Repository для управління ментальними картами в базі даних. **Операції:**

- save(entity: MindMap): MindMap - збереження або оновлення мапи
- findByOwnerUserId(userId: String): List<MindMap> - пошук всіх мап конкретного користувача
- (Інші стандартні методи JpaRepository: findById, deleteById тощо)

Клас **MindMapService** містить бізнес-логіку роботи з ментальними картами. **Операції:**

- saveMap(map: MindMap): MindMap - валідація та збереження мапи через репозиторій
- findMapsByUserId(userId: String): List<MindMap> - отримання списку мап для користувача

Клас **MindMapController** є точкою входу для HTTP-запитів (REST API). **Операції:**

- createMap(map: MindMap): MindMap - обробка POST-запиту на створення мапи
- getMapsForUser(userId: String): List<MindMap> - обробка GET-запиту на отримання списку мап

Взаємозв'язки:

- MindMap має один кореневий Node (один-до-одного).
- Node може мати багато дочірніх Node (один-до-багатьох, рекурсивний зв'язок).
- MindMapController використовує MindMapService.
- MindMapService використовує MindMapRepository.
- MindMapRepository керує сутностями MindMap.

Система побудована за принципами тришарової архітектури (Controller-Service-Repository) та використовує Spring Data JPA для автоматичної реалізації доступу до даних.

Сценарії використання системи

Сценарій 1 - Створення нової ментальної карти

Передумови: Програма "MindApp" запущена. Користувач успішно пройшов аутентифікацію в системі. **Постумови:** Створено нову ментальну карту з автоматично доданим кореневим вузлом. Карта відкрита у новій вкладці та готова для редагування. **Взаємодіючі сторони:** Користувач, Система (Клієнт). **Короткий опис:** Користувач ініціює створення нової порожньої ментальної карти для початку роботи над новим проєктом. **Основний потік подій:**

1. Користувач обирає опцію "Створити нову мапу" в головному меню програми.

2. Система (Клієнт) створює в оперативній пам'яті новий об'єкт MindMap з назвою за замовчуванням (наприклад, "Нова мапа").
3. Система автоматично генерує та додає до карти один кореневий вузол (Node) по центру полотна.
4. Система відкриває нову вкладку в інтерфейсі та відображає створену карту.
5. Система встановлює фокус на кореневому вузлі, дозволяючи користувачу одразу почати редагування тексту.
6. Система позначає карту статусом "не збережено".

Винятки:

- **Виняток №1:** Помилка ініціалізації графічного середовища (JavaFX) - система виводить повідомлення про внутрішню помилку та пропонує перезапустити додаток.

Примітки: Нова карта існує лише в пам'яті клієнтського додатка до моменту першого успішного збереження на сервері.

Сценарій 2 - Відкриття існуючої ментальної карти

Передумови: Користувач аутентифікований у системі. У хмарному сховищі (базі даних на сервері) існують раніше збережені цим користувачем карти. **Постумови:** Обрана ментальна карта успішно завантажена з сервера, десеріалізована та відображена у новій вкладці клієнтського додатка. **Взаємодіючі сторони:** Користувач, Система (Клієнт), Система (Сервер), База Даних. **Короткий опис:** Користувач завантажує з хмарного сховища раніше створену ментальну карту для перегляду або подальшого редагування. **Основний потік подій:**

1. Користувач натискає кнопку "Відкрити мапу" в меню програми.
2. Система (Клієнт) надсилає запит на Сервер для отримання списку всіх доступних карт поточного користувача.

3. Сервер звертається до Бази Даних, отримує перелік карт (ID та назви) і повертає його Клієнту.
4. Система (Клієнт) відображає діалогове вікно зі списком отриманих карт.
5. Користувач обирає потрібну карту зі списку і підтверджує вибір.
6. Система (Клієнт) надсилає запит на Сервер для завантаження повної структури обраної карти.
7. Сервер завантажує дані карти з БД, серіалізує їх у формат JSON і надсилає Клієнту.
8. Система (Клієнт) десеріалізує отримані дані, відновлюючи об'єктну модель карти та всіх її вузлів.
9. Система відкриває нову вкладку і візуалізує карту на полотні.

