МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНА СИСТЕМА ОБЛІКУ ТА ОЦІНЮВАННЯ УЧНІВ ЗАГАЛЬНООСВІТНІХ ШКІЛ

Здобувач освіти гр. ІН-81 Маргарита РОЗГОН

Науковий керівник,
кандидат техн. наук, старший викладач Олег БЕРЕСТ

Завідувач кафедри
доктор технічних наук, професор. Анатолій ДОВБИШ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____
Зав. кафедри Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи бакалавра

Студентки 4-го курсу, групи ІН-81 спеціальності 122 - Комп'ютерні науки, дистанційної форми навчання Розгон М. О.

Тема: Інформаційна система обліку та оцінювання учнів загальноосвітніх шкіл

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) інформаційний огляд; 2) вибір методів розв'язання поставленої задачі; 3) моделювання та реалізація додатку

Дата видачі завдання "_____"_____2022 р.
Керівник випускної роботи _____ Берест О.Б.
Завдання прийняв до виконання _____ Розгон М. О.

РЕФЕРАТ

Записка: 72 стор., 43 рис., 3 табл., 2 додатка, 21 джерело.

Об’єкт дослідження — процес обліку та оцінювання учнів загальноосвітніх шкіл.

Мета роботи — розробка інформаційної системи для обліку та оцінювання учнів загальноосвітніх шкіл.

Методи дослідження — аналітичні методи обробки інформації.

Результати — розроблено інформаційну систему, у вигляді веб-додатку «Gradebook», який являє собою електронний шкільний журнал. Додаток було розроблено за допомогою середовища розробки Intellij IDEA.

ЕЛЕКТРОННИЙ ЖУРНАЛ, ОЦІНЮВАННЯ УЧНІВ, ІНФОРМАЦІЙНА СИСТЕМА, ПРОГРАМУВАННЯ, ВЕБ-ДОДАТОК, JAVA, SPRING FRAMEWORK, БАЗИ ДАНИХ, POSTGRESQL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

ER-діаграма (entity-relationship diagram) – це візуальне представлення бази даних, яке показує структуру сутностей та як вони пов’язані між собою.

Діаграма варіантів використання (Use case diagram) – це графічне зображення можливої взаємодії користувача з системою. Діаграма варіантів використання показує різні варіанти використання та різні типи користувачів, яких має система.

Діаграма послідовності (sequence diagram) – UML-діаграма, на якій для деякого набору об'єктів на єдиній часовій осі показаний життєвий цикл об'єкта та взаємодія акторів інформаційної системи в рамках варіанту використання.

JSP (Java Server Pages) — технологія, що дозволяє веб-розробникам динамічно генерувати HTML, XML та інші веб-сторінки. Технологія дозволяє вставляти Java-код, в статичний вміст сторінки.

HTML (HyperText Markup Language) – це мова тегів, засобами якої здійснюється розмітка веб-сторінок для мережі Інтернет.

XML (eXtensible Markup Language) – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

Фреймворк (Framework) – інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проєкту та написання коду.

Oracle WebLogic Server — єдина платформа, що розширюється, для розробки, розгортання та запуску корпоративних додатків, таких як Java, у локальному та хмарному середовищі.

Java EE (Java Enterprise Edition) – це платформа для створення корпоративних додатків на мові Java. Перед усім це сфера веб-додатків та веб-сервісів.

Анотація – спеціальна форма метаданих, в мові Java, що може бути додана до вихідного коду.

HTTP (HyperText Transfer Protocol) – протокол передачі даних, що використовується в комп'ютерних мережах.

JSON (JavaScript Object Notation) — це текстовий формат обміну даними між комп'ютерами.

БД – база даних.

СКБД – система керування базами даних.

JS – JavaScript.

ВСТУП

В сучасному світі технології розвиваються дуже швидко, та продовжують збільшувати темп розвитку. Спроби заощадити час приводять до автоматизації всього навколо. Все частіше техніка заміняє людину, виконуючи замість неї певну роботу та полегшуючи життя. Спроби заощадити час приводять до автоматизації всього навколо. Від найелементарніших калькуляторів до суперкомп'ютерів – всі ці прилади були розроблені, щоб виконувати операції замість людини. Постійний технічний прогрес розкриває перед людиною все більше можливостей. Люди все частіше надають перевагу інформації в електронному вигляді. На підприємствах та державних установах журнали та документація все частіше зберігаються в електронному вигляді ніж в друкованому на папері. Дана робота присвячена розробленню веб-додатку «Інформаційна система обліку та оцінювання учнів загальноосвітніх шкіл». Або простішими словами це шкільний журнал в електронному вигляді. Використання таких додатків та інших сучасних технологій роблять можливим дистанційне навчання. Завдяки дистанційному навчанню багато учнів та студентів мають можливість продовжувати здобуття освіти, не зважаючи на ряд перешкод, з якими зіткнулось суспільство останнім часом. Використання сучасних технологій забезпечує зв'язок між учнями та вчителями під час дистанційної форми навчання.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

Додатки для автоматизації навчальних процесів можна умовно розділити за їх призначенням на ступні типи:

- Електронні щоденники – допомагають учням відслідковувати розклад занять, домашні завдання та оцінки;
- Електронні Журнали – є аналогами шкільних журналів;
- Додатки для тестування учнів – допомагають зекономити час на перевірку тестів;
- Додатки для автоматичного створення розкладу.

Обраний на розробку, додаток відноситься до типу електронних журналів.

1.1 Аналіз аналогів

Платформа e-journal.iea.gov.ua надає безкоштовний доступ, для шкіл, до електронного журналу та інших ресурсів. Демонстраційної версії ресурсу немає. Для його використання потрібно надіслати заявку на підключення школи. На сайті є інструкції для адміністратора, вчителя, класного керівника, учнів та батьків[1].

Сайт nz.ua (рисунок 1.1), як і попередній ресурс, надає безкоштовний доступ для шкіл до функціоналу, демонстраційної версії немає, є інструкції для користувачів різного типу[2].

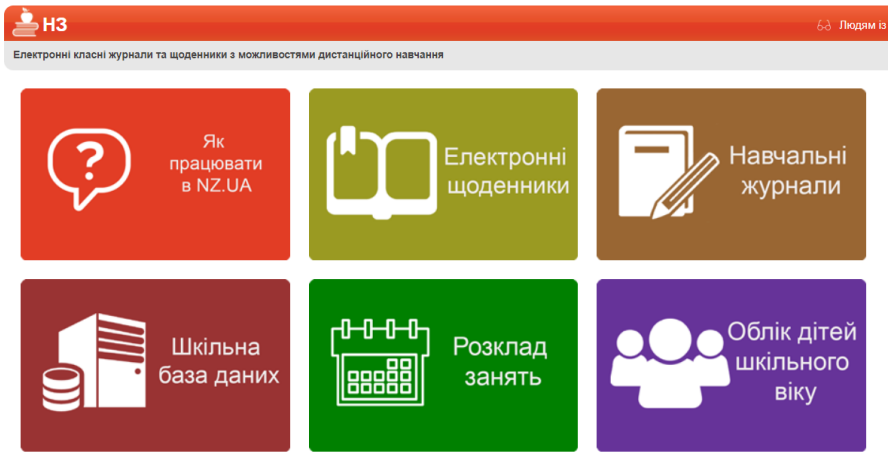


Рисунок 1.1 – головна сторінка сайту nz.ua

Сайт e-schools.info надає доступ для неавторизованих користувачів до демонстраційної версії свого ресурсу. Він містить онлайн журнал (рисунок 1.2) та щоденник, а також інші сервіси для автоматизації навчального процесу. До журналу, на даному сайті, мають доступ лише вчителі. Даний ресурс має широкий функціонал, але інтерфейс досить складний та заплутаний[3].

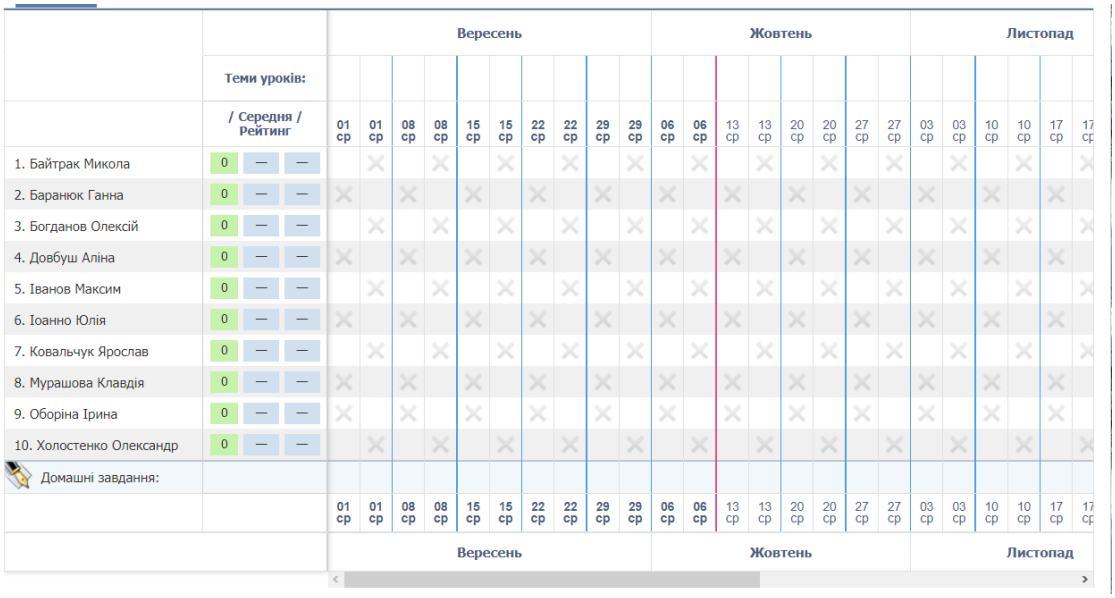


Рисунок 1.2 – демонстраційна версія електронного журналу з сайту e-schools.info

Сайт ukrschools.com.ua також надає демонстраційну версію. Розділ «електронний журнал» (рисунок 1.3) даного ресурсу відображає лише середні досягнення учнів[4].

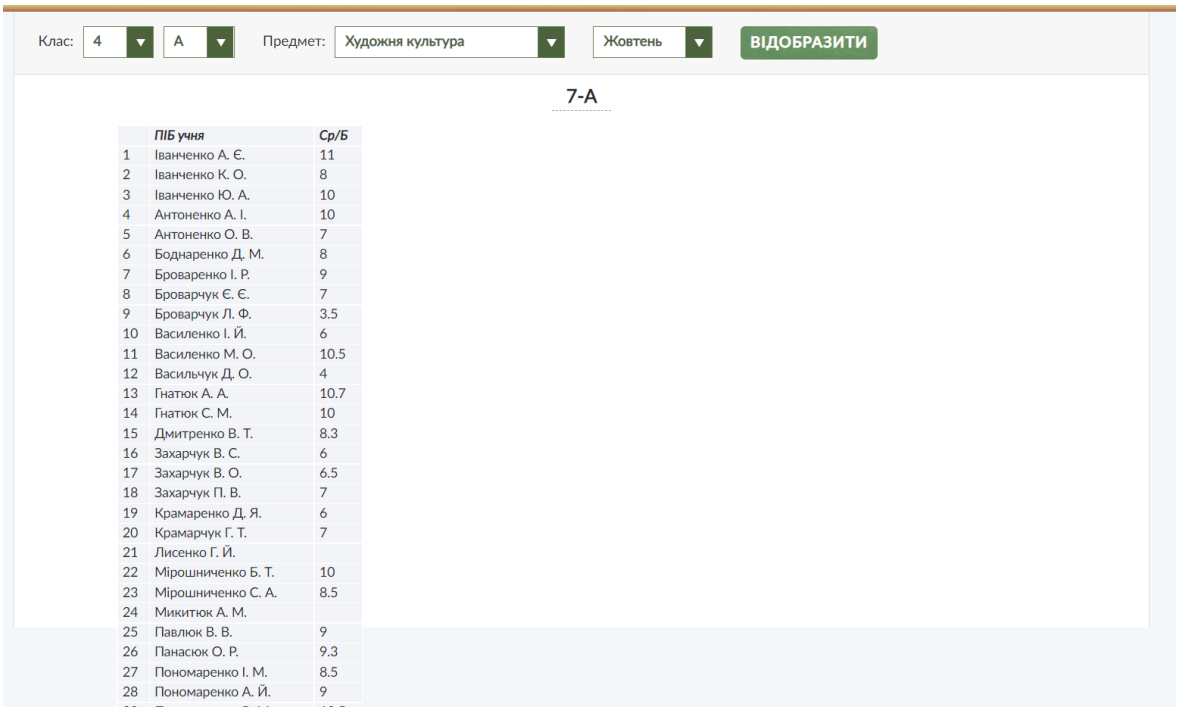


Рисунок 1.3 – демонстраційна версія електронного журналу з сайту ukrschools.com.ua

Дана сторінка (рисунок 1.3) має дефекти відображення, а також виникають деякі труднощі при спробі зробити вибір з випадаючих списків.

1.2 Постановка завдання

Завдання: розробити веб-додаток «Інформаційна система обліку та оцінювання учнів загальноосвітніх шкіл». Робоча назва додатку: «Gradebook».

Головні цілі додатку:

- Вчителям: надати можливість ведення шкільного журналу в електронному вигляді;

- Учням: надати можливість відслідковувати свої навчальні досягнення в журналі.

Для виконання завдання, необхідно обрати технічні засоби, такі як, архітектура, мова програмування, система керування базами даних та інше.

Далі потрібно розробити модель додатку. Для цього потрібно:

- Спроектувати базу даних у вигляді ER-діаграми;
- Виділити ролі користувачів системи. Визначити функціонал, який буде доступний користувачу в залежності від його ролі. Розробити діаграму варіантів використання;
- Для основних варіантів використання, розробити діаграми послідовностей;

Наступний крок – написання коду, згідно з розробленою моделлю.

Далі потрібно наповнити базу даних тестовою інформацією та перевірити коректність роботи додатку. Якщо в роботі додатку, виявлено помилки, то їх необхідно виправити.

2 ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

Додаток «Gradebook» буде мати клієнт-серверну архітектуру, так як це є стандартом для веб-додатків. На рисунку 2.1, наведено схему додатку.

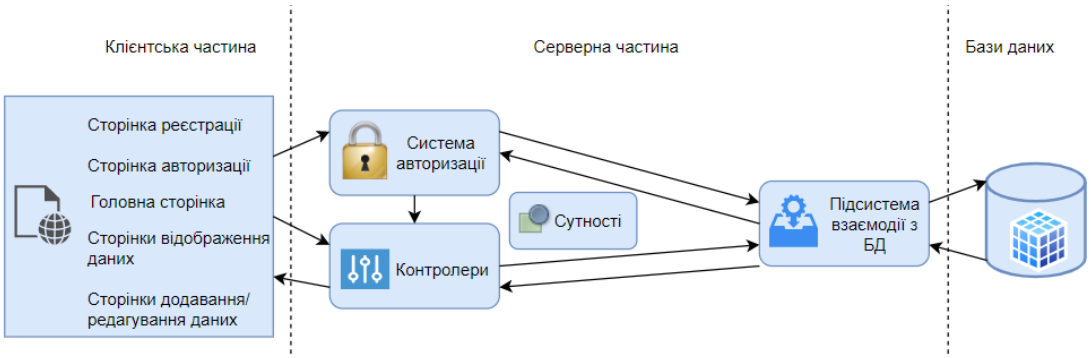


Рисунок 2.1 – Схема додатку

На схемі, клієнтська частина додатку включає перелік основних сторінок. Серверна частина складається з кількох підсистем. Підсистема взаємодії з базами даних, повинна забезпечувати операції читання, вставки, оновлення та видалення даних з бази даних. Система авторизації взаємодіє з підсистемою взаємодії з БД, щоб отримати дані користувача та перевірити коректність пароля, після чого повинно здійснюватися перенаправлення на головну сторінку. Контролери повинні обробляти запити користувачів, та повертати відповідну сторінку. Для цього вони взаємодіють із підсистемою взаємодії з БД. Сутності, в свою чергу, задають типи об'єктів, у вигляді яких, підсистеми обмінюються даними.

Для написання серверної частини додатку обрано мову Java. Зважаючи на ряд переваг, наведених у підрозділі 2.3, було вирішено використати Spring Framework. Авторизація та розділення на ролі буде реалізовано за допомогою Spring Security. Клієнтську частину додатку буде реалізовано у вигляді сторінок JSP, за допомогою мови розмітки HTML, з використанням мови JavaScript. Для розгортання додатку було обрано веб-сервер Oracle WebLogic,

адже він має повну підтримку Java EE додатків. Для зберігання даних було обрано СКБД PostgreSQL, так як вона є безкоштовною, має зручний інтерфейс для керування базами даних, підтримує взаємодію з обраним веб-сервером. Огляд обраних технологій наведено в наступних підрозділах.

