Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра мультимедійних систем

Курсова робота

на тему: «**Розробка веб-застосунку для спрощення керування шкільними процесами»**

Виконав: студент 3-го року навчання,

спеціальності

121 «Інженерія Програмного Забезпечення»

Пінкевич В.М.

Керівник курсової роботи

Старший викладач

Борозенний С.О.

5 червня 2022 р.

Київ – 2022

Календарний план виконання курсової роботи

Тема: Розробка веб-застосунку для спрощення керування шкільними процесами

Календарний план виконання роботи**:**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи	Жовтень 2021 р.	
2.	Отримання завдання на курсову роботу	Листопад 2021 р.	
3.	Ознайомлення із літературою та джерелами за темою роботи	Листопад - грудень 2021 р.	
4.	Пошук та огляд існуючих аналогів застосунку	Грудень 2021 р.	
5.	Знайомство із засобами розробки застосунку	Січень 2022 р.	
6.	Розробка серверної частини застосунку	Січень – лютий 2022 р.	
7.	Розробка клієнтської частини застосунку	Квітень - травень 2022 р.	
8.	Написання текстової частини роботи	Травень 2022 р.	
9.	Перегляд курсової роботи науковим керівником	Травень 2022 р.	
10.	Внесення змін до курсової роботи згідно з зауваженням наукового керівника	Травень 2022 р.	
11.	Захист курсової роботи	Червень 2022 р.	

Студент Пінкевич В.М.

Керівник Борозенний С.О.

Зміст

Анотація

У роботі розглядаються базові принципи організації навчального процесу в школах та аналізуються можливості переведення частини цього процесу в електронну форму. Розроблюється веб-застосунок, який має спростити організацію навчального процесу. Описано загальну архітектуру застосунку, повний процес його розробки, використані інструменти та можливості його подальшого розширення та покращення.

Використані скорочення

СКБД – система керування базами даних.

REST - Representational State Transfer.

HTTP - HyperText Transfer Protocol.

API - Application Programming Interface.

JSON - JavaScript Object Notation.

XML - eXtensible Markup Language.

DI – Dependency Injection.

IoC - Inversion of Control.

JDBC - Java Database Connectivity.

JPA - Java Persistence API.

ORM - Object-Relational Mapping.

JWT - JSON Web Token.

MVC - Model-View-Controller.

IDE - Integrated development environment.

URL - Uniform Resource Locator.

HQL - Hibernate Query Language.

SQL - Structured Query Language.

DTO - Data Transfer Object.

HTML - HyperText Markup Language.

CSS - Cascading Style Sheets.

DOM - Document Object Model.

JSX - JavaScript Syntax Extension.

SPA - Single-Page Application.

Вступ

Основи навчального процесу сучасних шкіл є давно відомими. Вони стабільні і досить нечасто змінюються. З одного боку, така стабільність є доказом надійності цієї системи, адже вона існує уже не один і не два десятки років. Проте, надійність системи ніяк не пов’язана зі зручністю користування нею. На жаль, сучасні школи все ще досить консервативні, вони не поспішають впроваджувати різні електронні рішення, які можуть значно покращити зручність навчального процесу для всіх його учасників. Наприклад, у більшості шкіл учні мають щотижня заповнювати щоденник, переписуючи один і той самий розклад занять знову і знову. Також, учні часто не можуть дізнатися усі свої оцінки з певного предмету, для цього їм треба підходити до вчителя та запитувати про свої бали у нього. Крім того, батькам досить важко контролювати успішність своїх дітей. Адже єдиний спосіб, який у них є – комунікація з вчителями чи класним керівником напряму. Саме для усунення таких банальних незручностей і було вирішено розробити електронну систему, що і є метою даної курсової роботи.

Текстова частина курсової роботи складається із чотирьох розділів.

У першому розділі проводиться аналіз предметної області, короткий огляд існуючих систем зі схожим функціоналом. В кінці розділу описуються технічні можливості застосунку, а також категорії користувачів системи та їх вимоги.

Другий розділ присвячений проектуванню моделі предметної області. Спочатку описується ER-модель, розроблена на основі аналізу предметної області, проведеного у першому розділі, а потім на її основі будується реляційна модель, яка буде застосована уже при розробці застосунку для побудови бази даних.

У третьому розділі описується процес розробки застосунку. Спочатку розглядається загальна архітектура застосунку, а потім окремо розглядаються

процеси побудови клієнтської та серверної частин, використані інструменти. Для кожної частини описано її власну архітектуру, основний підхід до розробки, його переваги та недоліки, принципи та методи розробки, а також окремі деталі реалізації.

У четвертому розділі наводиться інструкція користувача та описуються можливості, які надає застосунок. Також, робиться висновок щодо результатів розробки системи, аналізуються подальші можливості її розвитку та покращення.

Розділ 1. Аналіз предметної області

1.1 Загальна характеристика

Шкільний учень належить до певного класу. Цей клас має певний список предметів, які вивчає кожен його учень. Кожен клас має розклад уроків. У розкладі зазначено номер уроку, час його проведення, предмет, та вчителя. Якщо у класі багато учнів, то він може поділятися на групи (для проведення занять із деяких предметів). Відповідно, для кожної групи є свій розклад, оскільки один і той самий предмет у різних груп може викладати різний вчитель. Також, із кожного предмета учень отримує оцінки.