Винятки:

- **Виняток №1:** Відсутнє з'єднання з сервером - система повідомляє про неможливість завантаження списку карт і пропонує перевірити підключення до Інтернету.
- **Виняток №2:** Помилка десеріалізації даних - якщо отримані з сервера дані пошкоджені, система виводить повідомлення про помилку читання файлу.

Примітки: Завантаження великих карт з мультимедійними вкладеннями може зайняти певний час, протягом якого система має відображати індикатор прогресу.

Сценарій 3 - Збереження ментальної карти

Передумови: Користувач аутентифікований. У клієнтському додатку відкрита ментальна карта, до якої були внесені зміни (статус "не збережено"). **Постумови:** Поточний стан ментальної карти успішно синхронізовано з хмарною базою даних. Статус карти в додатку змінено на "збережено". **Взаємодіючі сторони:** Користувач, Система (Клієнт), Система (Сервер), База Даних. **Короткий опис:** Користувач

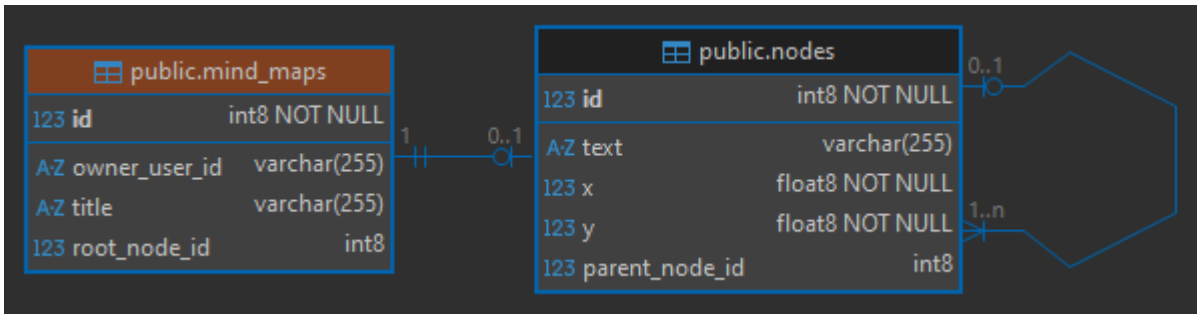
ініціює процес збереження внесених змін, синхронізуючи локальну версію карти з її копією на віддаленому сервері. **Основний потік подій:**

1. Користувач натискає кнопку "Зберегти" на панелі інструментів (або використовує гарячі клавіші Ctrl+S).
2. Система (Клієнт) перевіряє, чи карта вже має присвоєний унікальний ідентифікатор (чи зберігалася вона раніше).
3. Якщо це перше збереження (ID відсутній), система запитує у користувача назву для нової карти.
4. Система (Клієнт) серіалізує поточний стан об'єкта карти (включаючи всі вузли, їх позиції, текст та посилання на вкладення) у формат JSON.
5. Клієнт надсилає HTTP-запит (POST для нової карти або PUT для існуючої) на Сервер, передаючи JSON-дані у тілі запиту.
6. Сервер отримує запит, перевіряє права доступу користувача.
7. Сервер зберігає (або оновлює) дані карти у Базі Даних PostgreSQL.
8. Сервер надсилає Клієнту підтвердження успішного виконання операції (HTTP статус 200 OK).
9. Система (Клієнт) оновлює статус карти на "збережено" та виводить повідомлення користувачу.

Винятки:

- **Виняток №1:** Втрата зв'язку з сервером під час збереження - система повідомляє про помилку синхронізації та залишає статус карти "не збережено", пропонуючи повторити спробу пізніше.
- **Виняток №2:** Конфлікт версій (якщо карта була змінена з іншого пристрою) - сервер може повернути помилку конфлікту, і система запропонує користувачу варіанти вирішення (перезаписати або зберегти як копію).

Примітки: Автоматичне збереження може бути налаштоване для виконання цього сценарію через певні проміжки часу без прямої участі користувача.