2.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура (рисунок 2.2) передбачає розділення додатку на дві частини: клієнта та сервера. На рівні програмного забезпечення, клієнтом та сервером вважаються частини додатку. На рівні апаратного забезпечення клієнтом та сервером вважаються комп’ютери, на яких розміщено відповідні частини додатку. Клієнт – частина додатку, яка призначена для взаємодії з користувачем та надсилання запиту на сервер. У більшості веб-додатків, клієнтом є інтернет браузер. Основний функціонал додатку виконується на сервері. Сервер очікує від клієнтських програм запити і надає їм свої ресурси у вигляді даних або у вигляді сервісних функцій. Найчастіше клієнтська та серверна частини додатку знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми — і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Рисунок 2.2 – Клієнт-серверна архітектура

Етапи обробки запиту:

- Користувач робить запит, за допомогою клієнтської частини додатку (або браузеру) розташованої на клієнтському комп’ютері;
- Запит надсилається на сервер;
- Сервер обробляє запит клієнта;
- Якщо запит клієнта передбачає взаємодію з базою даних (читання, вставку, редагування або видалення даних), тоді сервер виконує відповідний запит до бази даних;
- База даних повертає на сервер дані, відповідно до отриманого запиту;
- Сервер відправляє відповідь клієнту, зазвичай у вигляді html сторінки;
- Клієнтська частина додатку або браузер показує вміст отриманої сторінки, у зрозумілому для користувача вигляді.

2.2 Мова програмування Java

Java – сильно типізована, об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems», викуплена в 2009 році компанією «Oracle» [5].

Сильно типізована означає, що всі змінні повинні мати тип. Крім того не можна присвоювати змінній одного типу, значення іншого типу без явного перетворення типів. В таблиці 2.1 наведено перелік простих типів даних.

Таблиця 2.1 – Прості типи даних мови Java [6]

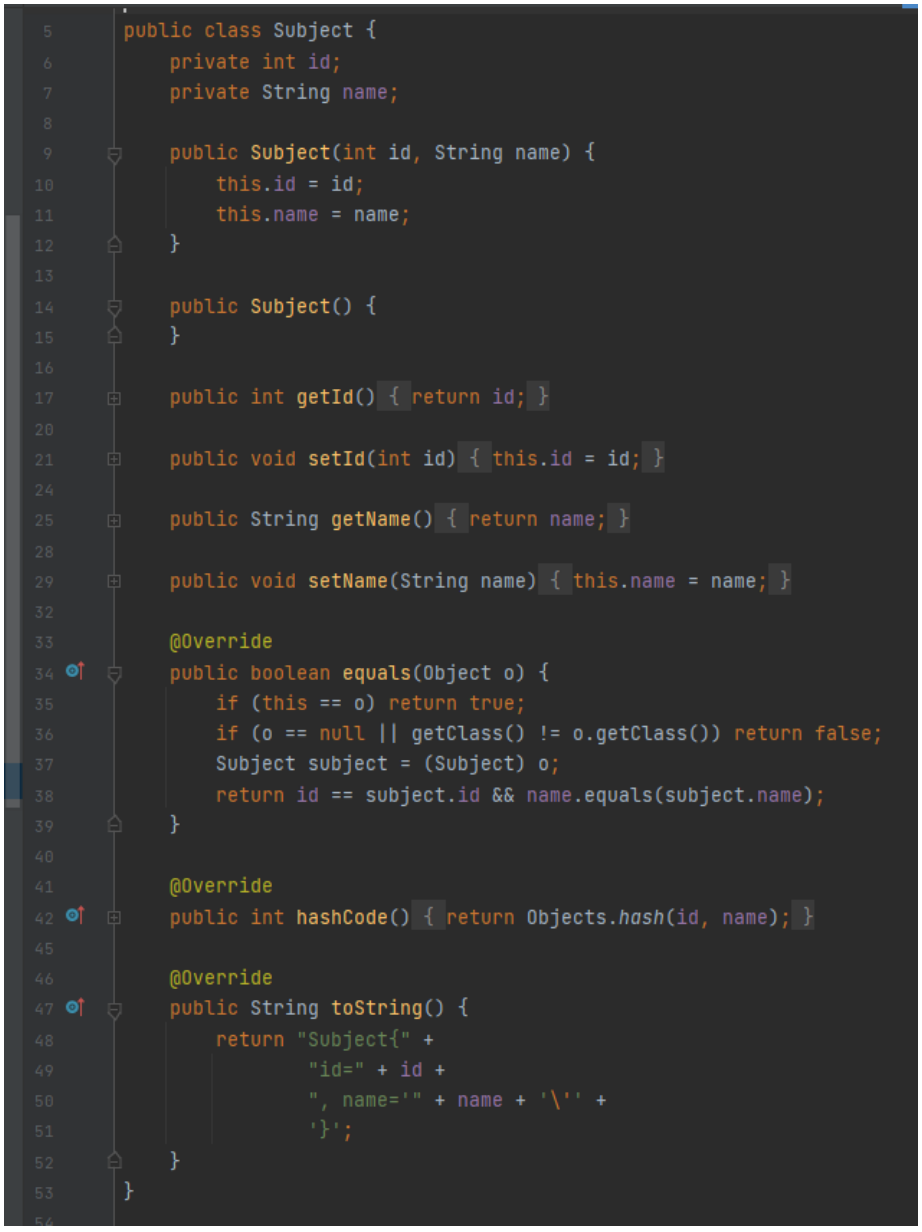
Тип	Довжина (в байтах)	Діапазон або набір значень
boolean	1 в масивах, 4 в змінних	true, false
byte	1	−128..127
char	2	$0..2^{16}-1$, або 0..65535
short	2	$-2^{15}..2^{15}-1$, або −32768..32767
int	4	$-2^{31}..2^{31}-1$, або −2147483648..2147483647
long	8	$-2^{63}..2^{63}-1$, або приблизно $-9.2\cdot10^{18}..9.2\cdot10^{18}$
float	4	$-(2-2^{-23})\cdot2^{127}..(2-2^{-23})\cdot2^{127}$, або приблизно $-3.4\cdot10^{38}..3.4\cdot10^{38}$
double	8	$-(2-2^{-52})\cdot2^{1023}..(2-2^{-52})\cdot2^{1023}$, або приблизно $-1.8\cdot10^{308}..1.8\cdot10^{308}$

Програма, на мові Java, згідно з парадигмою об’єктно-орієнтованого програмування, розглядається як множина об’єктів, що взаємодіють між собою. Об’єктом є екземпляр класу. Клас задає тип об’єкта. Об’єкт може містити поля, які описують його стан, та функції, які описують його поведінку. Поля об’єкта можуть бути як простого типу, так і містити інші об’єкти[7].

Основу об’єктно-орієнтованого програмування складають чотири основні концепції:

- Інкапсуляція;
- Успадкування (наслідування);
- Поліморфізм;
- Абстракція.

За інкапсуляцією, доступ до стану об’єкта не повинен відбуватися на пряму, а тільки через методи. Тобто поля об’єкта повинні бути приватними [8]. Приклад інкапсуляції наведено на рисунку 2.3.



```java
public class Subject {
    private int id;
    private String name;

    public Subject(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public Subject() {
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Subject subject = (Subject) o;
        return id == subject.id && name.equals(subject.name);
    }

    @Override
    public int hashCode() { return Objects.hash(id, name); }

    @Override
    public String toString() {
        return "Subject{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

Рисунок 2.3 – Клас Subject як приклад інкапсуляції

Успадкування - опис нового класу на основі вже існуючого (батьківського), при цьому властивості і функціональність батьківського класу запозичуються новим класом [8]. На рисунку 2.4 наведено приклад успадкування.

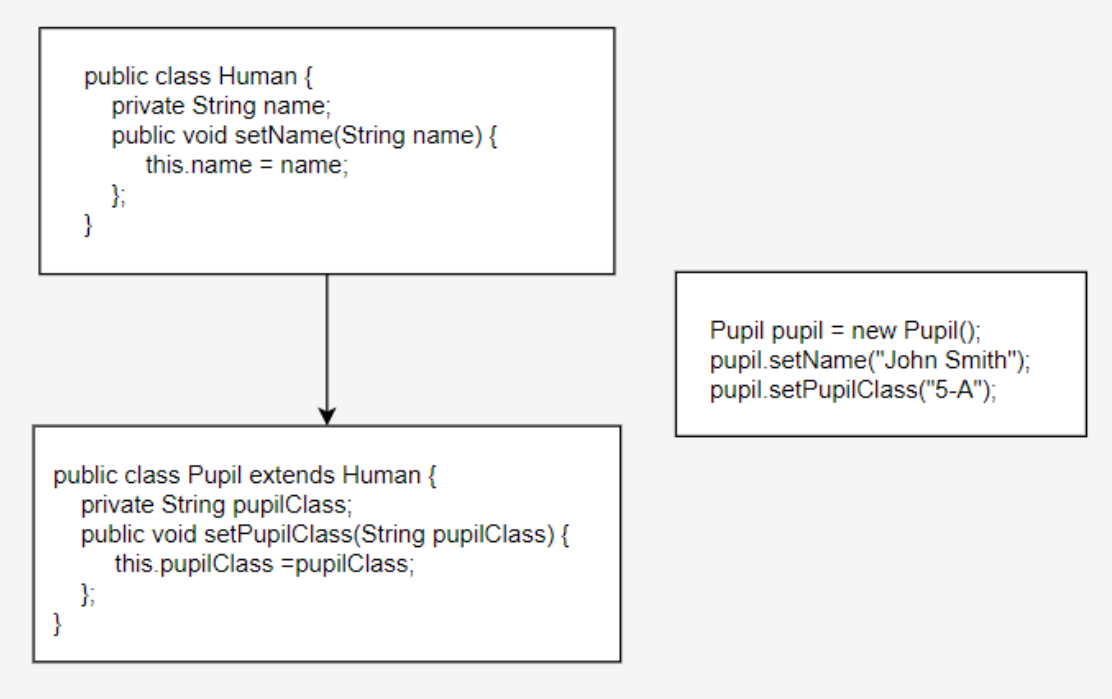


Рисунок 2.4 – Приклад успадкування

Поліморфізм - можливість об'єктів з однаковою специфікацією мати різну реалізацію. При цьому різні об'єкти можуть бути використані однаковим чином [8].

На основі простих типів даних та вже існуючих класів, розробник може створювати свої класи, які будуть задавати новий тип даних. Часто виникає необхідність описати деякий об’єкт із реального світу. Наприклад, в рамках розроблюваного додатку потрібно описати учня школи. В реальному світі, учень може мати безліч характеристик, таких як, ім’я, вік, клас, адреса проживання, колір волосся або очей, зріст і так далі. Але в рамках розробленої інформаційної системи, більшість цих характеристик не є важливими. Тому для опису учня в додатку було обрано ім’я та клас. Використання лише тих характеристик, які достатньо точно описують об’єкт в рамках заданої системи називається абстракцією [9].

2.3 Spring Framework

Spring Framework – це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

Об’єкти, які керуються Spring контейнером називаються бінами (beans). Зазвичай, на мові Java, всі необхідні об’єкти створюються за допомогою оператора new, але в Spring Framework передбачено механізм впровадження залежностей. Всі необхідні біни створюються в контейнері. Контейнер передає управління над біном іншому біну, якому це необхідно. Таким чином, один об’єкт можна перевикористовувати багато разів, замість того щоб створювати новий об’єкт при кожному використанні, що є однією з переваг Spring Framework. Для того, щоб бін було створено в контейнері, його потрібно сконфігурувати. Існує три способи конфігурації бінів:

- Створення Java класу з конфігураціями;
- Конфігурація в xml файлі;
- Анотації [10].

Для конфігурації бінів у класі, використовуються анотації @Configuration та @Bean. Анотація @Configuration використовується на рівні класу. Вона вказує, що даний клас є джерелом визначення бінів. Анотація @Bean використовується на рівні методу, та вказує на те, що об’єкт, який повертає даний метод, є біном [11]. На рисунку 2.5 наведено приклад конфігурації біна в класі, з розробленого додатку.

```java
@EnableWebMvc
@Configuration
@ComponentScan({"org.example"})
public class AppConfig implements WebMvcConfigurer {

    private final ApplicationContext applicationContext;

    @Autowired
    public AppConfig(ApplicationContext applicationContext) { this.applicationContext = applicationContext; }

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        viewResolver.setContentType("text/html;charset=UTF-8");
        return viewResolver;
    }

}
```

Рисунок 2.5 – Приклад конфігурації біна в класі

Одна із переваг Spring Framework полягає в тому, що винесення конфігурації в окремий файл полегшує заміну реалізацій в проекті. На рисунку 2.6 наведено приклад конфігурації бінів за допомогою xml файлу. Для цього використовується теги <beans></beans> та <bean/>. В тезі <bean/> потрібно вказати id біна, та клас, на основі якого його потрібно створити. Також за необхідності можна додати властивості біна, за допомогою тегу <property/>.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">


    <mvc:annotation-driven/>
    <mvc:resources mapping="/images/**" location="/WEB-INF/images/"/>

    <context:component-scan base-package = "org.example" />

    <bean id="lessonDAO" class="org.example.dao.postgres.PostgresLessonDAO"/>
    <bean id="markDAO" class="org.example.dao.postgres.PostgresMarkDAO"/>
    <bean id="pupilClassDAO" class="org.example.dao.postgres.PostgresPupilClassDAO"/>
    <bean id="pupilDAO" class="org.example.dao.postgres.PostgresPupilDAO"/>
    <bean id="roleDAO" class="org.example.dao.postgres.PostgresRoleDAO"/>
    <bean id="schoolYearDAO" class="org.example.dao.postgres.PostgresSchoolYearDAO"/>
    <bean id="semesterDAO" class="org.example.dao.postgres.PostgresSemesterDAO"/>
    <bean id="subjectDAO" class="org.example.dao.postgres.PostgresSubjectDAO"/>
    <bean id="subjectDetailsDAO" class="org.example.dao.postgres.PostgresSubjectDetailsDAO"/>
    <bean id="teacherDAO" class="org.example.dao.postgres.PostgresTeacherDAO"/>
    <bean id="themeDAO" class="org.example.dao.postgres.PostgresThemeDAO"/>
    <bean id="userDAO" class="org.example.dao.postgres.PostgresUserDAO"/>

</beans>
```

Рисунок 2.6 – Приклад конфігурації біна в xml файлі

Для автоматичної конфігурації бінів, використовуються наступні анотації: @Component, @Service, @Repository та @Controller. Ці анотації застосовуються на рівні класу. Анотація @Component є більш загальною. Інші три анотації є її підтипами. Всі, перелічені вище анотації, вказують, що на основі класу, до якого вони застосовуються, потрібно створити бін. Крім того:

- @Service вказує, що даний клас є сервісом, тобто містить бізнес логіку;
- @Repository вказує, що клас забезпечує механізм зберігання, пошуку, оновлення та операції видалення об'єктів;
- @Controller вказує, що даний клас є контролером [11].

На рисунку 2.7 вказано приклад використання анотації @Controller у розробленому додатку.

```
@Controller
public class MarkController {
    private final MarkDAO dao;
    private final LessonDAO lessonDAO;
    private final PupilDAO pupilDAO;
    private final SubjectDAO subjectDAO;
    private final SubjectDetailsDAO subjectDetailsDAO;
    private final MarkService markService;
    private static final int marksPerPage = 12;
    private static final Logger LOGGER = Logger.getLogger(MarkController.class.getName());

    public MarkController(MarkDAO dao,
                          LessonDAO lessonDAO,
                          PupilDAO pupilDAO,
                          SubjectDAO subjectDAO,
                          SubjectDetailsDAO subjectDetailsDAO, MarkService markService) {...}
```

Рисунок 2.7 – Приклад використання анотації @Controller

Модулі Spring Framework:

- Inversion of Control-контейнер (IoC-контейнер, контейнер інверсії управління);
- Фреймворк аспектно-орієнтованого програмування;

- Фреймворк доступу до даних;
- Фреймворк керування транзакціями;
- Фреймворк MVC;
- Фреймворк віддаленого доступу;
- Фреймворк аутентифікації і авторизації;
- Фреймворк віддаленого управління;
- Фреймворк роботи з повідомленнями;
- Тестування [12].

Ще одна із переваг Spring Framework, полягає у тому, що його модулі можуть працювати незалежно один від одного. Завдяки цьому, розробник може використовувати лише ті модулі, які йому необхідні, а всі інші не використовувати.

2.3.1 Spring Security

Spring Security це Java/JavaEE фреймворк, що надає механізми побудови систем аутентифікації та авторизації, а також інші можливості забезпечення безпеки для промислових додатків, створених за допомогою Spring Framework [13].

Для того, щоб застосувати Spring Security в своєму додатку, потрібно реалізувати наступні інтерфейси та класи:

- GrantedAuthority;
- UserDetails;
- UserDetailsService;
- AuthenticationProvider;
- WebSecurityConfigurerAdapter;
- WebApplicationInitializer;
- AbstractSecurityWebApplicationInitializer.

За допомогою Spring Security можна обмежити доступ до частини контенту сторінки в залежності від ролі користувача. Для цього використовується тег authorize. Приклад його використання наведено на рисунку 2.8. В результаті комірка з id буде відображатися лише для користувача з роллю ADMIN.

```
<tr class="card">
    <sec:authorize access="hasAuthority('ADMIN')">
        <td><%=lesson.getId()%></td>
    </sec:authorize>
    <td>
        <div class="inline"><i class='material-icons'>today</i></div>
        <div class="inline"><%=strDate%></div>
    </td>
    <td>
        <div class="inline"><i class='material-icons'>event_note</i></div>
        <div class="inline"><%=lesson.getTopic()%></div>
    </td>
```

Рисунок 2.8 – Приклад використання тегу authorize

Крім того, можна обмежити доступ до методів. Для цього існує ряд анотацій таких, як @Secured, @RoleAllowed, @PreAuthorize, @PostAuthorize і так далі [14]. На рисунку 2.9 наведено приклад використання анотації @Secured.

```
64      @RequestMapping(value = "/users")
65      @Secured("ADMIN")
66      public ModelAndView viewAllUsers(@RequestParam("page") int page) {...}
85
```

Рисунок 2.9 – Приклад використання анотації @Secured

В результаті, якщо користувач, який не має ролі ADMIN, спробує перейти за шляхом /users, він отримає помилку з HTTP-статусом 403.