Вчитель викладає предмети у певних класах. Відповідно, кожен вчитель має власний розклад уроків. Один вчитель може викладати декілька різних предметів у різних класах, або навіть в одному класі. Учитель контролює присутність учнів на уроках, і на кожному уроці відмічає присутніх та відсутніх учнів. За кожен урок вчитель виставляє учням оцінки у класний журнал. Оцінки бувають двох видів – поточні і тематичні. Поточні оцінки виставляються учневі за роботу на уроці: відповідь біля дошки, активність на уроці, тощо. Зрозуміло, що якщо учень не був присутнім на уроці, він не може отримати поточну оцінку за цей урок. Тематичні оцінки – оцінки, які виставляються за певні види робіт – контрольні роботи, самостійні роботи та інші. Тобто, основна задача вчителя, крім викладання – ведення класних журналів.

Класний керівник – вчитель, який відповідає за певний шкільний клас. Він може не викладати у цьому класі, але має усю інформацію про нього. Класний керівник відповідає за всіх учнів свого класу, за кожну групу класу, усі предмети, які викладаються цьому класу, розклад класу. Крім цього, для класного керівника актуальними є усі задачі та можливості вчителя, описані вище. Також він має деякі інші функції, які виходять за межі системи, що розробляється.

Заступник директора відповідає за організаційну частину навчального процесу. Він розподіляє учнів по класам та групам, може переводити учнів між класами та групами. Крім того, заступник директора створює та редагує розклад для класів та вчителів. Заступник директора може бути також вчителем та класним керівником, тоді у нього з’являються відповідні задачі та можливості.

1.2 Огляд існуючих рішень

Спроби хоча б частково перемістити освіту у цифрову форму були не раз, але жодна з них не була настільки успішною, щоб стати масовим явищем. В основному, усі знайдені електронні рішення спрямовані на здобуття освіти у дистанційній формі.

Наприклад, Google Classroom – безкоштовний веб-сервіс, метою якого є розвиток та використання електронних ресурсів для проведення уроків та організації різних видів робіт: домашніх, самостійних, контрольних. Основна його функція – спрощення обміну файлами між учнями та вчителями шляхом централізації в одному місці, замість використання електронної пошти чи месенджерів.

Moodle – веб-сервіс, система керування курсами. Вона призначена для створення сайтів для онлайн-навчання. На основі Moodle побудовано багато сайтів для онлайн-навчання. Зокрема, Moodle дуже популярний серед університетів по всьому світу.

САЗ – система автоматизованого запису на дисципліни, яка використовується у НаУКМА. Вона дозволяє переглядати список усіх доступних в університеті дисциплін, проводити запис на дисципліни. Також, у САЗі можна побачити індивідуальний план навчання студента – усі дисципліни, які він опановує протягом навчального року, та розклад занять для усіх спеціальностей.

Тобто, серед існуючих рішень немає жодного, яке б повністю задовольняло вимогам предметної області. Усі вони мають власну специфіку та орієнтовані більше саме на процес дистанційного навчання, ніж на цифровізацію шкіл.

1.3 Опис технічних вимог та можливостей системи

На основі проведеного вище аналізу предметної області та огляду існуючих рішень, можна виділити чотири основні категорії користувачів системи: учень, вчитель, класний керівник та заступник директора. Причому, ці категорії не є виключними, тобто класний керівник може бути і вчителем, а заступник директора – і вчителем, і класним керівником.

Учень має мати такі можливості:

- переглядати інформацію про власний клас – назву класу, класного керівника, список учнів класу, список учнів кожної групи свого класу;
- бачити свій тижневий розклад – номер та час уроку, предмет та інформацію про вчителя;
- переглядати список предметів, які він вивчає, та інформацію про вчителя для кожного предмету;
- переглядати свої поточні та тематичні оцінки з кожного предмету.

Вчитель має мати такі можливості:

- переглядати інформацію про предмети, які вчитель викладає – для кожного предмету бачити класи та групи, для яких цей предмет викладається;
- бачити свій тижневий розклад – номер та час уроку, предмет, клас та групу;

- переглядати журнал оцінок для кожного класу, групи та предмету, які викладає вчитель;
- додавати та видаляти теми у журналі оцінок;
- додавати та видаляти дати у журналі оцінок;
- відмічати присутність учнів у журналі оцінок;
- виставляти поточні оцінки учням;
- виставляти тематичні оцінки учням.

Класний керівник має мати такі можливості:

- бачити повну інформацію про учнів свого класу – список учнів класу та список учнів кожної групи свого класу;
- бачити інформацію про предмети, які викладаються класу чи його групи, а також інформацію про вчителів, які викладають ці предмети;
- переглядати тижневий розклад свого класу - номер та час уроку, предмет, групу класу та інформацію про вчителя;
- переглядати журнал оцінок свого класу – для кожного предмету, який вивчається класом, бачити оцінки кожного учня за кожну тему та кожну поточну оцінку;
- якщо класний керівник є також вчителем – він має мати усі можливості вчителя.