Структура бази даних

Код реалізації:

MindMapController.java

```java
package com.mindapp.server.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.mindapp.server.models.MindMap;
import com.mindapp.server.services.MindMapService;

@RestController
@RequestMapping("/api/maps")
public class MindMapController {

    @Autowired
    private MindMapService mindMapService;



    @GetMapping
    public List<MindMap> getMapsForUser(@RequestParam String userId) {
        return mindMapService.findMapsByUserId(userId);
    }


    @PostMapping
    public MindMap createMap(@RequestBody MindMap map) {
        return mindMapService.saveMap(map);
    }
}
```

MindMap.java

```java

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;

@Entity //
@Table(name = "mind_maps") //
public class MindMap {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) //
    private Long id;

    private String title; //

    private String ownerUserId; //



    @OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "root_node_id", referencedColumnName = "id")
    private Node rootNode;



    public MindMap() {}


    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    public String getOwnerUserId() { return ownerUserId; }
    public void setOwnerUserId(String ownerUserId) { this.ownerUserId = ownerUserId; }
    public Node getRootNode() { return rootNode; }
    public void setRootNode(Node rootNode) { this.rootNode = rootNode; }
```

Node.java

src > main > java > com > mindapp > server > models > Node.java > Java > Node

```java
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;

@Entity
@Table(name = "nodes") //
public class Node {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String text;
    private double x; //
    private double y; //


    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER)
    @JoinColumn(name = "parent_node_id") //
    private List<Node> children = new ArrayList<>();



    public Node() {}


    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getText() { return text; }
    public void setText(String text) { this.text = text; }
    public double getX() { return x; }
    public void setX(double x) { this.x = x; }
    public double getY() { return y; }
    public void setY(double y) { this.y = y; }
    public List<Node> getChildren() { return children; }
    public void setChildren(List<Node> children) { this.children = children; }
}
```

MindMapRepository.java

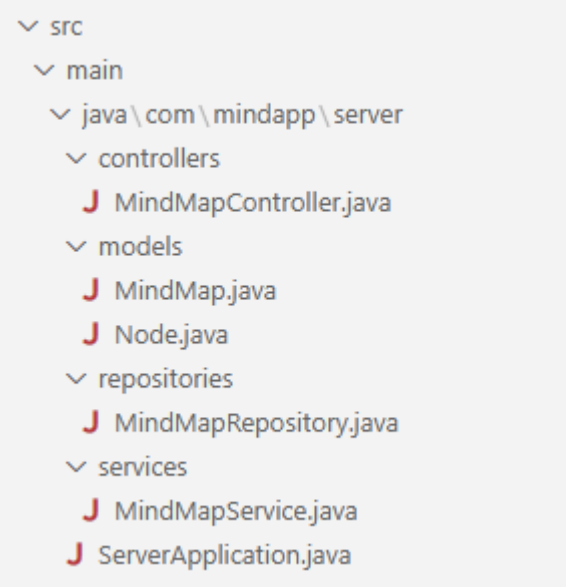
```java
package com.mindapp.server.repositories;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import com.mindapp.server.models.MindMap;



public interface MindMapRepository extends JpaRepository<MindMap, Long> {


    List<MindMap> findByOwnerUserId(String userId);
}
```

MindMapService.java

```java
package com.mindapp.server.services;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.mindapp.server.models.MindMap;
import com.mindapp.server.repositories.MindMapRepository;

@Service //
public class MindMapService {

    @Autowired //
    private MindMapRepository mindMapRepository;


    public List<MindMap> findMapsByUserId(String userId) {
        return mindMapRepository.findByOwnerUserId(userId);
    }


    public MindMap saveMap(MindMap map) {
        return mindMapRepository.save(map);
    }
}

```

ServerApplication.java

```java
package com.mindapp.server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ServerApplication {

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(ServerApplication.class, args);
    }

}

```



Реалізовані класи

Висновки:

У межах лабораторної роботи я спроєктував клієнт-серверну систему "MindApp", яка дозволяє користувачам працювати з ментальними картами.