2.4 Мова програмування JavaScript

JavaScript (JS) — динамічна, прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на боці клієнта взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки [15].

2.4.1 AJAX

AJAX (Asynchronous JavaScript And XML) – підхід в веб-розробці, що полягає в створенні асинхронних запитів до сервера. В результаті таких запитів відбувається не повне оновлення сторінки, а лише частини її контенту.

Застосування AJAX:

- На мові JavaScript створюється функція, яка реагує на певні події такі, як натискання на кнопку чи посилання або на введення тексту в текстове поле або інші;
- Користувач робить дії, які викликають виконання цієї функції;
- Функція формує та надсилає запит до серверу та чекає від серверу відповіді;
- Сервер обробляє запит та повертає відповідь у одному з наступних форматів: XML, JSON або текст;
- Функція отримує відповідь та змінює контент сторінки згідно з отриманими даними, без повного перезавантаження сторінки [16].

Приклад використання: користувач намагається зареєструватися, для цього вводить своє ім’я. Ім’я користувача повинно бути унікальним, тому потрібно перевірити чи є в базі даних користувач з таким ім’ям. Якщо є, то потрібно повідомити про це користувача до того як він відправив дані на сервер. Тому перевірка повинна відбуватися при введенні даних у поле. Для

цього у полі було описано властивість onkeyup (рисунок 2.10), яка викликає функцію для перевірки, при введенні даних у поле.

```
<p id="placeToShow" class="warning"></p>
<div class="row">
    <div class="col-25">
        <label>Ім<span>&#39;</span>я</label>
    </div>
    <div class="col-75">
        <form:input type="text" path="username" onkeyup="checkUsername(0)" placeholder="Ім'я користувача"/>
        <br/><br/>
    </div>
</div>
```

Рисунок 2.10 – Поле, в якому викликається JS функція

На рисунку 2.11 вказано JavaScript функцію, в якій надсилається запит до серверу.

```
var request = new XMLHttpRequest();
function checkUsername(userID) {
    var val = document.getElementById("username").value;
    var url = "/Gradebook/checkUsername?val=" + val + "&id=" + userID;
    try {
        request.onreadystatechange = function () {
            if (request.readyState === 4) {
                document.getElementById("placeToShow").innerHTML = request.responseText;
                if (request.responseText !== "") {
                    document.getElementById("save").disabled = true;
                } else {
                    document.getElementById("save").disabled = false;
                }
            }
        }
        request.open("GET", url, true);
        request.send();
    } catch (e) {
        alert("Unable to connect to server");
    }
}
```

Рисунок 2.11 – JS функція, яка здійснює асинхронний запит до серверу

На рисунку зображено функцію на сервері, яка обробляє асинхронний запит. Якщо в базі даних вже є задане ім’я користувача, то функція повертає відповідне повідомлення, в іншому випадку – порожній рядок.

```java
@RequestMapping(value = "/checkUsername", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
@Secured({"ADMIN", "TEACHER", "PUPIL"})
@ResponseBody
public String checkUsername(@RequestParam("val") String name, @RequestParam("id") int id){
    User user = dao.getUserByUsername(name);
    String response = "";
    if(user != null) {
        if (user.getId() != id) {
            response = "Ім'я користувача зайняте!";
        }
    }
    return response;
}
```

Рисунок 2.12 – Функція на сервері, яка обробляє асинхронний запит

Після отримання відповіді від серверу, функція (рисунок 2.11) вставляє отриманий текст в елемент з id placeToShow, та вимикає кнопку для збереження.

На рисунку 2.13 зображено форму для реєстрації.

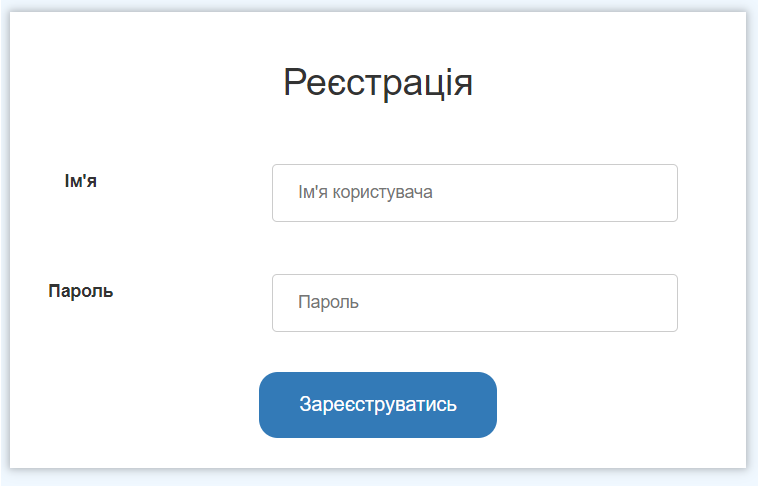


Рисунок 2.13 – Форма реєстрації

При введенні існуючого імені користувача, форма має вигляд зображений на рисунку 2.14.

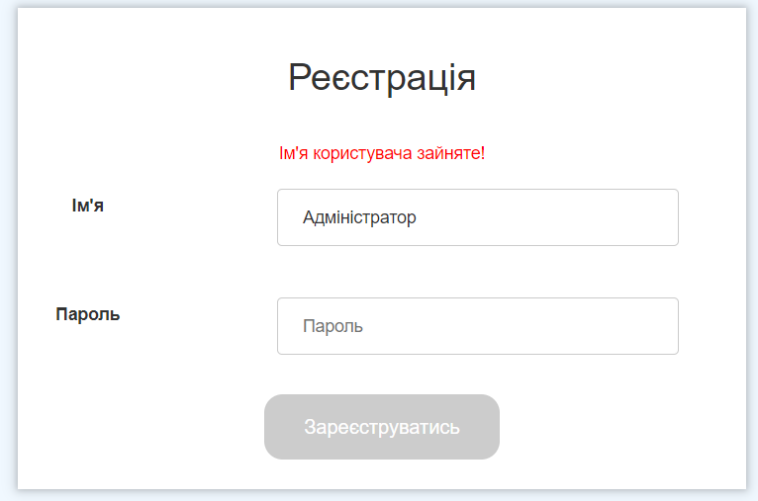


Рисунок 2.14 – Форма реєстрації при введенні імені користувача, яке уже існує в базі даних

В результаті, користувач отримує повідомлення, що введене ім’я користувача зайняте. А також кнопка «Зареєструватись» - disabled, щоб користувач не зміг відправити дані на сервер.

2.5 База даних

База даних – структурований набір взаємопов'язаних даних які зберігаються та доступні в електронному вигляді.

Система керування базою даних – програмна система, яка дозволяє створити базу даних, керувати нею та маніпулювати даними. Функціональність, яку забезпечує СКБД, може значно відрізнятися. Основною функціональністю є зберігання, пошук та оновлення даних. За моделлю організації даних розрізняють такі бази даних:

- Ієрархічна;
- Мережева;
- Реляційна;
- Об'єктно-орієнтована;
- Об’єктно-реляційна;

- Інші [17].

Список найпопулярніших СКБД:

- Oracle;
- MySQL;
- Microsoft SQL Server;
- PostgreSQL;
- MongoDB.

Всі СКБД із цього списку, окрім MongoDB, є реляційними [18].

Для розроблюваного додатку було обрано СКБД PostgreSQL.

PostgreSQL – безкоштовна об'єктно-реляційна система керування базами даних [19]. Деякі джерела відносять PostgreSQL до реляційних БД [20], деякі до об’єктно-реляційних. Щоб використовувати PostgreSQL, потрібно завантажити інсталятор з офіційного сайту та запустити його. Інсталятор встановить ряд необхідних утиліт серед яких – pgAdmin (рисунок 2.15) – утиліта для управління базами даних PostgreSQL через користувацький інтерфейс.

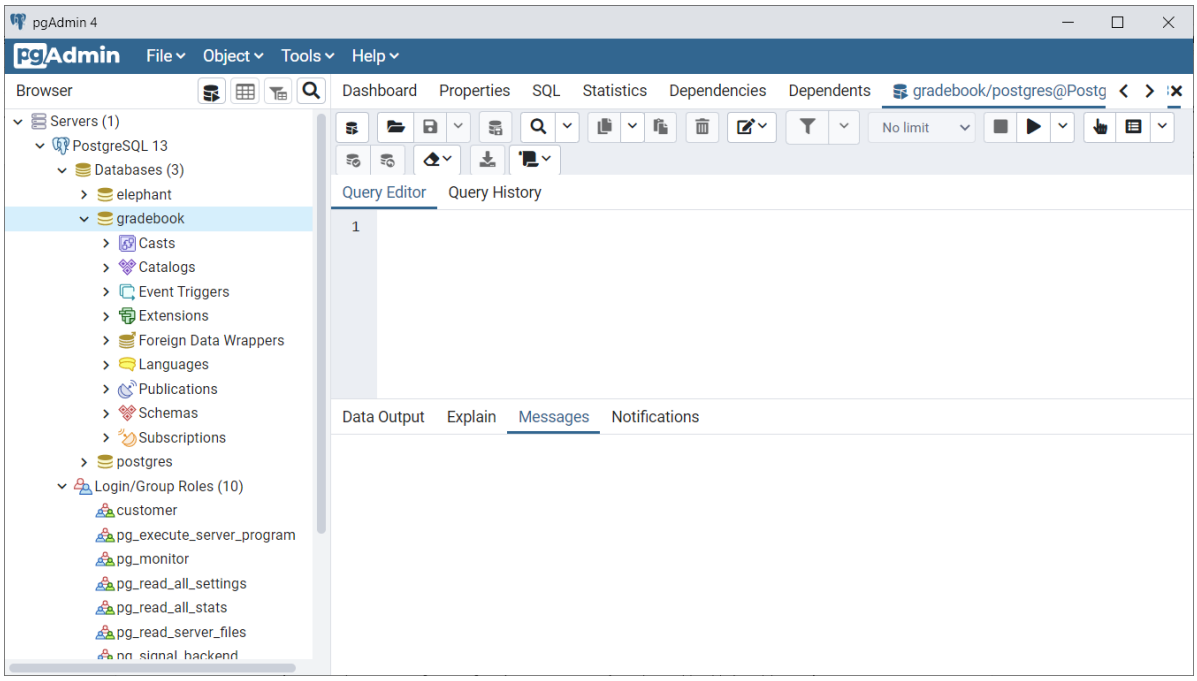


Рисунок 2.15 – утиліта pgAdmin

Для взаємодії користувача або додатку з реляційною базою даних, використовується SQL. SQL (Structured query language) – декларативна мова програмування, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Кожна СКБД реалізує власний діалект SQL [21].

В реляційних базах даних, дані зберігаються у вигляді взаємопов’язаних таблиць (відношень). Таблиці складаються зі стовпців (атрибутів) та рядків (записів, кортежів). Кожен рядок таблиці має однакову структуру. Стовпці задають структуру рядків. Кожен стовпець має певний тип даних.

Зазвичай таблиці містять первинний ключ, який допомагає однозначно ідентифікувати рядок. Первинний ключ може бути простим, тобто складатися з одного атрибуту, або складеним, тобто складатися з кількох атрибутів. Первинний ключ повинен бути унікальним, та обов’язково мати значення. В якості первинного ключа часто використовують автоматично згенерований ідентифікатор. В PostgreSQL для цього існує окремий тип даних – serial.

Зв’язки між таблицями реалізуються за допомогою зовнішніх ключів. Зовнішнім ключем називається такий атрибут таблиці, який посилається на первинний ключ іншої таблиці.

3 МОДЕЛЮВАННЯ ТА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Проектування бази даних

Для бази даних додатку «Gradebook» було виділено наступні сутності:

- Користувач;
- Роль;
- Навчальний рік;
- Семестр;
- Учень;
- Клас;
- Предмет;
- Тема;
- Урок;
- Оцінка;
- Відсутність.

Користувач може мати декілька ролей. Роль може належати багатьом користувачам. Отже, між сутностями користувач та роль, зв’язок багато-до-багатьох. Щоб реалізувати такий зв’язок, потрібно додатково виділити сутність User-role (рисунок 3.1).

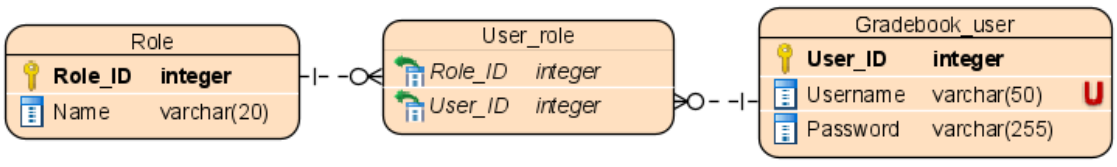


Рисунок 3.1 – Реалізація зв’язку між сутностями користувач та роль

Користувач може бути вчителем. Користувач може бути учнем. Вчителі та учні повинні бути користувачами. Отже, між сутностями вчитель та учень та сутністю користувач, зв’язки один-до-одного (рисунок 3.2).

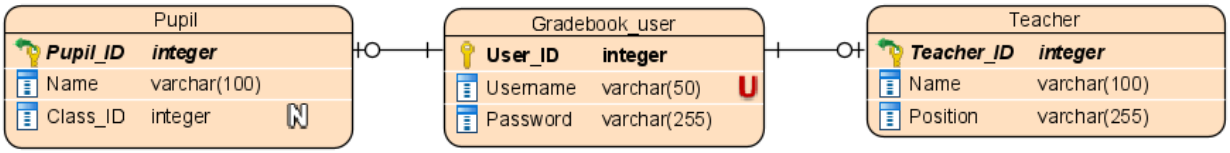


Рисунок 3.2 – Зв’язок між сутностями користувач, учень, вчитель

Кожен учень повинен навчатися в одному класі. Кожен клас повинен містити багатьох учнів. Такий зв’язок реалізувати не можливо, адже в такому випадку, в базу даних необхідно відразу (одним запитом до БД) додати клас та хоча б одного учня. Тому, зв’язок було пом’якшено з обох боків, щоб можна було додавати класи окремо, учнів окремо. Зв’язок, отриманий в результаті, зображено на рисунку 3.3.

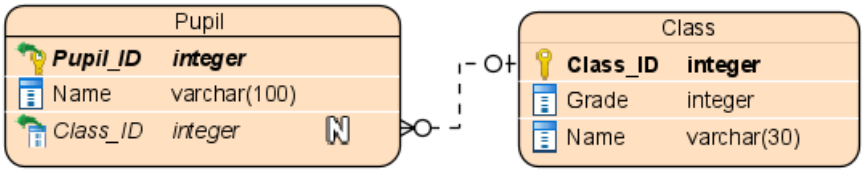


Рисунок 3.3 – Зв’язок між сутностями учень та клас

Навчальний рік повинен містити два семестра. Семестр повинен належати одному навчальному року. Зв’язок (рисунок 3.4) було пом’якшено з одного боку.

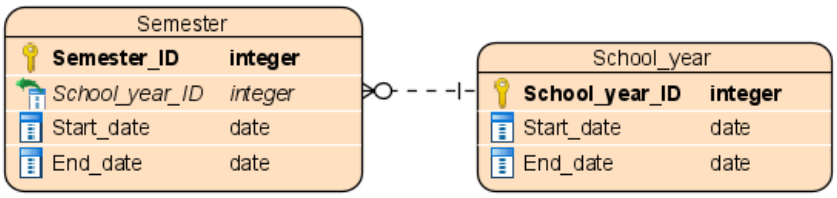


Рисунок 3.4 – Зв’язок між сутностями навчальний рік та семестр

Предмет може викладатися у багатьох класах, в різних семестрах. Клас може вивчати багато предметів за семестр. Предмет повинен викладатися

вчителем. Вчитель може викладати різні предмети у різних класах у різних семестрах. Щоб реалізувати ці вимоги, було виділено ще одну сутність – деталі предмету (рисунок 3.5). Деякі зв’язки було пом’якшено.