Заступник директора має мати такі можливості:

- переглядати інформацію про учнів будь-якого класу – список усіх учнів та список учнів кожної групи класу;
- бачити тижневий розклад кожного класу - номер та час уроку, предмет, групу класу та інформацію про вчителя;
- додавати та видаляти варіації уроків (наприклад, перший урок із 8.10 до 8.55 або інший варіант – із 8.20 до 9.05);

- додавати та видаляти записи із розкладу для кожного класу;
- додавати та видаляти шкільні класи;
- додавати та видаляти групи для кожного класу;
- розподіляти студентів по класам та групам;
- якщо заступник директора є також вчителем або класним керівником – він має мати усі можливості вчителя або класного керівника відповідно.

Крім описаних вище можливостей користувачів, існує ще декілька вимог до застунку:

- наявність автентифікації користувачів – оскільки система є закритою, до неї мають мати доступ лише деякі користувачі – учні та персонал певної школи;
- наявність авторизації користувачів – різні користувачі системи мають різні можливості, тому треба подбати про надання користувачу мінімально необхідних йому прав для повноцінного використання застосунку згідно з його роллю.

Розділ 2. Проектування моделі предметної області

2.1 Побудова ER-моделі предметної області

ER-модель – модель даних, яка дозволяє описати певну предметну область. Вона використовується при проектуванні баз даних. Основною перевагою створення ER-моделі є можливість виділити основні сутності предметної області та зв’язки між ними. [1]

На основі проведеного у розділі 1 аналізу предметної області можемо виділити такі основні сутності: учень, вчитель, шкільний клас, група класу, вчитель, шкільний предмет, урок, розклад, журнал оцінок, тема журналу оцінок, дата у журналі оцінок, тематична оцінка, поточна оцінка. Побудовану ER-модель можна побачити на рисунку 2.1.1:

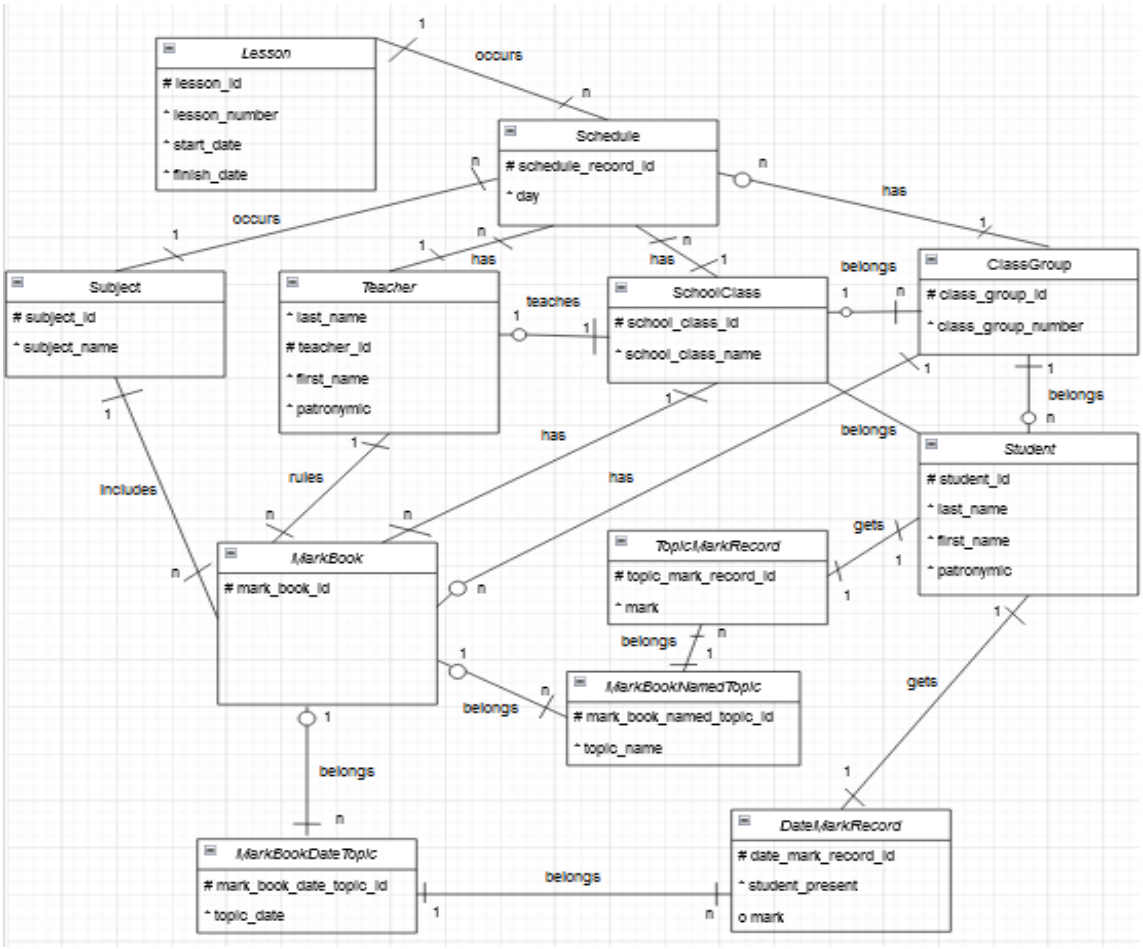


Рисунок 2.1.1. ER-модель застосунку

Сутність MarkBook у ER-моделі не має ніяких атрибутів, крім ключа, тому може здатись, що вона є зайвою. Але без неї потрібно буде дублювати значну кількість з’єднань для сутностей MarkBookNamedTopic та MarkBookDateTopic. Більше того, з ними буде важче працювати, оскільки не буде окремої сутності для класного журналу.

2.2 Побудова реляційної моделі предметної області

Реляційна модель даних – модель даних, у якій дані зберігаються у таблицях, які називаються відношеннями, кожна з яких складається із рядків, які називаються кортежами, та стовпчиків, які називаються атрибутами. Кожен атрибут має певний тип, а кожен кортеж описує одну сутність і є унікальним. [2] Перевагами реляційної моделі є простота її розуміння, простота доступу до даних та зручне зберігання відношень між реляціями.

При переході від ER-моделі до реляційної моделі структури сутностей та зв’язки між сутностями було повністю збережено. Побудовану реляційну модель (реалізовану у СКБД PostgreSQL) можна побачити на рисунку 2.2.1. Проте у реляційній моделі з’явились деякі нові сутності (principal, role, principal_role principal_jwt), яких не було у ER-моделі, оскільки вони відносяться уже до логіки роботи застосунку (а саме - до авторизації та автентифікації користувачів) і не є частиною предметної області.

2.3 Вибір СКБД

Для збереження даних у веб-застосунках найчастіше використовуються реляційні бази даних. Проте база даних не може використовуватись самостійно, для роботи з нею потрібно використовувати СКБД. СКБД – це програма, яка дозволяє створювати та редагувати бази даних, контролює доступ до баз даних та