Були створені діаграми варіантів використання та класів, розроблені детальні сценарії для трьох основних функцій системи: створення нової мапи, відкриття існуючої мапи з сервера та збереження мапи у хмарному сховищі.

Також була спроєктована структура реляційної бази даних PostgreSQL з двома основними таблицями (mind_maps, nodes) та реалізовано серверну частину додатка на Spring Boot. Для ефективної абстакції взаємодії з даними було використано архітектурний шаблон Repository.

Система передбачає базовий функціонал для візуалізації ієрархічних даних та забезпечує їх надійну синхронізацію з віддаленим сервером.

Відповіді на контрольні питання:

1. Що таке UML? UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується для графічного представлення, документування та проектування програмних систем. Вона допомагає описати структуру, поведінку та взаємодію компонентів системи.

2. Що таке діаграма класів UML?

Діаграма класів UML — це статична діаграма, яка показує класи системи, їх атрибути, методи та зв’язки між класами. Вона використовується для моделювання структури об’єктно-орієнтованих програм.

3. Які діаграми UML називають канонічними?

Канонічні (основні) діаграми UML — це ті, що найчастіше використовуються для опису системи:

• Діаграма класів

• Діаграма варіантів використання (Use Case)

• Діаграма послідовності (Sequence)

• Діаграма станів (State)

• Діаграма активностей (Activity)

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use Case Diagram) показує взаємодію користувачів (акторів) з системою через варіанти використання. Вона відображає, хто і як використовує систему, без деталізації внутрішньої реалізації.

5. Що таке варіант використання?

Варіант використання (Use Case) — це послідовність дій, які система виконує

для досягнення певної мети користувача. Наприклад, “Реєстрація користувача” або “Надіслати повідомлення”.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання можуть бути:

• Association (асоціація) — зв’язок між актором і варіантом використання.

• Include (включення) — один варіант використання завжди виконує інший.

• Extend (розширення) — додатковий варіант, який може виконуватися у певних умовах.

• Generalization (успадкування) — актор або варіант використання наслідує інший.

7. Що таке сценарій?

Сценарій — це конкретна реалізація варіанту використання, тобто послідовність кроків, які відбуваються під час виконання дії користувачем та системою.

8. Що таке діаграма класів?

Діаграма класів — це графічне представлення класів, їх атрибутів і методів, а також зв’язків між класами. Вона описує структуру системи.

9. Які зв’язки між класами ви знаєте?

Основні типи зв’язків:

• Асоціація (Association) — загальний зв’язок між класами.

• Агрегація (Aggregation) — «має»; клас складається з інших, але частини можуть існувати окремо.

• Композиція (Composition) — сильніша агрегація; частини не можуть існувати 22 без цілого.

• Успадкування (Generalization) — один клас наслідує властивості іншого.

• Залежність (Dependency) — один клас використовує інший тимчасово.

10. Чим відрізняється композиція від агрегації?

• Агрегація: частини можуть існувати без цілого.

• Композиція: частини не можуть існувати без цілого; знищення цілого знищує частини.

11. Чим відрізняється агрегація від композиції на діаграмах класів?

• Агрегація позначається порожнім ромбом на стороні цілого.

• Композиція позначається заповненим ромбом на стороні цілого.

12. Що являють собою нормальні форми баз даних?

Нормальні форми — це правила організації таблиць БД для уникнення надлишковості та аномалій при оновленні даних. Основні:

• 1НФ — всі атрибути атомарні.

• 2НФ — всі неключові атрибути залежать від усього первинного ключа.

• 3НФ — немає транзитивних залежностей між неключовими атрибутами.

13. Що таке фізична і логічна модель БД?

• Логічна модель — структура даних з таблицями, полями та зв’язками, незалежно від СУБД.

• Фізична модель — конкретна реалізація БД у СУБД, включає типи даних, індекси, обмеження, фізичне зберігання.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Кожна таблиця БД часто відповідає класу у програмі, а рядки таблиці —

об’єктам класу. Поля таблиці відображаються як атрибути класу, а зв’язки між таблицями як зв’язки між класами.