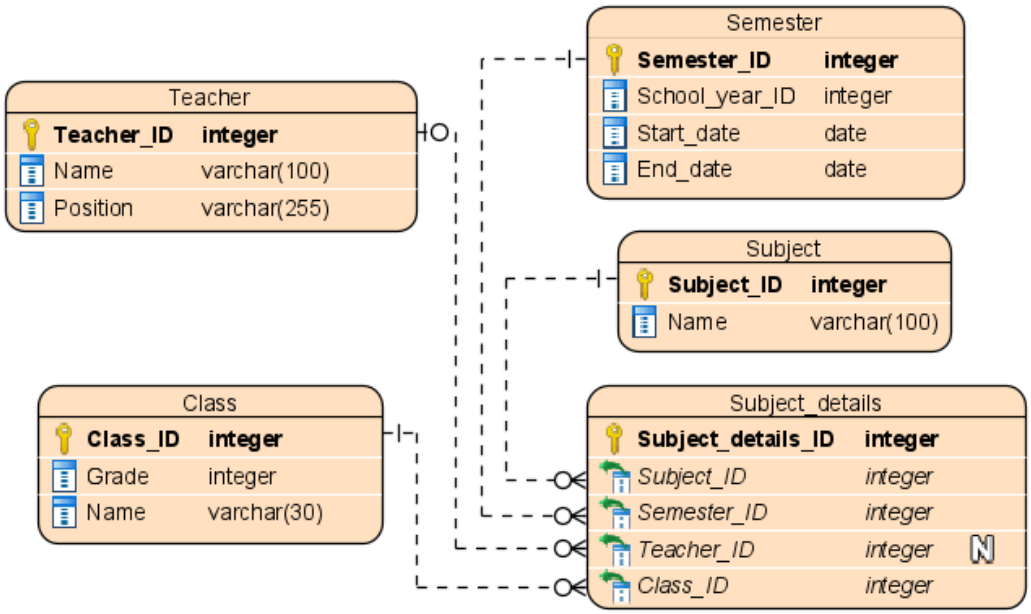


Рисунок 3.5 – Зв’язки між сутностями семестр, вчитель, клас, предмет та деталі предмету

Кожен предмет (деталі предмету) може містити багато тем. Кожна тема повинна належати одному предмету (деталям предмету). Кожна тема може містити багато уроків. Кожен урок повинен належати одній темі. Дані зв’язки зображено на рисунку 3.6.

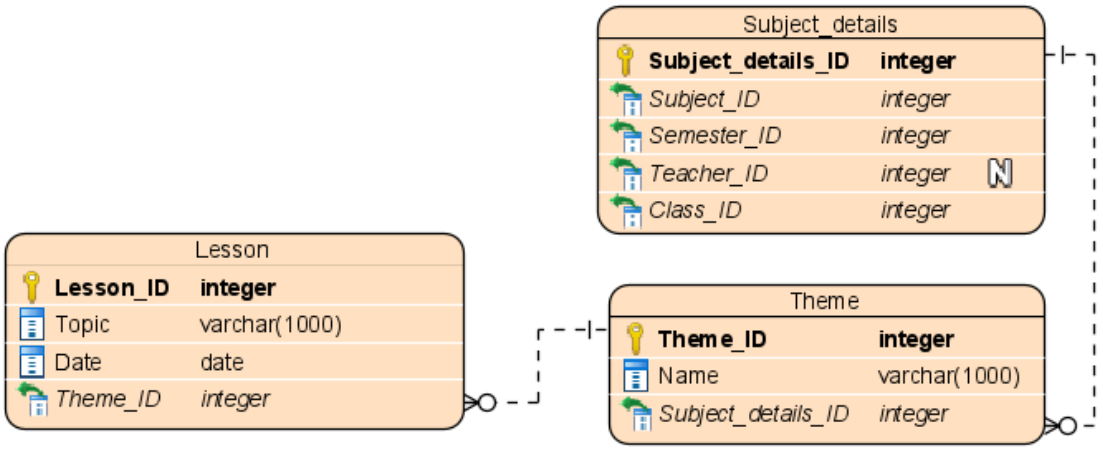


Рисунок 3.6 – Зв’язки між сутностями деталі предмету, тема та урок

Учень може бути відсутнім на багатьох уроках. На уроці, багато учнів можуть бути відсутніми. Учень може отримати оцінки за багато уроків, але за один урок тільки одна оцінка. За урок багато учнів можуть отримати оцінки. В результаті отримано зв’язки, зображені на рисунку 3.7.

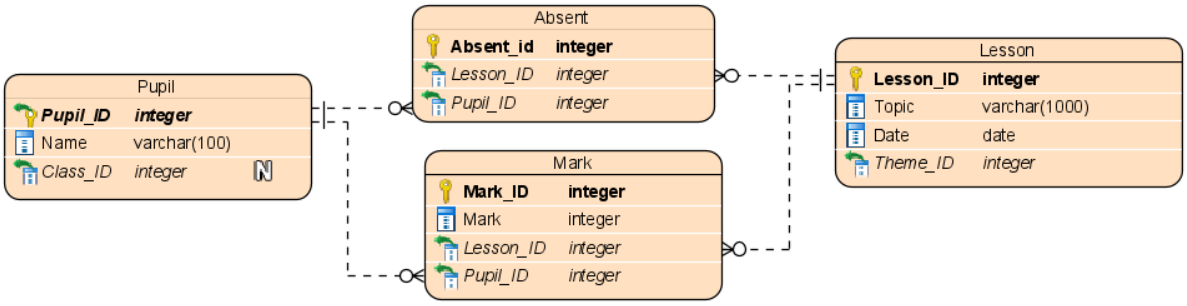


Рисунок 3.7 – Зв’язок між сутностями учень, урок, оцінка, відсутній

Повну ER-діаграму наведено в додатку А.

Список всіх виділених сутностей наведено у таблиці 3.1.

Таблиця 3.1 – Список сутностей

Назва	Опис
Role	Список ролей користувачів додатку.
Gradebook_user	Список користувачів додатку.
User_role	З’єднує користувачів та їх ролі. Реалізує відношення багато до багатьох між сутностями Gradebook_user та Role
School_year	Список навчальних років.
Semester	Список семестрів.
Teacher	Список вчителів школи.
Pupil	Список учнів школи.
Class	Список класів школи.
Subject	Список предметів, які викладаються у школі.
Subject_details	З’єднує семестр, предмет, вчителя та клас. За даними цієї таблиці надається доступ вчителям та учням до предмету.
Theme	Список тем з певного предмету.
Lesson	Список уроків певної теми.
Mark	Список оцінок.
Absent	З’єднує учня та урок на якому він був відсутнім.

Структуру таблиць бази даних описано у вигляді таблиці 3.2.

Таблиця 3.2 – Структура таблиць бази даних додатку

Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
Role	Role_ID	Ідентифікатор ролі	Integer	PK	Not null, unique
	Name	Назва ролі	varchar(20)		Not null
Gradebook_user	User_ID	Ідентифікатор користувача	Serial	PK	Not null, unique
	Username	Ім’я користувача	varchar(30)		Not null, unique
	Password	Пароль користувача	varchar(200)		Not null
	Photo	Шлях до фото користувача у файловій системі	varchar(100)		
User_role	User_ID	Ідентифікатор користувача	Integer	FK	Not null
	Role_ID	Ідентифікатор ролі користувача	Integer	FK	Not null
School_year	School_year_ID	Ідентифікатор навчального року	Serial	PK	Not null, unique
	Start_date	Дата початку навчального року	Date		Not null
	End_date	Дата закінчення навчального року	Date		Not null
Semester	Semester_ID	Ідентифікатор семестру	Serial	PK	Not null, unique
	School_year_ID	Ідентифікатор навчального року	Integer	FK	Not null
	Start_date	Дата початку семестру	Date		Not null
	End_date	Дата закінчення семестру	Date		Not null

Продовження таблиці 3.2

Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
Teacher	Teacher_ID	Ідентифікатор вчителя	Integer	PK, FK	Not null, unique
	Name	Повне ім’я вчителя	varchar(100)		Not null
	Position	Посада вчителя	varchar(100)		Not null
Pupil	Pupil_ID	Ідентифікатор учня	Integer	PK, FK	Not null, unique
	Name	Ім’я учня	varchar(100)		Not null
	Class_ID	Ідентифікатор класу учня	Integer	FK	
Class	Class_ID	Ідентифікатор класу	Serial	PK	Not null, unique
	Grade	Рік навчання	Integer		Not null
	Name	Назва класу	varchar(30)		Not null
Subject	Subject_ID	Ідентифікатор предмету	Serial	PK	Not null, unique
	Name	Назва предмету	varchar(100)		Not null
Subject_details	Subject_details_ID	Ідентифікатор деталей предмету	Serial	PK	Not null, unique
	Semester_ID	Ідентифікатор семестру	Integer	FK	Not null
	Class_ID	Ідентифікатор класу	Integer	FK	Not null
	Teacher_ID	Ідентифікатор вчителя	Integer	FK	
	Subject_ID	Ідентифікатор предмету	Integer	FK	Not null

Продовження таблиці 3.2

Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
Theme	Theme_ID	Ідентифікатор теми	Serial	PK	Not null, unique
	Subject_details_ID	Ідентифікатор деталей предмету	Integer	FK	Not null
	Name	Назва теми	varchar(1000)		Not null
Lesson	Lesson_ID	Ідентифікатор уроку	Serial	PK	Not null, unique
	Theme_ID	Ідентифікатор теми	Integer	FK	Not null
	Date	Дата уроку	Date		Not null
	Topic	Тема уроку	varchar(1000)		Not null
Mark	Mark_ID	Ідентифікатор оцінки	Serial	PK	Not null, unique
	Pupil_ID	Ідентифікатор учня	Integer	FK	Not null
	Lesson_ID	Ідентифікатор уроку	Integer	FK	Not null
	Mark	Оцінка	Integer		Not null
Absent	Absent_ID	Ідентифікатор відсутнього	Serial	PK	Not null, unique
	Pupil_ID	Ідентифікатор учня	Integer	FK	Not null
	Lesson_ID	Ідентифікатор уроку	Integer	FK	Not null

3.2 Функціональні можливості користувачів системи

В додатку «Gradebook» виділено три ролі користувачів:

- учень;
- вчитель;
- адміністратор.

На рисунку 3.8 наведено діаграму варіантів використання.

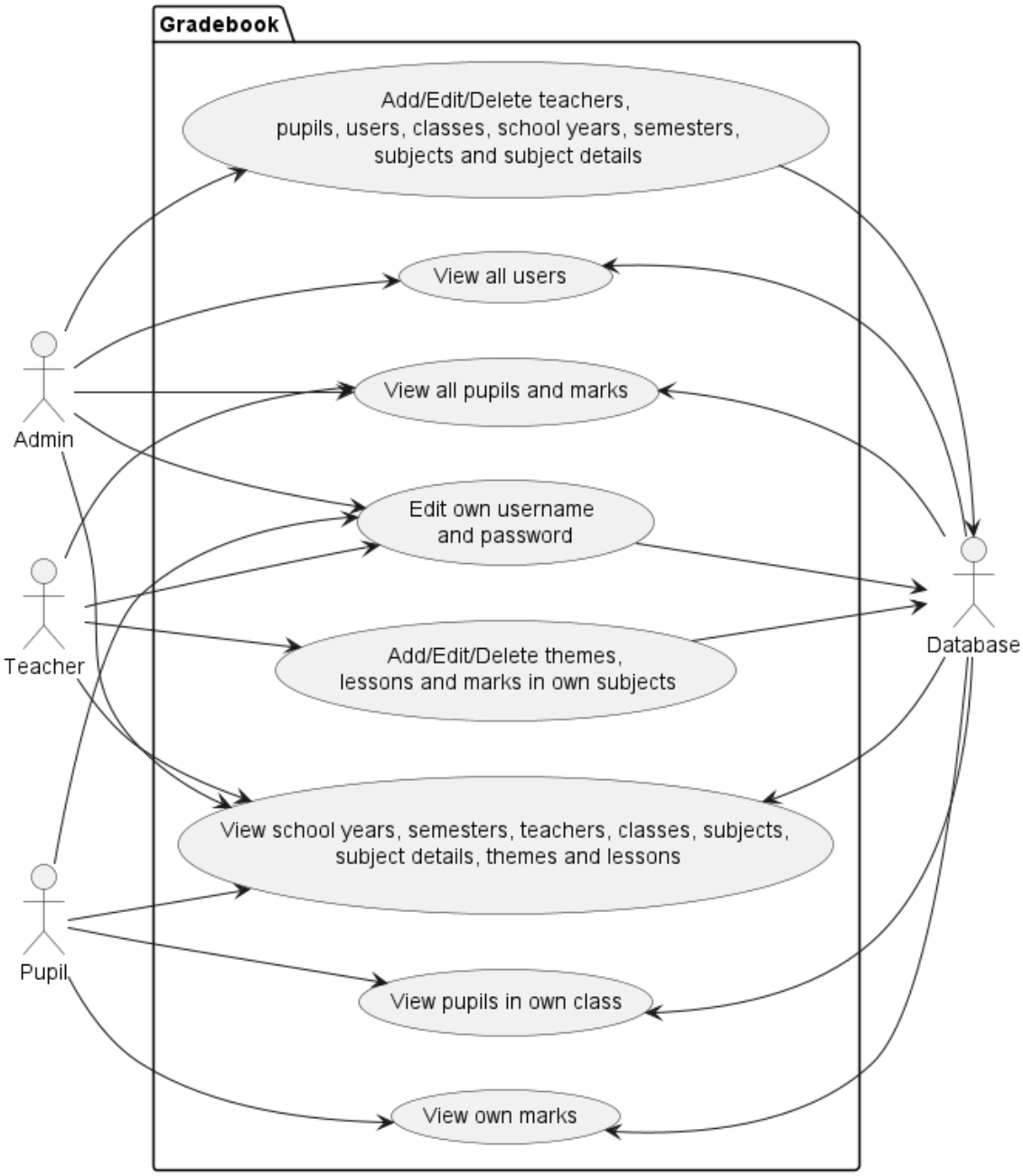


Рисунок 3.8 – діаграма варіантів використання

Учень має можливість переглядати свої оцінки, які можуть бути відібрані за одним із критеріїв: учень, предмет або урок. Крім того учень має можливість переглядати список учнів власного класу, а також списки навчальних років, семестрів, вчителів, класів та предметів, як повністю, так і відібраних за певним критерієм. Також учень має можливість переглядати список тем, відібраних за предметом та список уроків, відібраних за темою.

Учитель має можливість переглядати списки навчальних років, семестрів, вчителів, класів, предметів, тем та уроків так само як і учень. Крім того учитель може переглядати оцінки всіх учнів, та списки учнів у всіх класах. Також учитель має можливість додавати, редагувати та видаляти теми, уроки та оцінки, з тих предметів, які він викладає.

Адміністратор має можливість переглядати списки вчителів, класів, предметів, тем, уроків, оцінок та учнів так само як і вчитель. Крім того адміністратор має можливість переглядати список користувачів. Також адміністратор має можливість додавати, редагувати та видаляти навчальні роки, семестри, вчителів, учнів, предмети, та встановлювати відповідність між семестром, класом, предметом та вчителем.

Всі користувачі повинні мати можливість змінювати своє ім’я користувача та пароль. Адміністратор повинен мати можливість редагувати всіх користувачів, в тому числі і змінювати їх роль.

3.3 Діаграми послідовності

На рисунку 3.9 наведено діаграму послідовності завантаження сторінки для перегляду списку об’єктів:

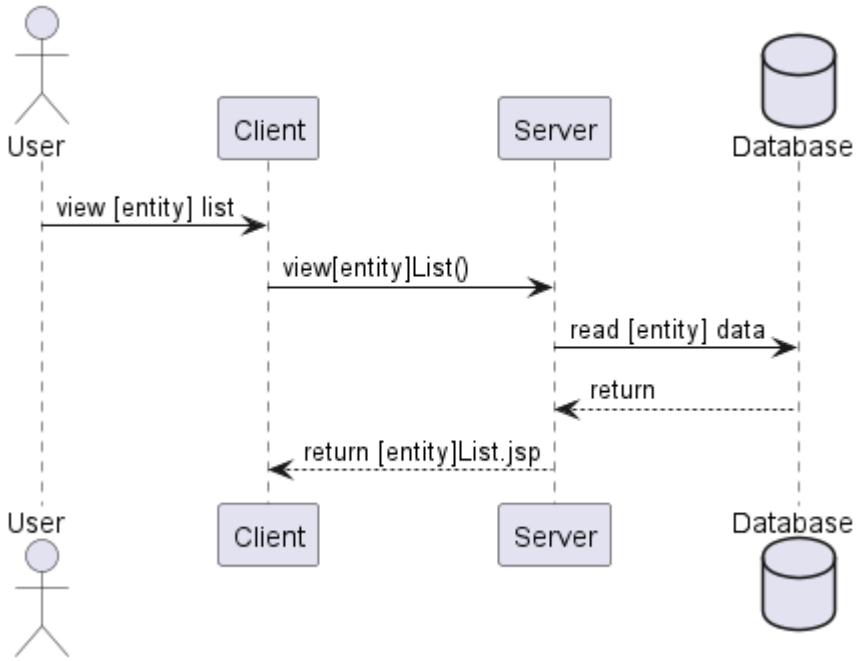


Рисунок 3.9 – діаграма послідовності для перегляду списку об’єктів

Опис до діаграми (рисунок 3.9):

- Користувач натискає на посилання для завантаження сторінки;
- Браузер відправляє відповідний запит на сервер;
- Сервер робить запит до бази даних;
- База даних повертає дані на сервер;
- Використовуючи дані, отримані від БД, сервер формує сторінку та повертає її клієнту.