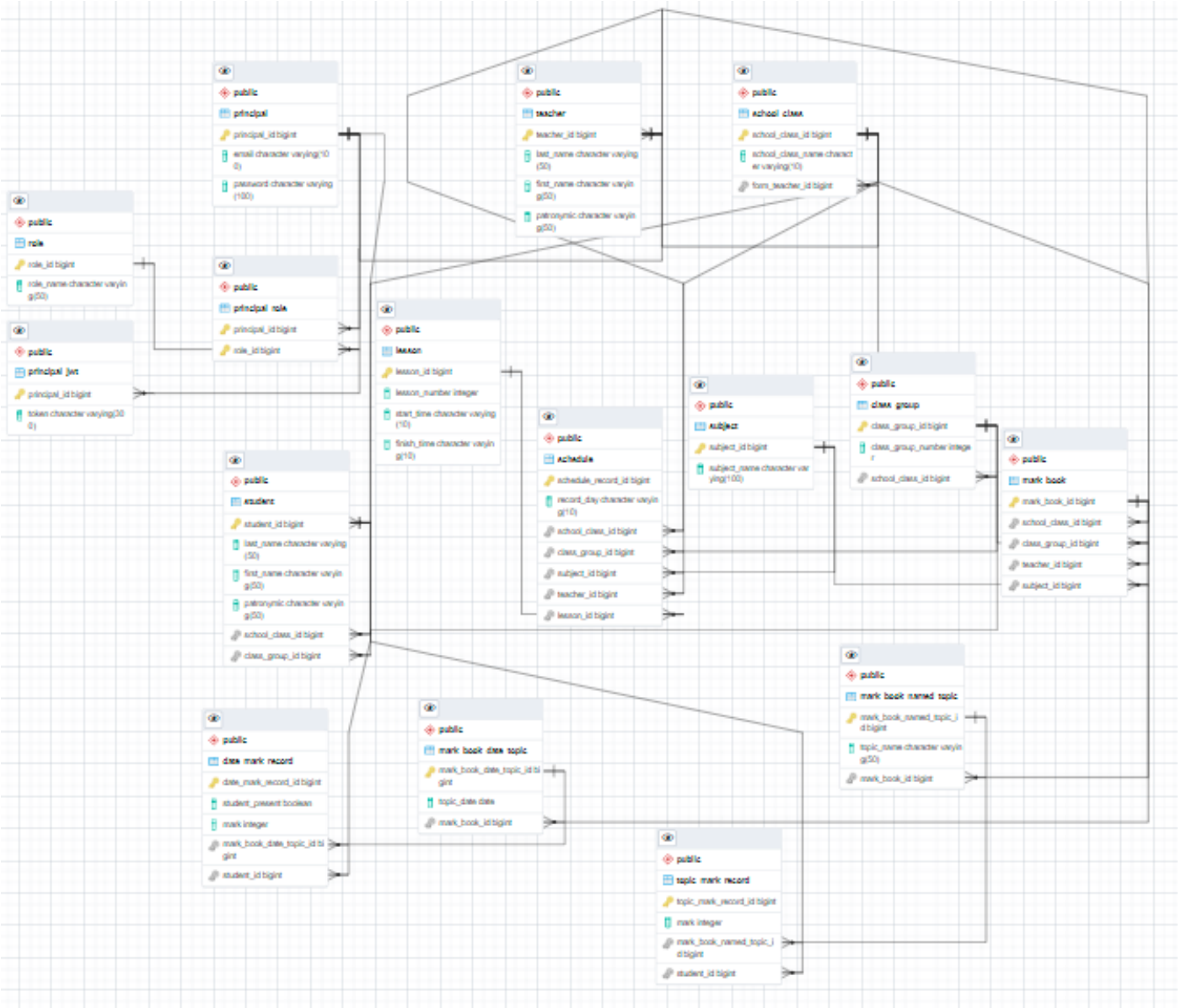


Рисунок 2.3.1. Схема бази даних

дозволяє здійснювати пошук інформації у базах даних. Існує великий вибір СКБД із різними можливостями та власними особливостями. Проте для розробки систему було обрано СКБД PostgreSQL. Вона має декілька значних переваг, а саме [3]:

- велика кількість стандартних вбудованих типів даних;
- висока продуктивність;
- хороша документація та легкість розробки;
- підтримка великої кількості стандартів мови SQL.

Розділ 3. Розробка застосунку

3.1 Опис загальної архітектури застосунку

3.1.1 Клієнт-серверна архітектура

Традиційно, веб-застосунки будують на основі клієнт-серверної архітектури, тому вона є базою для розробки системи.

Клієнт-серверна архітектура – модель побудови застосунків, у якій сервер відповідає за збереження, постачання та керування певними ресурсами, а клієнт робить запити до сервера на отримання ресурсу чи виконання певної дії із ресурсом. Зв'язок клієнт із сервером зазвичай відбувається через мережу. [4] Загальну ілюстрацію такої архітектури можна побачити на рисунку 3.1.1.1:

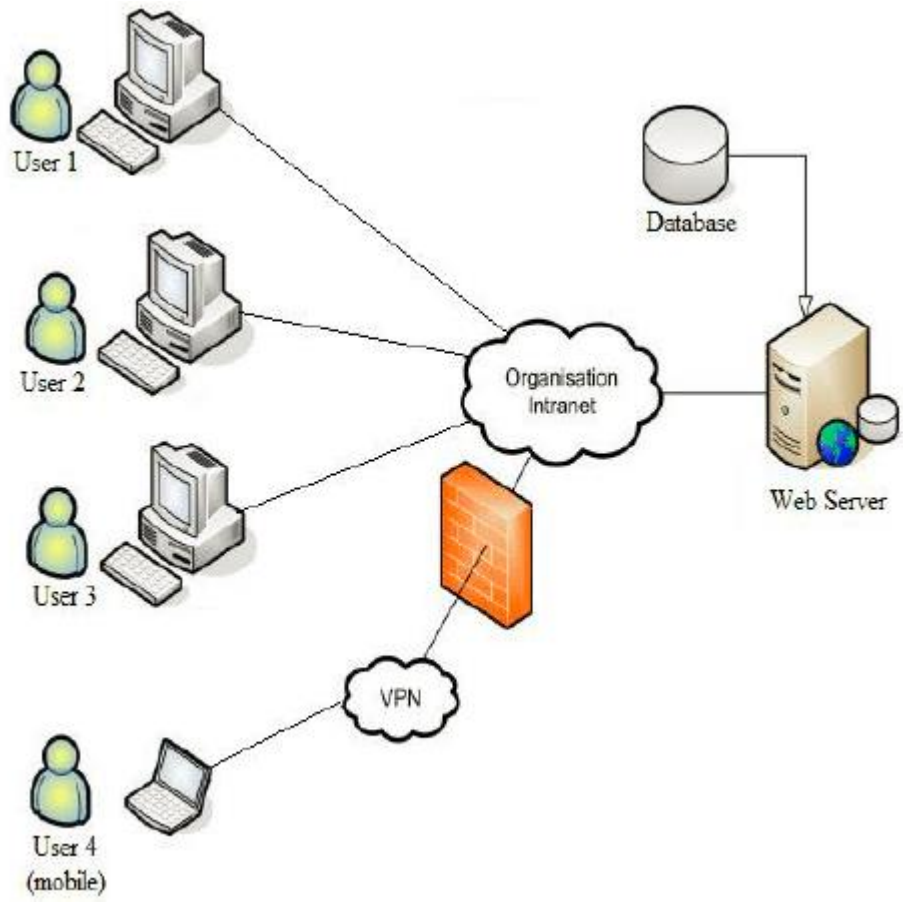


Рисунок 3.1.1.1 Ілюстрація клієнт-серверної архітектури

Зазвичай, власне веб-застосунок розміщується на сервері, дані зберігаються на окремому сервері, на якому встановлена СКБД. Клієнт комунікує із сервером надсилаючи запити і отримуючи відповіді від сервера.