На рисунку 3.10 зображено діаграму послідовності для додавання об’єктів:

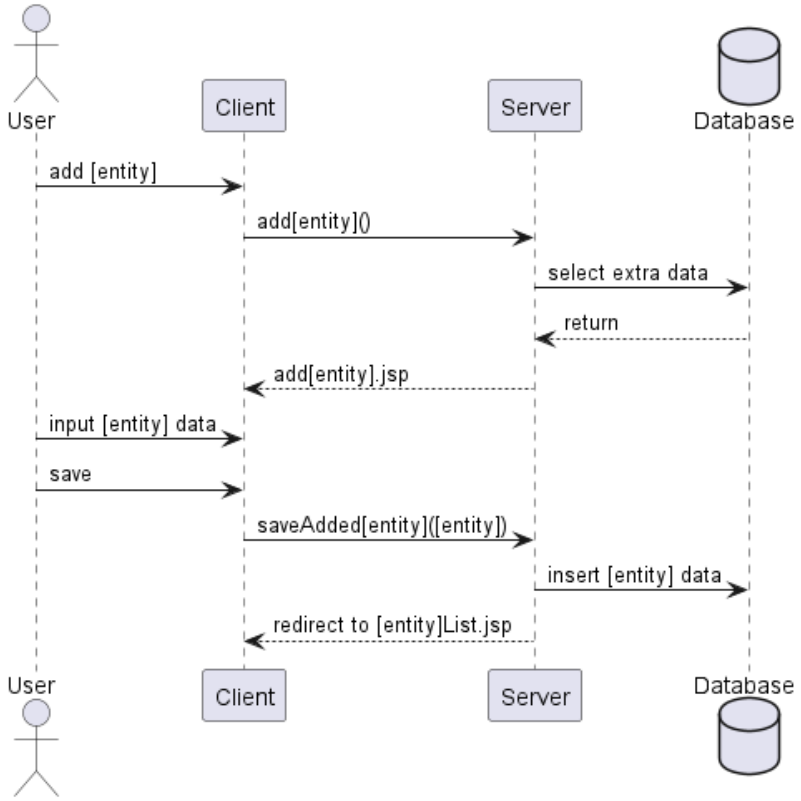


Рисунок 3.10 – діаграма послідовності для додавання об’єктів

Опис до діаграми (рисунок 3.10):

- Користувач натискає на посилання, яке спрямовує на сторінку для додавання об’єкту;
- Браузер робить запит на сервер;
- Якщо для форми додавання, необхідні додаткові дані, то сервер робить запит до БД, щоб отримати ці дані;
- БД повертає дані серверу;
- Сервер повертає клієнту сторінку для додавання об’єкту;
- Користувач вводить дані в поля форми;
- Користувач натискає на кнопку для збереження;
- Браузер надсилає введені дані на сервер;
- Сервер робить запит, для вставки отриманих даних до БД;
- Сервер здійснює перенаправлення на сторінку зі списком об’єктів.

На рисунку 3.11 наведено діаграму послідовності для видалення об’єкта:

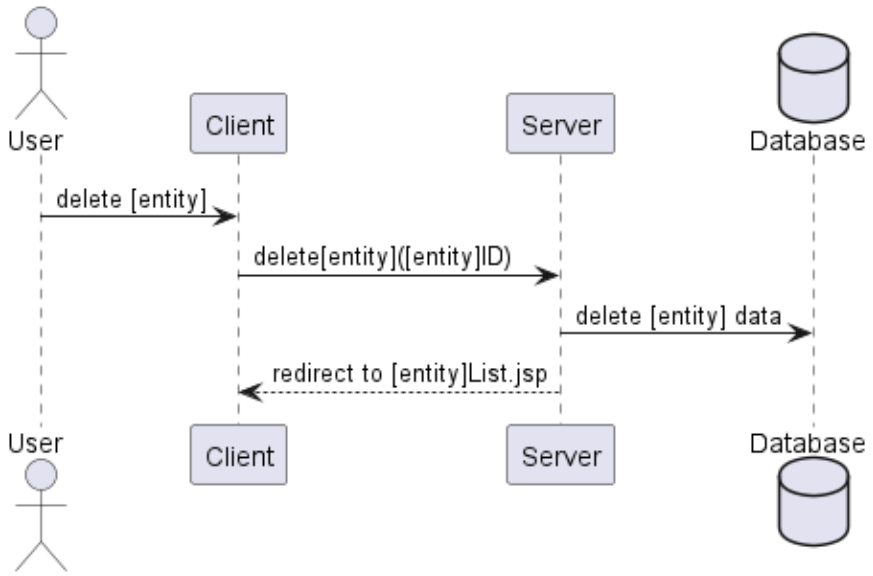


Рисунок 3.11 – діаграма послідовності для видалення записів деякої сутності

Опис до діаграми (рисунок 3.11):

- Користувач натискає на посилання для видалення об’єкта;
- Клієнт робить відповідний запит на сервер;
- Сервер робить запит для видалення об’єкта до БД;
- Сервер здійснює перенаправлення на сторінку зі списком об’єктів.

На рисунку 3.12 зображено діаграму послідовності для редагування об’єктів:

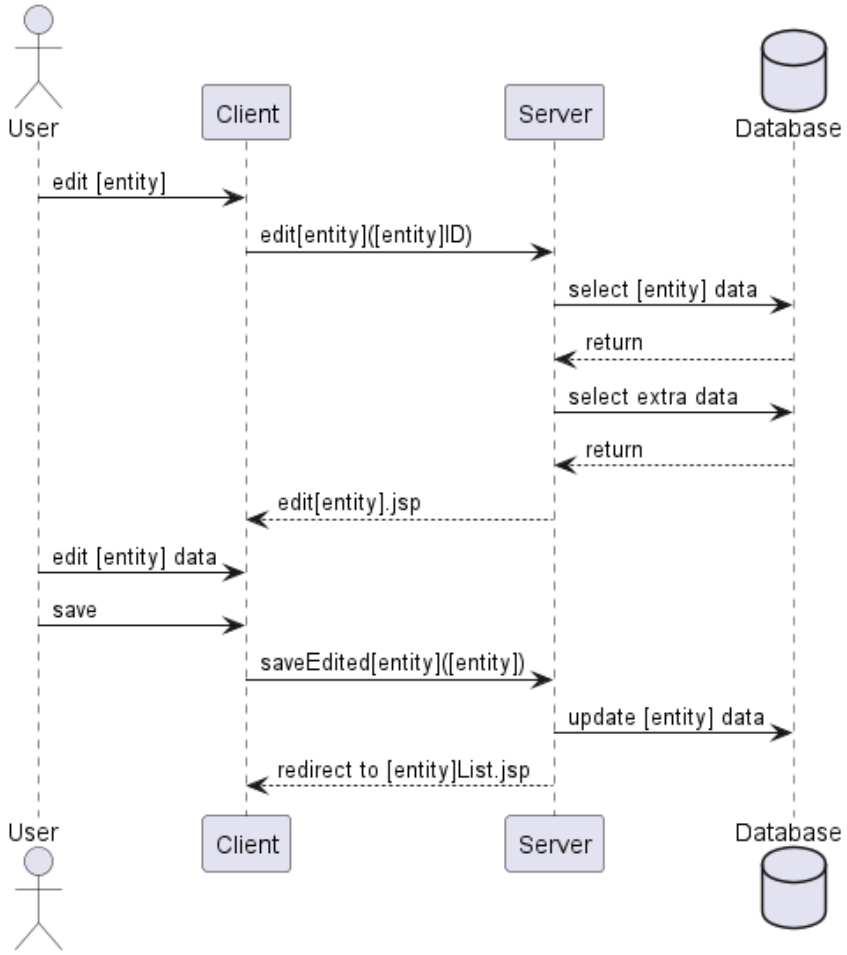


Рисунок 3.12 – діаграма послідовності для редагування записів деякої сутності

Опис до діаграми (рисунок 3.12):

- Користувач натискає на посилання, яке спрямовує на сторінку для редагування об’єкту;
- Браузер робить запит на сервер;
- Сервер робить запит до БД, щоб отримати дані про об’єкт, який потрібно відредагувати;
- БД повертає дані серверу;
- Якщо для форми редагування, необхідні додаткові дані, то сервер робить запит до БД, щоб отримати ці дані;
- БД повертає дані серверу;
- Сервер повертає клієнту сторінку для редагування об’єкту;

- Користувач вводить дані в поля форми;
- Користувач натискає на кнопку для збереження;
- Браузер надсилає введені дані на сервер;
- Сервер робить запит, для редагування об’єкта в БД;
- Сервер здійснює перенаправлення на сторінку зі списком об’єктів.

3.4 Огляд додатку

В додатку Б наведено частину коду для сутності урок. А саме, наведено клас, який описує сутність урок в рамках системи; клас для взаємодії з базою даних, для цієї сутності; контролер, який обробляє запити користувача, пов’язані з цією сутністю; код сторінки, яка містить список уроків; код сторінки, яка містить форму для додавання та редагування уроків.

Щоб продемонструвати роботу додатку «Gradebook», базу даних було наповнено тестовою інформацією.

Кожна сторінка додатку містить хедер та футер. Для неавторизованих користувачів, хедер містить посилання на сторінки авторизації та реєстрації, а також посилання на сторінку для додавання школи. Для кожної школи, при додаванні, створюється окрема база даних. Після додавання школи, відбувається переадресація на сторінку для додавання головного адміністратора школи. Обліковий запис головного адміністратора не можна видалити, щоб уникнути ситуації, коли всі облікові записи було видалено. Для авторизованих користувачів, головна сторінка додатку містить головне меню та зображення школи. Хедер містить скорочений варіант навігаційного меню, а також посилання на сторінку користувача та кнопку виходу. Меню в хедері та на головній сторінці залежить від ролі користувача. На рисунках 3.13-3.15 зображено головну сторінку для користувачів з різними ролями.

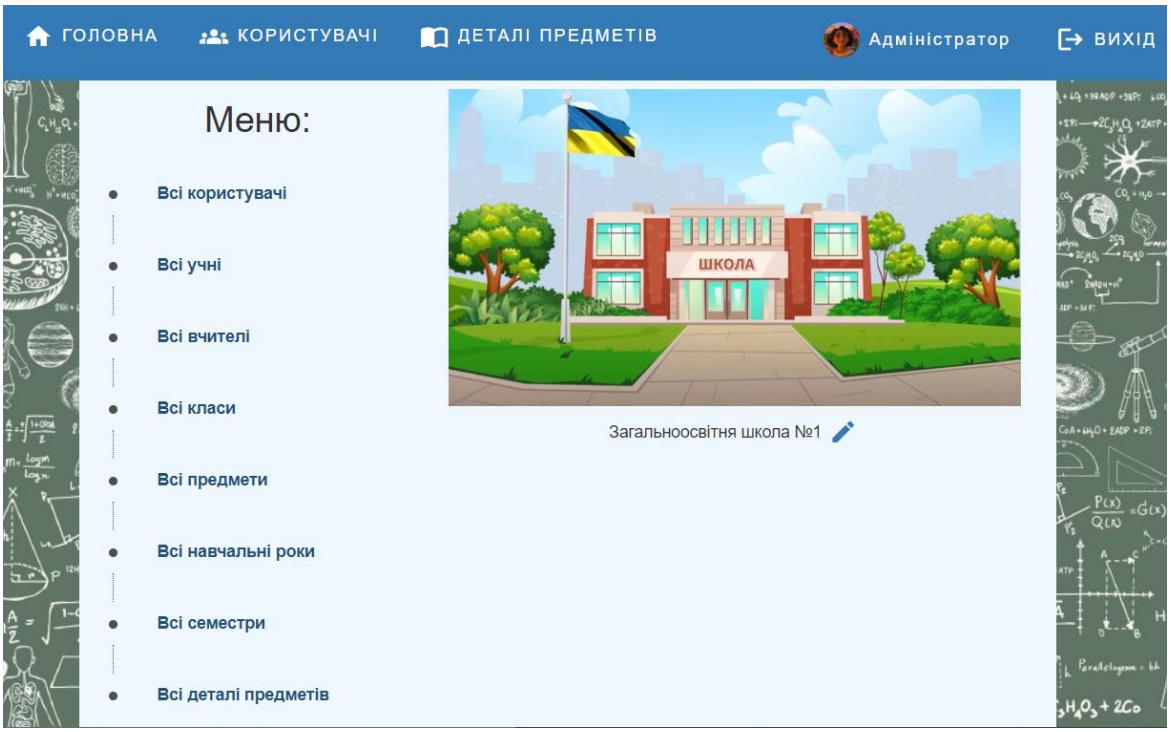


Рисунок 3.13 – Головна сторінка адміністратора

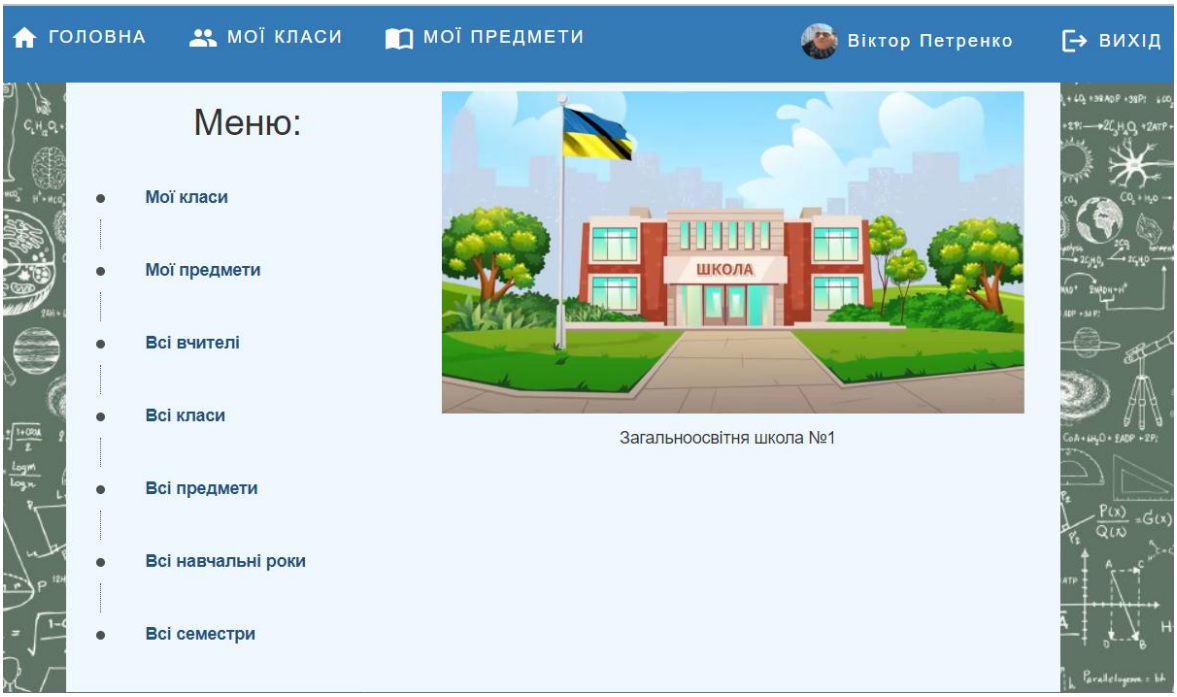


Рисунок 3.14 – Головна сторінка вчителя

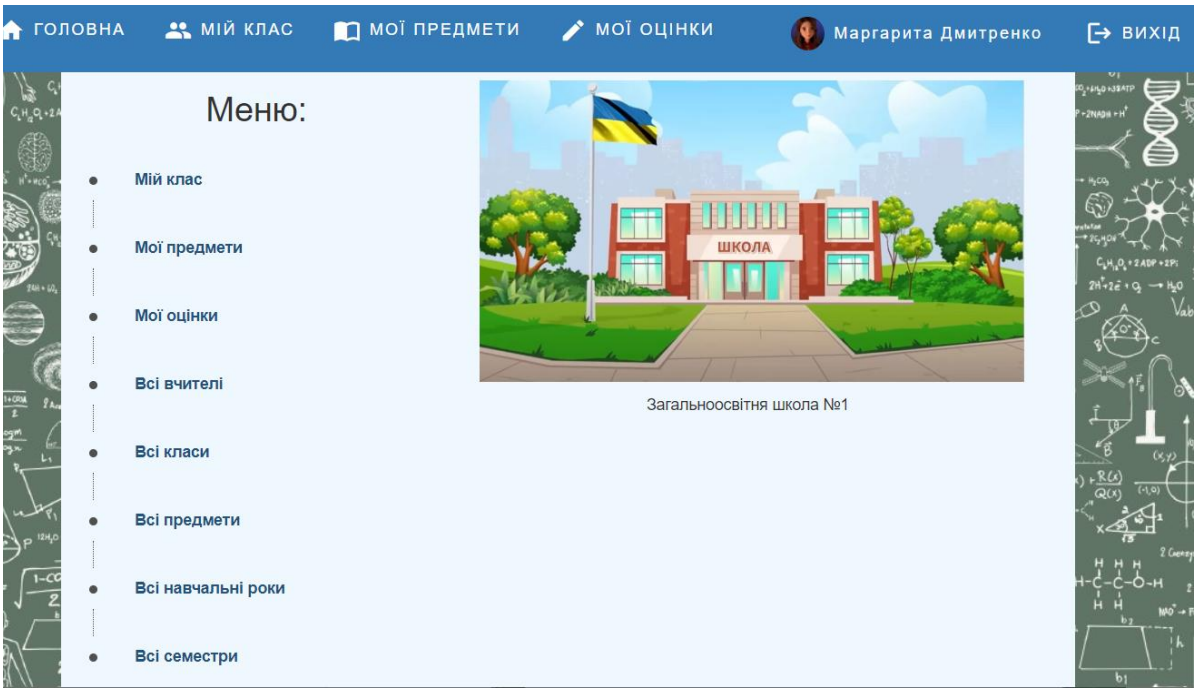


Рисунок 3.15 – Головна сторінка учня

Додаток містить сторінки для виведення списків: користувачів, вчителів, учнів, навчальних років, семестрів, класів, предметів, деталей предмету, тем, уроків та оцінок. Списки можуть бути повними, або відібраними за певним критерієм. Списки виводяться у вигляді таблиць. Сторінки зі списками містять посторінкове виведення а також пошук за кожним стовпцем таблиці.

Адміністратор може переглядати список користувачів системи, а також додавати, редагувати або видаляти їх. На рисунку 3.16 зображено таблицю зі списком користувачів. Таблиця містить посилання для редагування та видалення користувача, під таблицею міститься кнопка для додавання нового користувача.

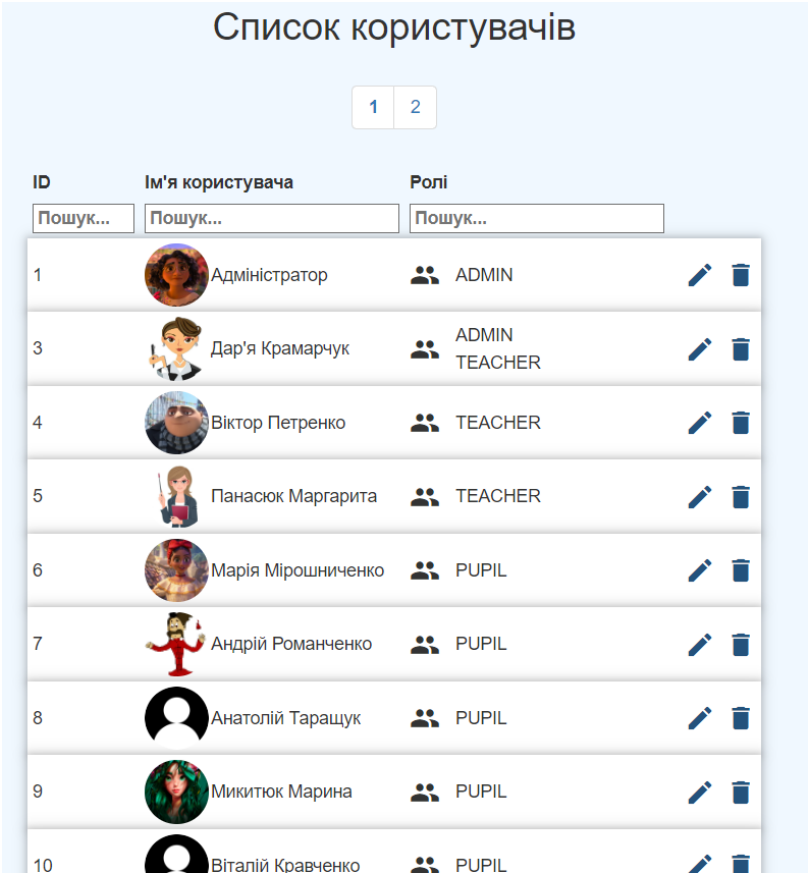


Рисунок 3.16 – Список користувачів

На рисунку 3.17 зображено результат пошуку за ім’ям користувача.

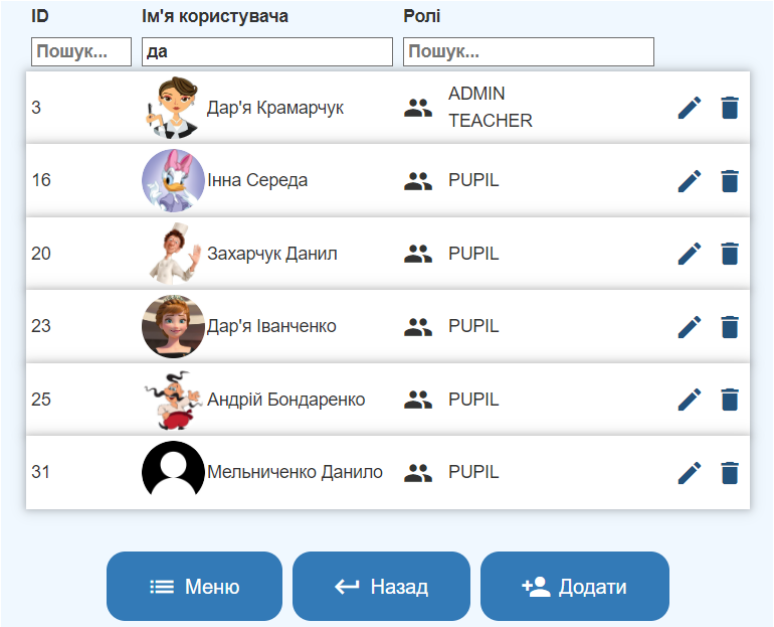


Рисунок 3.17 – Результат пошуку

Крім того додаток містить сторінки для додавання та редагування елементів, згаданих вище, списків. На рисунку 3.18 зображено контент сторінки користувача. Кожен користувач може змінити власне ім’я користувача, пароль та фото, всі інші дані змінює лише адміністратор. Для створення облікового запису є два способи: додавання користувача адміністратором або користувач може зареєструватися сам. Але користувач, який зареєструвався сам, не має жодної ролі, поки адміністратор не надасть йому певну роль.

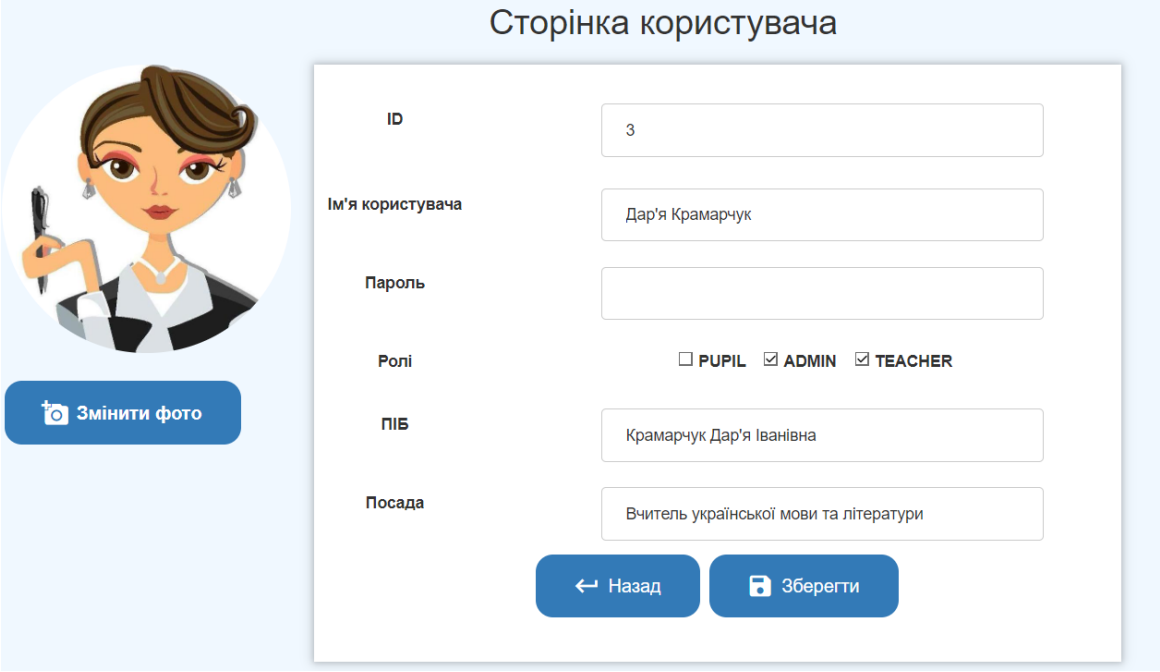
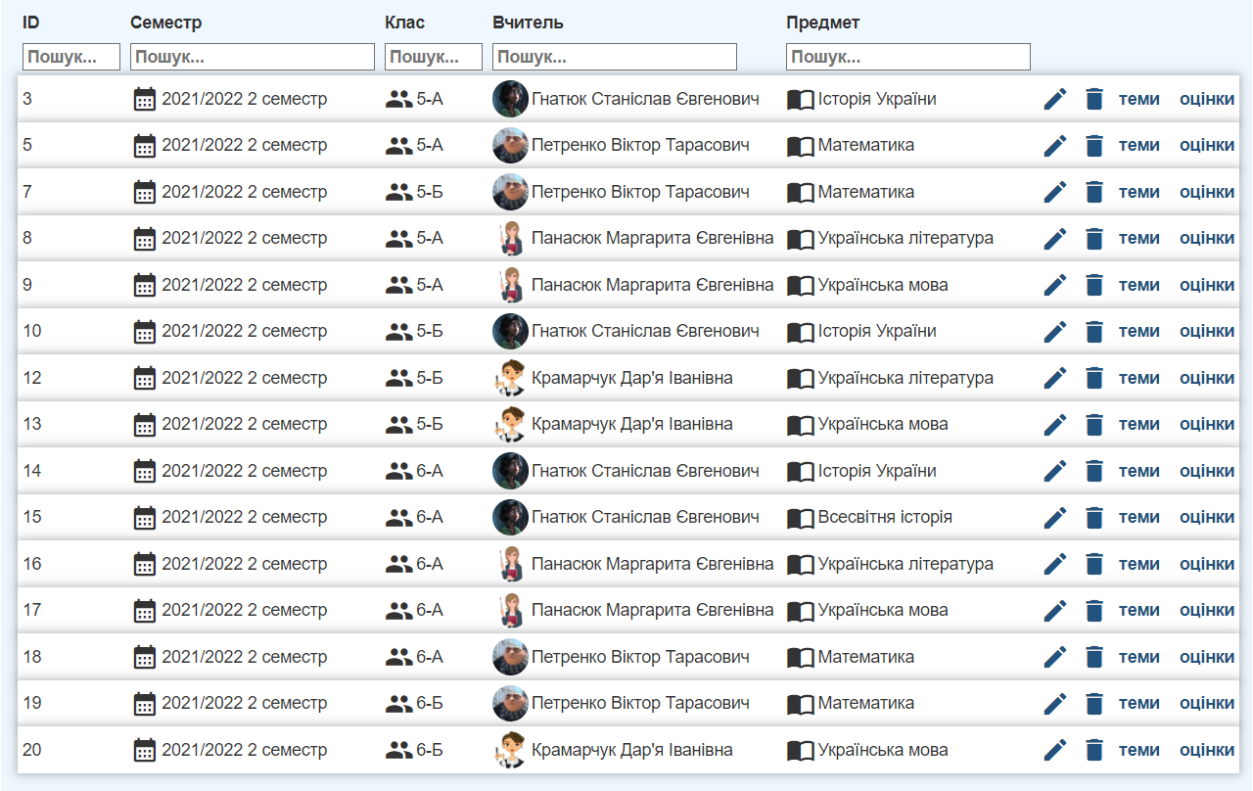


Рисунок 3.18 – сторінка користувача

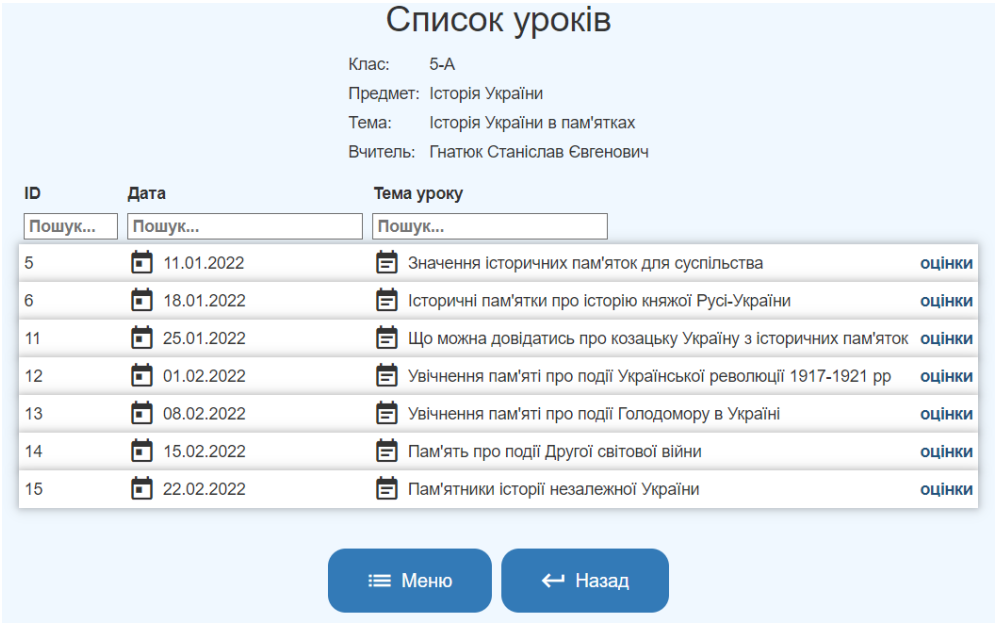
Адміністратор також може додавати, редагувати та видаляти: навчальні роки, семестри, предмети, та деталі предмета. На рисунку 3.19 зображено таблицю деталей предметів. За цією таблицею вчителям та учням надається доступ до предметів.



ID	Семестр	Клас	Вчитель	Предмет			
Пошук...	Пошук...	Пошук...	Пошук...	Пошук...			
3	2021/2022 2 семестр	5-А	Гнатюк Станіслав Євгенович	Історія України	теми	оцінки	
5	2021/2022 2 семестр	5-А	Петренко Віктор Тарасович	Математика	теми	оцінки	
7	2021/2022 2 семестр	5-Б	Петренко Віктор Тарасович	Математика	теми	оцінки	
8	2021/2022 2 семестр	5-А	Панасюк Маргарита Євгенівна	Українська література	теми	оцінки	
9	2021/2022 2 семестр	5-А	Панасюк Маргарита Євгенівна	Українська мова	теми	оцінки	
10	2021/2022 2 семестр	5-Б	Гнатюк Станіслав Євгенович	Історія України	теми	оцінки	
12	2021/2022 2 семестр	5-Б	Крамарчук Дар'я Іванівна	Українська література	теми	оцінки	
13	2021/2022 2 семестр	5-Б	Крамарчук Дар'я Іванівна	Українська мова	теми	оцінки	
14	2021/2022 2 семестр	6-А	Гнатюк Станіслав Євгенович	Історія України	теми	оцінки	
15	2021/2022 2 семестр	6-А	Гнатюк Станіслав Євгенович	Всесвітня історія	теми	оцінки	
16	2021/2022 2 семестр	6-А	Панасюк Маргарита Євгенівна	Українська література	теми	оцінки	
17	2021/2022 2 семестр	6-А	Панасюк Маргарита Євгенівна	Українська мова	теми	оцінки	
18	2021/2022 2 семестр	6-А	Петренко Віктор Тарасович	Математика	теми	оцінки	
19	2021/2022 2 семестр	6-Б	Петренко Віктор Тарасович	Математика	теми	оцінки	
20	2021/2022 2 семестр	6-Б	Крамарчук Дар'я Іванівна	Українська мова	теми	оцінки	

Рисунок 3.19 – Таблиця деталей предметів

Вчителі можуть додавати теми, уроки, ставити оцінки та відмічати відсутніх. На рисунку 3.20 зображено список уроків з теми «Історія України в пам’ятках».



Список уроків

Клас: 5-А
Предмет: Історія України
Тема: Історія України в пам'ятках
Вчитель: Гнатюк Станіслав Євгенович

ID	Дата	Тема уроку	
Пошук...	Пошук...	Пошук...	
5	11.01.2022	Значення історичних пам'яток для суспільства	оцінки
6	18.01.2022	Історичні пам'ятки про історію княжої Русі-України	оцінки
11	25.01.2022	Що можна довідатись про козацьку Україну з історичних пам'яток	оцінки
12	01.02.2022	Увічнення пам'яті про події Української революції 1917-1921 рр	оцінки
13	08.02.2022	Увічнення пам'яті про події Голодомору в Україні	оцінки
14	15.02.2022	Пам'ять про події Другої світової війни	оцінки
15	22.02.2022	Пам'ятники історії незалежної України	оцінки

Меню Назад

Рисунок 3.20 – Список уроків

Додавати оцінки та відмічати присутніх, вчитель може на двох сторінках: На сторінці, яка містить оцінки відібрані за уроком (рисунок 3.21) та на сторінці, де оцінки відібрані за деталями предмету (рисунок 3.22).

На полях для вводу оцінок передбачено валідацію вхідних даних. В ці поля можна вводити лише цілі числа від 1 до 12 або літеру «н».