Клієнт-серверна архітектура є досить поширеною через велику кількість переваг, серед них:

- принцип розділення обов’язків: клієнтська та серверна частини можуть бути повністю незалежними одна від одної;
- ізольованість клієнтів: клієнти нічого не знають один про одного, тому ніяк напряму не впливають один на одного;
- відмовостійкість: при клієнт-серверній архітектурі сервер можна легко перемістити на інший фізичний пристрій, а користувач не помітить жодних змін.

При розробці застосунку була використана дещо змінена версія клієнт-серверної архітектури. Оскільки клієнтська та серверна частини застосунку будуть розгорнуті на окремих серверах, то схема роботи застосунку буде такою: кінцевий користувач звертається до веб-сервера, який повертає йому html-сторінку, а вже при взаємодії із цією сторінкою клієнтська частина буде надсилати запити до окремого сервера, на якому розгорнута серверна частина застосунку. Це дозволяє уникнути тісного зв’язку між клієнтською та серверною частиною застосунку, оскільки вони розгортаються повністю незалежно одна від одної.

3.1.2 REST

Оскільки клієнтська та серверна частина застосунку розгортаються окремо, то комунікація між ними буде відбуватись по мережі за протоколом HTTP. Клієнтська частина відповідає за відображення даних для кінцевого користувача, а для отримання даних вона має звертатись до серверної частини. Для маніпуляції із

даними вона також має звертатись до сервера. Для такої комунікації при розробці застосунку використано архітектурний підхід, який називається REST.

REST – архітектурний підхід, який використовується при побудові клієнт-серверних систем. Основним принципом REST є відсутність збереження стану на сервері, тобто кожен запит може бути оброблений сервером самостійно та незалежно від попередніх запитів. Основним поняттям у REST є ресурс – будь-яка сукупність даних. Ресурсом також може бути колекція інших ресурсів. [5]

Маніпуляція ресурсами відбувається через обмін даними та відповідний тип HTTP-методу. Метод GET відповідає операції отримання ресурсу, POST – створення нового ресурсу, PUT або PATCH – за оновлення ресурсу, а DELETE – за видалення ресурсу. Обмін даними зазвичай відбувається у певному форматі – XML, JSON чи будь-якому іншому, проте наведені є найпоширенішими. На основі принципів REST будується API, з яким і взаємодіє клієнт. API, побудований згідно з принципами REST називається RESTful API. Приклад взаємодії клієнта та сервера з використанням REST-підходу можна побачити на рисунку 3.1.2.1:

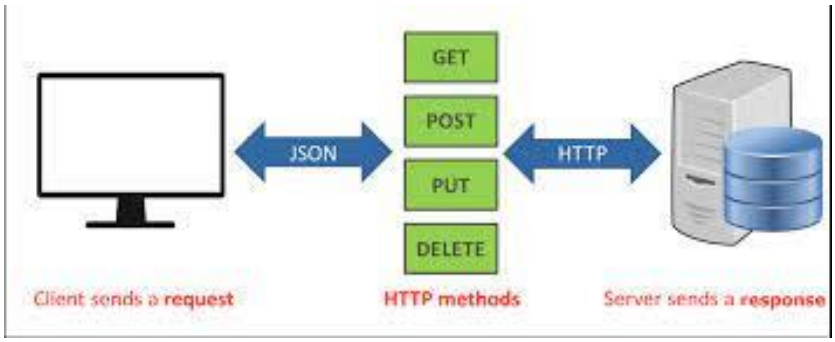


Рисунок 3.1.2.1 Взаємодія клієнта та сервера за REST-підходом

REST-підхід було обрано для розробки застосунку, оскільки він дозволяє розробити гнучкий API, який можна легко розширювати та змінювати при зміні функціоналу застосунку.

3.2 Розробка серверної частини

3.2.1. Інструменти розробки серверної частини

Для розробки серверної частини застосунку були використані мова програмування Java, фреймворк Spring, бібліотека Hibernate, бібліотека Java-JWT та програма для тестування API – Postman.

3.2.1.1 Spring

Spring – фреймворк для мови програмування Java, який використовується для розробки різних видів застосунків, серед них і веб-застосунки. Основними