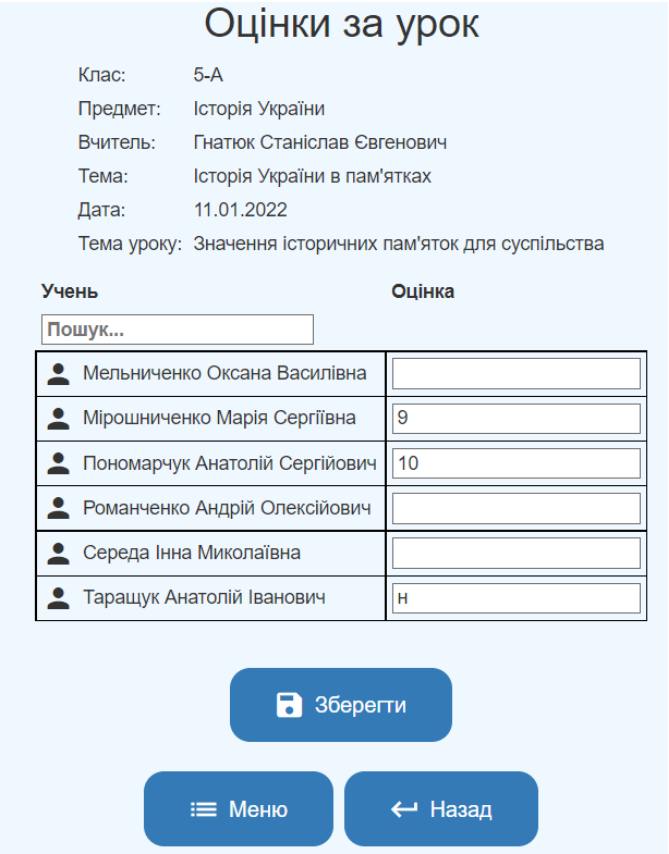


Рисунок 3.21 – Список оцінок за урок

На сторінці (рисунок 3.22) також виводяться тематичні та семестрова оцінки, які розраховуються автоматично. Тематичні оцінки розраховуються як середнє арифметичне від оцінок за уроки з даної теми. Семестрова оцінка розраховується як середнє арифметичне від тематичних оцінок.

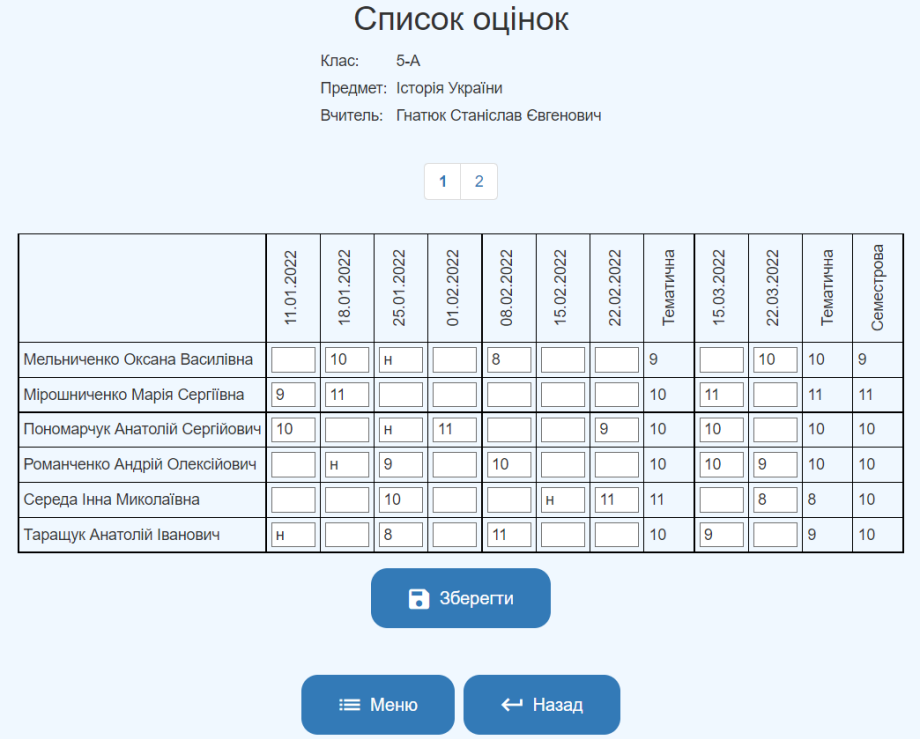


Рисунок 3.22 – Оцінки з предмету (сторінка для вчителя)

Ці ж сторінки (рисунки 3.21-3.22) використовуються також для перегляду оцінок учнями (рисунок 3.23), де учень може бачити лише свої оцінки.

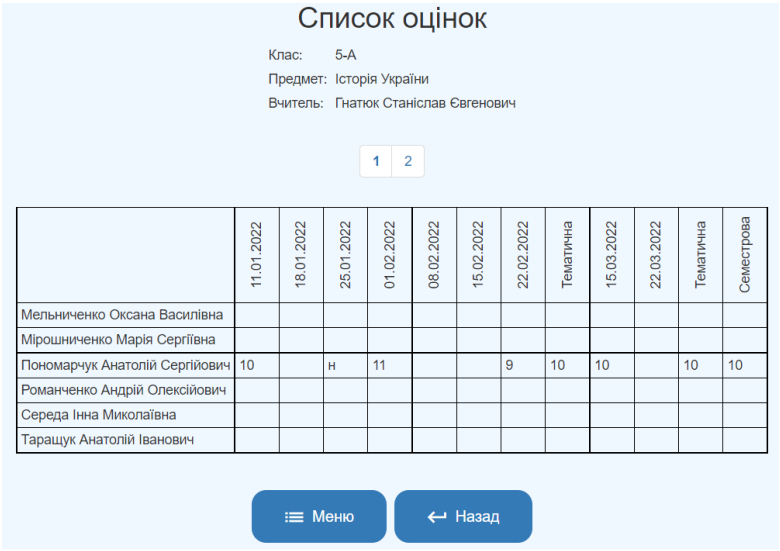


Рисунок 3.23 - Оцінки з предмету (сторінка для учня)

Також є сторінка, де учень може переглянути всі свої оцінки з усіх предметів (рисунок 3.24).

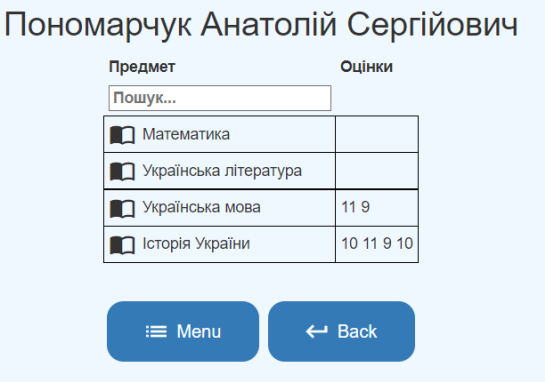


Рисунок 3.24 – Оцінки учня

На даній сторінці (рисунок 3.24) відображаються лише оцінки за уроки поточного семестру, без тематичних та семестрових.

ВИСНОВКИ

В результаті виконання дипломної роботи було розроблено веб-додаток «Gradebook». Для цього:

- було проведено аналіз аналогів;
- було обрано інструменти для реалізації додатку та проведено їх огляд;
- було спроектовано базу даних, у вигляді ER-діаграми;
- було спроектовано модель додатку, у вигляді діаграми варіантів використання та діаграми послідовності;
- було написано код;
- було здійснено огляд додатку.

Для написання серверної частини додатку використано мову Java та Spring Framework. Клієнтська частина додатку реалізована у вигляді сторінок JSP, за допомогою мови розмітки HTML. Авторизація та розділення на ролі реалізовано за допомогою Spring Security.

СПИСОК ЛІТЕРАТУРИ

1. Державні безкоштовні електронні щоденники та журнали для закладів загальної середньої освіти [Електронний ресурс] – Режим доступу до ресурсу: https://e-journal.iea.gov.ua
2. Електронні класні журнали та щоденники з можливостями дистанційного навчання [Електронний ресурс] – Режим доступу до ресурсу: https://nz.ua
3. Електронні щоденники та журнали [Електронний ресурс] – Режим доступу до ресурсу: https://e-schools.info/e-service
4. Електронний журнал | Електронний щоденник [Електронний ресурс] – Режим доступу до ресурсу: https://ukrschools.com.ua
5. Мова програмування Java [Електронний ресурс] – Режим доступу до ресурсу: http://javaphp.ptngu.com/javalang
6. Primitive Types and Values [Електронний ресурс] – Режим доступу до ресурсу: http://web.archive.org/web/20111124105915/http:/java.sun.com/docs/books/jvms/second_edition/html/Overview.doc.html#22239
7. Эккель Б. Философия Java. - СПб.: Питер, 2014. - 640 с.
8. Основні принципи ООП інкапсуляція, успадкування, поліморфізм. [Електронний ресурс] – Режим доступу до ресурсу: https://jak.koshachek.com/articles/osnovni-principi-oop-inkapsuljacija-uspadkuvannja.html
9. Основні поняття ООП [Електронний ресурс] – Режим доступу до ресурсу: http://www.e-helper.com.ua/node/104
10. Spring для ленивых. Основы, базовые концепции и примеры с кодом. Часть 1 [Електронний ресурс]. – Режим доступу: https://javarush.ru/groups/posts/spring-framework-java-1

11. Spring Framework Annotations [Електронний ресурс]. – Режим доступу: https://springframework.guru/spring-framework-annotations/
12. Introduction to Spring Framework [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/introduction-to-spring-framework/
13. Краткий обзор Spring Security [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/203318/
14. Introduction to Spring Method Security [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/spring-security-method-security
15. JavaScript [Електронний ресурс]. – Режим доступу: http://яваскрипт.укр/javascript
16. AJAX для новичков [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/14246/
17. What Is a Database? [Електронний ресурс]. – Режим доступу: https://www.oracle.com/database/what-is-database/
18. DB-Engines Ranking [Електронний ресурс]. – Режим доступу: https://db-engines.com/en/ranking
19. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org
20. What is PostgreSQL? [Електронний ресурс]. – Режим доступу: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/
21. Дунаев В. В. Базы данных. Язык SQL / В. В. Дунаев. – СанктПетербург: БХВ-Петербург, 2006. – 288 с.