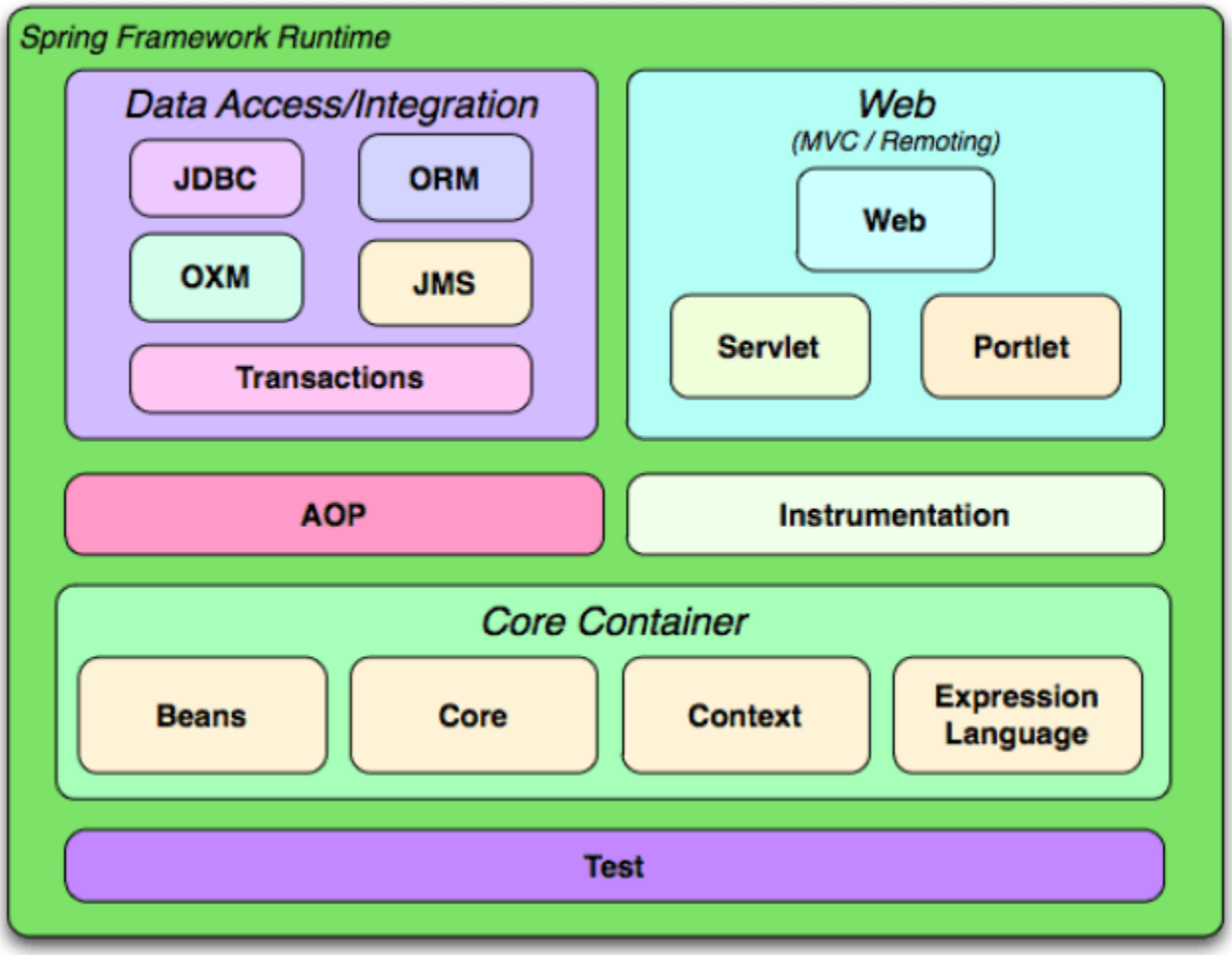


Рисунок 3.2.1.1.1. Основні частини фреймворку Spring

перевагами фреймворку є висока продуктивність, наявність великої кількості готових рішень для шаблонних завдань, гнучкість та модульність [6].

Модульність Spring є ключовою його складовою, оскільки дозволяє для кожної задачі обрати з великої кількості доступних модулів лише необхідні. Серед основних модулів Spring можна виділити:

- Spring Core – ядро фреймворку, яке дозволяє забезпечити роботу основних принципів Spring. Перший з них – DI – процес надання зовнішніх залежностей програмних компонентам, які їх потребують. [7] Наприклад, отримання об’єктом одного класу необхідного йому об’єкта іншого класу. Другий принцип – IoC – принцип побудови програми, при якому її частини отримують керування програмою із певного загального місця, яке також називають IoC контейнером. [7] Власне забезпечення DI та IoC контейнера є основною задачею Spring Core;
- Spring MVC та Spring WebFlux – модулі, які дозволяють будувати веб-застосунки і беруть на себе велику кількість рутинних задач. Spring MVC базується на класичному архітектурному підході – MVC, в той час як WebFlux – на новому, реактивному підході до побудови програм;
- Spring Data – модуль, який дозволяє забезпечити комфортну роботу із базами даних. Він підтримує велику кількість реляційних та нереляційних баз даних, Java JDBC – стандарт взаємодії Java-застосунків із базами даних та Java JPA – реалізація концепції ORM для Java, яка дозволяє в зручному вигляді встановлювати відповідність між Java-класами та реляціями баз даних;
- Spring Boot – модуль, який забезпечує простоту та легкість налаштування та розгортання застосунку. У класичному Spring проект потребує дуже великої кількості налаштувань, які часто є шаблонними, що не є комфортним. Spring Boot дозволяє вирішити цю проблему, оскільки

виконує велику кількість налаштувань автоматично, проте дозволяє виконувати глибоке налаштування застосунку, як і в Spring без Spring Boot;

- Spring Security – модуль, який надає велику кількість засобів для автентифікації, авторизації та обліку користувачів. Має велику кількість готових рішень і є зручною платформою для побудови захищених систем на базі Spring.