Додаток А

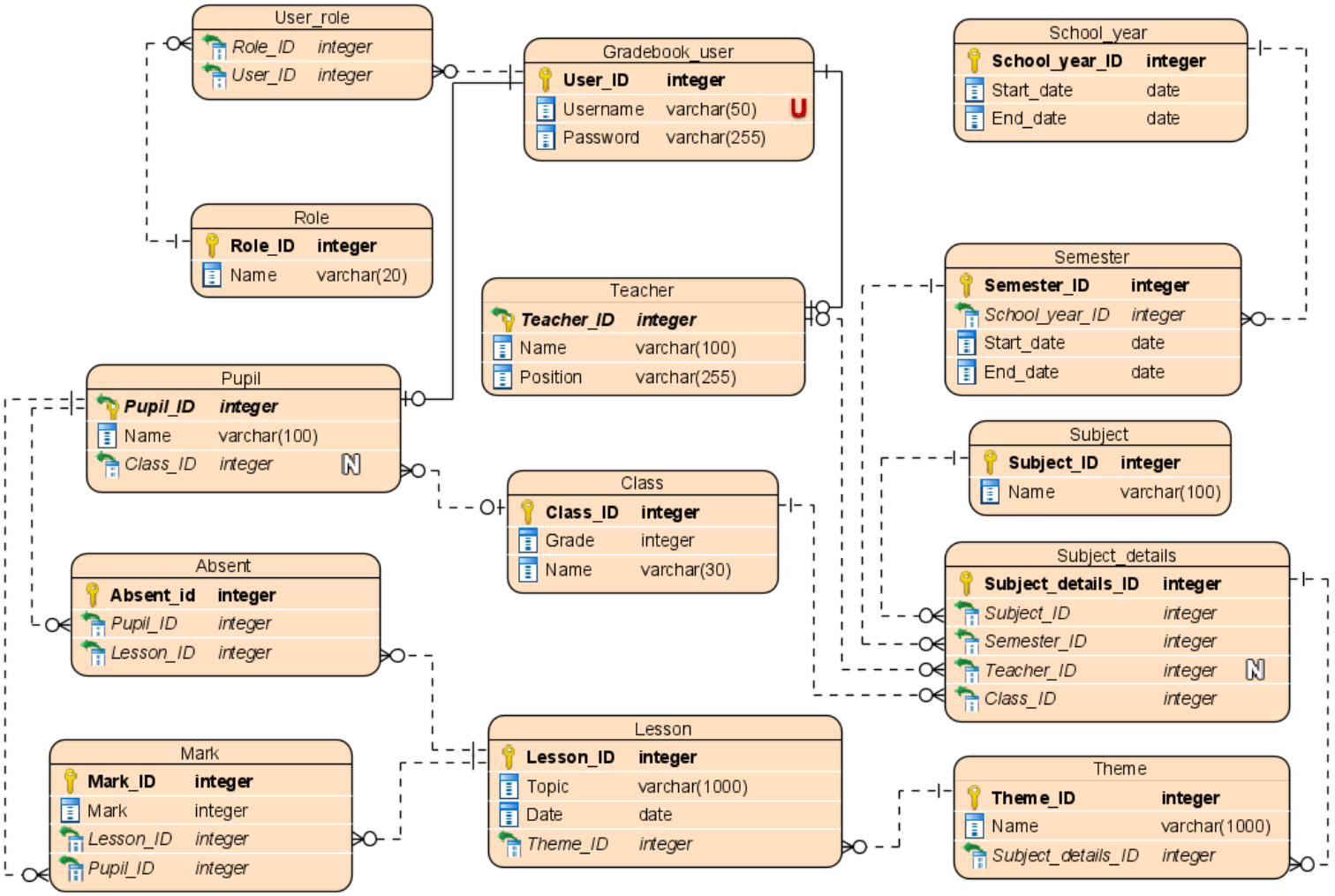


Рисунок А.1 – ER-діаграма

Додаток Б

```
public class Lesson {
  private int id;
  private Theme theme;
  private Date date;
  private String topic;

  public Lesson(int id, Theme theme, Date date, String topic) {
    this.id = id;
    this.theme = theme;
    this.date = date;
    this.topic = topic;
  }

  public Lesson() {}

  public Lesson(Theme theme) {
    this.theme = theme;
  }

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public Theme getTheme() {
    return theme;
  }

  public void setTheme(Theme theme) {
    this.theme = theme;
  }

  public Date getDate() {
    return date;
  }

  public void setDate(Date date) {
    this.date = date;
  }

  public String getTopic() {
    return topic;
  }

  public void setTopic(String topic) {
    this.topic = topic;
```

```
  }

  @Override
  public String toString() {
    return "Lesson{" +
           "id=" + id +
           ", theme=" + theme +
           ", date=" + date +
           ", topic='" + topic + '\'' +
           '}';
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Lesson lesson = (Lesson) o;
    return id == lesson.id && theme.equals(lesson.theme)
                && date.equals(lesson.date)
                && topic.equals(lesson.topic);
  }

  @Override
  public int hashCode() {
    return Objects.hash(id, theme, date, topic);
  }
}

public class PostgresLessonDAO implements LessonDAO {
  private static final String GET_LESSON = "SELECT * FROM LESSON where
LESSON_ID = ?";
  private static final String INSERT_LESSON = "Insert into LESSON
(theme_id, lesson_date, topic) values (?, ?, ?)";
  private static final String UPDATE_LESSON = "UPDATE LESSON set THEME_ID
= ?, LESSON_DATE = ?, TOPIC = ? where LESSON_ID = ?";
  private static final String DELETE_LESSON = "Delete from LESSON where
LESSON_ID = ?";
  private static final String GET_LESSONS_BY_THEME = "select * from LESSON
where THEME_ID = ? order by LESSON_DATE";
  private static final String GET_COUNT_OF_LESSONS_BY_SUBJECT_DETAILS =
"select count(lesson_id) as AMOUNT from lesson join theme using(theme_id)
where subject_details_id=?";
  private final ThemeDAO themeDAO;
  private final ConnectionPool connectionPool;
  private static final Logger LOGGER =
Logger.getLogger(PostgresLessonDAO.class.getName());
```

```
  public PostgresLessonDAO(ThemeDAO themeDAO, ConnectionPool
connectionPool) {
    this.themeDAO = themeDAO;
    this.connectionPool = connectionPool;
  }

  private Lesson parseLesson(ResultSet resultSet, String dbName) {
    Lesson lesson = null;
    try {
      int id = resultSet.getInt("lesson_ID");
      int themeID = resultSet.getInt("theme_id");
      Date data = resultSet.getDate("lesson_date");
      String topic = resultSet.getString("topic");
      lesson = new Lesson(id, themeDAO.getTheme(themeID, dbName), data,
topic);
    } catch (SQLException throwables) {
            LOGGER.error(throwables.getMessage(), throwables);
    }
    return lesson;
  }

  @Override
  public Lesson getLesson(int id, String dbName) {
    LOGGER.info("Reading lesson " + id + " from database.");
    Lesson lesson = null;
    try (Connection connection = connectionPool.getConnection(dbName);
         PreparedStatement preparedStatement =
connection.prepareStatement(GET_LESSON)) {
      preparedStatement.setInt(1, id);
      try (ResultSet resultSet = preparedStatement.executeQuery()) {
        if (resultSet.next()) {
          lesson = parseLesson(resultSet, dbName);
        }
        LOGGER.info("Reading complete");
      }
    } catch (SQLException throwables) {
      LOGGER.error(throwables.getMessage(), throwables);
    }
    return lesson;
```

```
  }

  @Override
  public void addLesson(Lesson lesson, String dbName) {
    LOGGER.info("Inserting lesson " + lesson.getId() +
" into database.");
    try (Connection connection = connectionPool.getConnection(dbName);
         PreparedStatement preparedStatement =
connection.prepareStatement(INSERT_LESSON)) {
      preparedStatement.setInt(1, lesson.getTheme().getId());
      preparedStatement.setDate(2, lesson.getDate());
      preparedStatement.setString(3, lesson.getTopic());
      preparedStatement.executeUpdate();
      LOGGER.info("Inserting complete.");
    } catch (SQLException throwables) {
      LOGGER.error(throwables.getMessage(), throwables);
    }
  }

  @Override
  public void updateLesson(Lesson lesson, String dbName) {
    LOGGER.info("Updating lesson " + lesson.getId() + ".");
    try (Connection connection = connectionPool.getConnection(dbName);
         PreparedStatement preparedStatement =
connection.prepareStatement(UPDATE_LESSON)) {
      preparedStatement.setInt(1, lesson.getTheme().getId());
      preparedStatement.setDate(2, lesson.getDate());
      preparedStatement.setString(3, lesson.getTopic());
      preparedStatement.setInt(4, lesson.getId());
      preparedStatement.executeUpdate();
      LOGGER.info("Updating complete.");
    } catch (SQLException throwables) {
      LOGGER.error(throwables.getMessage(), throwables);
    }
  }

  @Override
  public void deleteLesson(int id, String dbName) {
    try (Connection connection = connectionPool.getConnection(dbName);
         PreparedStatement preparedStatement =
```

```
connection.prepareStatement(DELETE_LESSON)) {
      LOGGER.info("Deleting " + id + " lesson.");
      preparedStatement.setInt(1, id);
      preparedStatement.executeUpdate();
      LOGGER.info("Deleting complete.");
    } catch (SQLException throwables) {
      LOGGER.error(throwables.getMessage(), throwables);
    }
  }

  @Override
  public List<Lesson> getLessonsByTheme(int id, String dbName) {
    LOGGER.info("Reading lessons for " + id + " theme.");
    List<Lesson> list = new ArrayList<>();
    try (Connection connection = connectionPool.getConnection(dbName);
         PreparedStatement preparedStatement =
connection.prepareStatement(GET_LESSONS_BY_THEME)) {
      preparedStatement.setInt(1, id);
      try (ResultSet resultSet = preparedStatement.executeQuery()){
        while (resultSet.next()) {
          list.add(parseLesson(resultSet, dbName));
        }
        LOGGER.info("List of lessons complete.");
      }
    } catch (SQLException throwables) {
      LOGGER.error(throwables.getMessage(), throwables);
    }
    return list;
  }

  @Override
  public int getCountOfLessonsBySubjectDetails(int id,
String dbName) {
    LOGGER.info("Counting themes for " + id + " subject details.");
    int count = 0;
    try (Connection connection = connectionPool.getConnection(dbName);
         PreparedStatement preparedStatement =
connection.prepareStatement(GET_COUNT_OF_LESSONS_BY_SUBJECT_DETAILS)) {
      preparedStatement.setInt(1, id);
      try (ResultSet resultSet = preparedStatement.executeQuery()){
```

```
        resultSet.next();
        count = resultSet.getInt("AMOUNT");
        LOGGER.info("Counting complete.");
      }
    } catch (SQLException throwables) {
      LOGGER.error(throwables.getMessage(), throwables);
    }
    return count;
  }
}

@Controller
public class LessonController {
  private final LessonDAO dao;
  private final ThemeDAO themeDAO;
  private final SchoolDAO schoolDAO;
  private static final Logger LOGGER =
Logger.getLogger(LessonController.class.getName());

  public LessonController(LessonDAO dao, ThemeDAO themeDAO, SchoolDAO
schoolDAO) {
    this.dao = dao;
    this.themeDAO = themeDAO;
    this.schoolDAO = schoolDAO;
  }

  @RequestMapping(value = "/theme/{id}/lesson")
  @Secured("TEACHER")
  public ModelAndView addLesson(@PathVariable int id) {
    LOGGER.info("Add new lesson for theme " + id + ".");
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String dbName = user.getDbName();
    Theme theme = themeDAO.getTheme(id, dbName);
    if (theme == null) {
      LOGGER.error("Theme " + id + " not found.");
      return new ModelAndView("errorPage", HttpStatus.NOT_FOUND);
    }
    if (user.getId() != theme.getSubjectDetails().getTeacher().getId()) {
      return new ModelAndView("errorPage", HttpStatus.FORBIDDEN);
```

```
    }
    LOGGER.info("Form a model.");
    Map<String, Object> model = new HashMap<>();
    School school = schoolDAO.getSchool(Integer.parseInt(dbName));
    model.put("school", school);
    Map<String, String> crumbsMap = getBasicCrumbsMap(theme);
    crumbsMap.put("Додати урок", "");
    model.put("crumbs", BreadcrumbsController.getBreadcrumbs(crumbsMap));
    model.put("command", new Lesson(theme));
    model.put("title", "Додати урок");
    model.put("formAction", "../../lesson");
    LOGGER.info("Printing form for input lesson's data.");
    return new ModelAndView("lessonForm", model);
  }

  @RequestMapping(value = "/lesson", method = RequestMethod.POST)
  @Secured("TEACHER")
  public ModelAndView saveAddedLesson(@ModelAttribute Lesson lesson) {
    LOGGER.info("Saving added lesson.");
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    dao.addLesson(lesson, user.getDbName());
    LOGGER.info("Redirect to list of lessons for " +
lesson.getTheme().getId() + " theme.");
    return new ModelAndView("redirect:/theme/" + lesson.getTheme().getId()
+ "/lessons");
  }

  @RequestMapping(value = "/lesson/{id}", method = RequestMethod.GET)
  @Secured("TEACHER")
  public ModelAndView editLesson(@PathVariable int id) {
    LOGGER.info("Edit lesson " + id + ".");
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String dbName = user.getDbName();
    Lesson lesson = dao.getLesson(id, dbName);
    if (lesson == null) {
      LOGGER.error("Lesson " + id + " not found.");
      return new ModelAndView("errorPage", HttpStatus.NOT_FOUND);
    }
```

```
    Theme theme = lesson.getTheme();
    if (user.getId() != theme.getSubjectDetails().getTeacher().getId()) {
      return new ModelAndView("errorPage", HttpStatus.FORBIDDEN);
    }
    Map<String, Object> model = new HashMap<>();
    School school = schoolDAO.getSchool(Integer.parseInt(dbName));
    model.put("school", school);
    Map<String, String> crumbsMap = getBasicCrumbsMap(theme);
    crumbsMap.put("Редагувати урок", "");
    model.put("crumbs", BreadcrumbsController.getBreadcrumbs(crumbsMap));
    model.put("title", "Редагувати урок");
    model.put("command", lesson);
    model.put("formAction", "../lesson/" + lesson.getId());
    LOGGER.info("Printing form for changing lesson's data.");
    return new ModelAndView("lessonForm", model);
  }

  @RequestMapping(value = "/lesson/{id}", method = RequestMethod.POST)
  @Secured("TEACHER")
  public ModelAndView saveEditedLesson(@ModelAttribute Lesson lesson) {
    LOGGER.info("Saving edited lesson.");
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    dao.updateLesson(lesson, user.getDbName());
    LOGGER.info("Redirect to list of lessons for " +
lesson.getTheme().getId() + " theme.");
    return new ModelAndView("redirect:/theme/" + lesson.getTheme().getId()
+ "/lessons");
  }

  @RequestMapping(value = "/lesson/{id}/delete", method =
RequestMethod.POST)
  @Secured("TEACHER")
  public ModelAndView deleteLesson(@PathVariable int id) {
    LOGGER.info("Deleting lesson " + id);
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    Lesson lesson = dao.getLesson(id, user.getDbName());
    if (lesson == null) {
      LOGGER.error("Lesson " + id + " not found.");
```

```
      return new  ModelAndView("errorPage", HttpStatus.NOT_FOUND);
    }
    Theme theme = lesson.getTheme();
    if (user.getId() != theme.getSubjectDetails().getTeacher().getId()) {
      return new ModelAndView("errorPage", HttpStatus.FORBIDDEN);
    }
    dao.deleteLesson(id, user.getDbName());
    LOGGER.info("Redirect to list of lessons for " + theme.getId() + "
theme.");
    return new ModelAndView("redirect:/theme/" + theme.getId() +
"/lessons");
  }

  @RequestMapping(value = "/theme/{id}/lessons")
  @Secured({"ADMIN", "TEACHER", "PUPIL"})
  public ModelAndView viewLessonsByTheme(@PathVariable int id) {
    LOGGER.info("Getting list of lessons for " + id + " theme.");
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String dbName = user.getDbName();
    Theme theme = themeDAO.getTheme(id, user.getDbName());
    if (theme == null) {
      LOGGER.error("Theme " + id + " not found.");
      return new ModelAndView("errorPage", HttpStatus.NOT_FOUND);
    }
    List<Lesson> list;
    Map<String, Object> model = new HashMap<>();
    School school = schoolDAO.getSchool(Integer.parseInt(dbName));
    model.put("school", school);
    list = dao.getLessonsByTheme(id, dbName);
    model.put("crumbs",
BreadcrumbsController.getBreadcrumbs(getBasicCrumbsMap(theme)));
    model.put("list", list);
    model.put("theme", theme);
    LOGGER.info("Printing lessons list.");
    return new ModelAndView("viewLessonList", model);
  }

  private Map<String, String> getBasicCrumbsMap(Theme theme) {
```

```
    User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    Map<String, String> crumbsMap = new LinkedHashMap<>();
    if (user.hasRole("ADMIN")) {
      crumbsMap.put("Деталі предметів", "/subject-details?page=1");
    } else if (user.hasRole("TEACHER")) {
      crumbsMap.put("Деталі предметів", "/teacher/" + user.getId() +
"/subject-details");
    } else if (user.hasRole("PUPIL")) {
      crumbsMap.put("Деталі предметів", "/pupil/" + user.getId() +
"/subject-details");
    }
    crumbsMap.put("Теми", "/subject-details/" +
theme.getSubjectDetails().getId() + "/themes");
    crumbsMap.put("Уроки", "/theme/" + theme.getId() + "/lessons");
    return crumbsMap;
  }
}

<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags"
%>
<%@ page import="java.util.List" %>
<%@ page import="org.example.entities.*" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="java.text.DateFormat" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
 <head>
  <title>Список уроків</title>
  <link rel="icon" type="img/png" href="../images/icon.png">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com
/bootstrap/3.3.7/css/bootstrap.min.css" >
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js
/bootstrap.min.js"></script>
  <style><%@include file="../css/style.css"%></style>
 </head>
 <body>
  <%@include file="header.jsp"%>
  <div align="center">
   <div align="center" class="box">
```

```
<br/>
<ul class="breadcrumb"><%=request.getAttribute("crumbs")%></ul>
<h2>Список уроків</h2>
<%
  Theme theme = (Theme) request.getAttribute("theme");
  SubjectDetails subjectDetails = theme.getSubjectDetails();
  Teacher teacher = subjectDetails.getTeacher();
  List<Lesson> list = (List<Lesson>)request.getAttribute("list");
  int colspan = 3;
%>
<table>
 <tr>
  <td>Клас:</td>
  <td><%=subjectDetails.getPupilClass().getName()%></td>
 </tr>
 <tr>
  <td>Предмет:</td>
  <td><%=subjectDetails.getSubject().getName()%></td>
 </tr>
 <tr>
  <td>Тема:</td>
  <td><%=theme.getName()%></td>
 </tr>
 <%
   if(teacher != null) {
 %>
 <tr>
  <td>Вчитель:</td>
  <td><%=teacher.getName()%></td>
 </tr>
 <%
   }
 %>
</table>
<table id="myTable">
 <tr>
  <sec:authorize access="hasAuthority('ADMIN')">
   <th>ID</th>
   <%colspan++;%>
  </sec:authorize>
```

```
 <th>Дата</th>
 <th>Тема уроку</th>
 <th></th>
 <sec:authorize access="hasAuthority('TEACHER')">
  <%
    if(teacher != null && currUser.getId() == teacher.getId()) {
  %>
     <th></th>
     <th></th>
  <%
    colspan += 2;
  %>
 </sec:authorize>
</tr>
<tr>
 <%
   int i = 0;
 %>
 <sec:authorize access="hasAuthority('ADMIN')">
  <th>
   <input type="text"
          id="id"
          onkeyup="filter(id, <%=i++%>)"
          class="search-slim"
          placeholder="Пошук...">
  </th>
 </sec:authorize>
  <th>
   <input type="text"
          id="date"
          onkeyup="filter(id, <%=i++%>)"
          class="search"
          placeholder="Пошук...">
  </th>
  <th>
   <input type="text"
          id="topic"
          onkeyup="filter(id, <%=i%>)"
          class="search"
          placeholder="Пошук...">
```

```
 </th>
 <th></th>
 <sec:authorize access="hasAuthority('TEACHER')">
  <%
    if(teacher != null && currUser.getId() == teacher.getId()) {
  %>
     <th></th>
     <th></th>
  <%
    }
  %>
 </sec:authorize>
</tr>
<tbody>
 <%
   if (list.isEmpty()) {
 %>
    <tr class="card">
     <td colspan="<%=colspan%>">Список уроків пустий</td>
    </tr>
 <%
   }
   for (Lesson lesson:list) {
    DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy");
    String strDate = dateFormat.format(lesson.getDate());
 %>
 <tr class="card">
 <sec:authorize access="hasAuthority('ADMIN')">
  <td><%=lesson.getId()%></td>
 </sec:authorize>
 <td>
  <div class="inline"><i class='material-icons'>today</i></div>
  <div class="inline"><%=strDate%></div>
 </td>
 <td>
  <div class="inline">
   <i class='material-icons'>event_note </i>
  </div>
  <div class="inline"><%=lesson.getTopic()%></div>
 </td>
```

```
    <td>
     <a href="<%=root%>lesson/<%=lesson.getId()%>/marks">
      Оцінки
     </a>
    </td>
    <sec:authorize access="hasAuthority('TEACHER')">
     <%
       if(teacher != null && currUser.getId() == teacher.getId()) {
     %>
        <td>
         <a href="../../lesson/<%=lesson.getId()%>">
          <i class="material-icons">edit</i>
         </a>
        </td>
        <td>
         <a>
          <form action="<%=root%>lesson/<%=lesson.getId()%>/delete"
                method=post>
           <sec:csrfInput />
           <button type="submit">
            <i class="material-icons">delete</i>
           </button>
          </form>
         </a>
        </td>
     <%
       }
     %>
    </sec:authorize>
   </tr>
   <%
     }
   %>
  </tbody>
</table>
<br/>
<button onclick='location.href="<%=root%>main"'
        class="bg-primary">
 <div class="inline"><i class='material-icons'>list</i></div>
 <div class="inline">Меню</div>
```

```
    </button>
    <button onclick=history.back() class="bg-primary">
     <div class="inline">
      <i class='material-icons'>keyboard_return</i>
     </div>
     <div class="inline">Назад</div>
    </button>
    <sec:authorize access="hasAuthority('TEACHER')">
     <%
       if(teacher != null && currUser.getId() == teacher.getId()) {
     %>
        <button onclick='location.href="lesson"' class="bg-primary">
         <div class="inline">
          <i class='material-icons'>edit_calendar</i>
         </div>
         <div class="inline">Додати</div>
        </button>
     <%
       }
     %>
    </sec:authorize>
   </div>
  </div>
  <%@include file="footer.jsp"%>
 </body>
 <script>
  <%@include file="../js/filter.js"%>
 </script>
</html>

<%@ page import="org.example.entities.Theme" %>
<%@ page import="org.example.entities.SubjectDetails" %>
<%@ page import="org.example.entities.Lesson" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
 <head>
  <title><%=request.getAttribute("title")%></title>
  <link rel="icon" type="img/png" href="../images/icon.png">
```

```
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com
/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
/js/bootstrap.min.js"></script>
  <style><%@include file="../css/style.css"%></style>
 </head>
 <body>
  <%@include file="header.jsp"%>
  <div align="center">
   <div align="center" class="box">
    <br/>
    <ul class="breadcrumb"><%=request.getAttribute("crumbs")%></ul>
    <br/>
    <div class="card" style="width: 50%">
     <br/>
     <h2 align="center"><%=request.getAttribute("title")%></h2>
     <%
       Lesson lesson = (Lesson) request.getAttribute("command");
       Theme theme = lesson.getTheme();
       SubjectDetails subjectDetails = theme.getSubjectDetails();
     %>
     <p>Предмет:<%=subjectDetails.getSubject().getName()%></p>
     <p>Вчитель:<%=subjectDetails.getTeacher().getName()%></p>
     <p>Клас:<%=subjectDetails.getPupilClass().getName()%></p>
     <p>Тема:<%=theme.getName()%></p>
     <form:form>
      <br/>
      <div class="row">
       <div class="col-25">
        <label>Дата</label>
       </div>
       <div class="col-75">
        <form:input type="date" path="date" required="true"/>
        <br/><br/>
       </div>
      </div>
      <div class="row">
       <div class="col-25">
        <label>Тема уроку</label>
       </div>
```

```
        <div class="col-75">
         <form:input path="topic" required="true"/><br/><br/>
        </div>
       </div>
       <form:input path="id" type="hidden"/>
       <form:input path="theme.id" type="hidden"/>
       <button onclick="history.back()"
               type="button" class="bg-primary">
        <div class="inline">
         <i class='material-icons'>keyboard_return</i>
        </div>
        <div class="inline">Назад</div>
       </button>
       <button formmethod="post"
               formaction="<%=request.getAttribute("formAction")%>"
               class="bg-primary">
        <div class="inline"><i class='material-icons'>save</i></div>
        <div class="inline">Зберегти</div>
       </button>
       <br/><br/>
      </form:form>
     </div>
    </div>
   </div>
   <%@include file="footer.jsp"%>
  </body>
</html>
```