Також Spring має багато інших модулів, які використовуються для побудови розподілених та Cloud-застосунків, роботи із брокерами повідомлень, автоматизації, інтеграції зі сторонніми сервісами та інші, вузькоспеціалізовані модулі.

Для розробки веб-застосунку використовуються Spring Core, Spring MVC, Spring Security, Spring Data та Spring Boot, оскільки вони надають достатньо засобів для розробки застосунку згідно з описаними раніше вимогами та можливостями.

3.2.1.2 Hibernate

Hibernate – бібліотека для Java, яка реалізує концепцію ORM та дозволяє спростити роботу із базами даних, додаванням, редагуванням та видаленням об’єктів. Її основними перевагами є висока продуктивність, надійність та розширюваність та масштабованість. Вона також реалізує Java JPA, що дозволяє інтегрувати її в уже існуючі застосунки, які до цього використовували інші ORM. Hibernate бере на себе велику кількість однотипних задач, дозволяє уникнути повторень коду та легко додавати нові реляції до уже існуючих баз даних, без необхідності змінювати велику частину системи. [8] Основні принципи роботи

Java-застосунку із базами даних з використанням Hibernate можна побачити на рисунку 3.2.1.2.1:

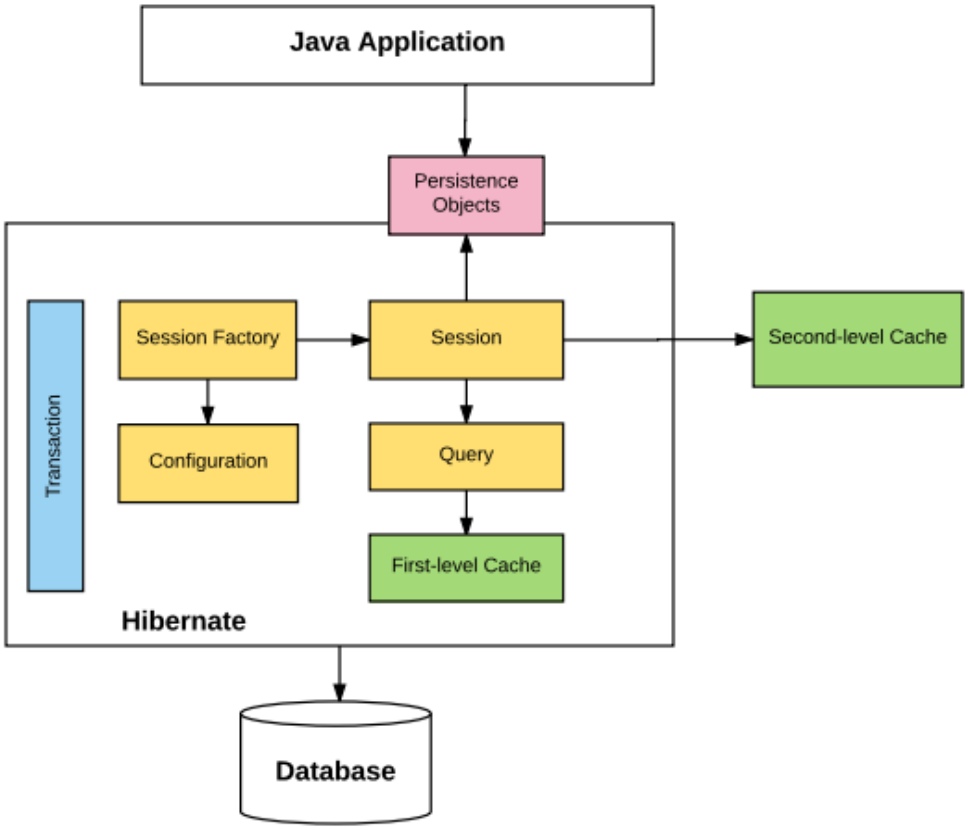


Рисунок 3.2.1.2.1. Взаємодія Java-застосунку із базами даних з використанням Hibernate

3.2.1.3 Java-JWT

Java-JWT – невелика бібліотека, яка реалізує стандарт JWT для мови Java. Вона надає зручний інструментарій для створення, валідації та роботи із JWT та може легко бути інтегрована у будь-який Java-застосунок. JWT буде описано пізніше, коли буде йтись мова про автентифікацію та авторизацію користувачів.

3.2.1.4 Postman

Postman – програма, яка дозволяє спростити процес побудови API. Postman спрощує розробку та супровід API, дозволяючи створювати, тестувати та покращувати API легко і швидко. [9] Postman має велику кількість функцій та можливостей, серед яких: написання API-запитів із повно підтримкою протоколу HTTP, можливістю контролю за кожною частиною створюваних запитів, написання скриптів та тестів для тестування API та створення колекцій API-запитів. Однією із переваг Postman є зручність при тестуванні RESTful API, тому його й було обрано в якості основного інструменту розробки та тестування API застосунку.

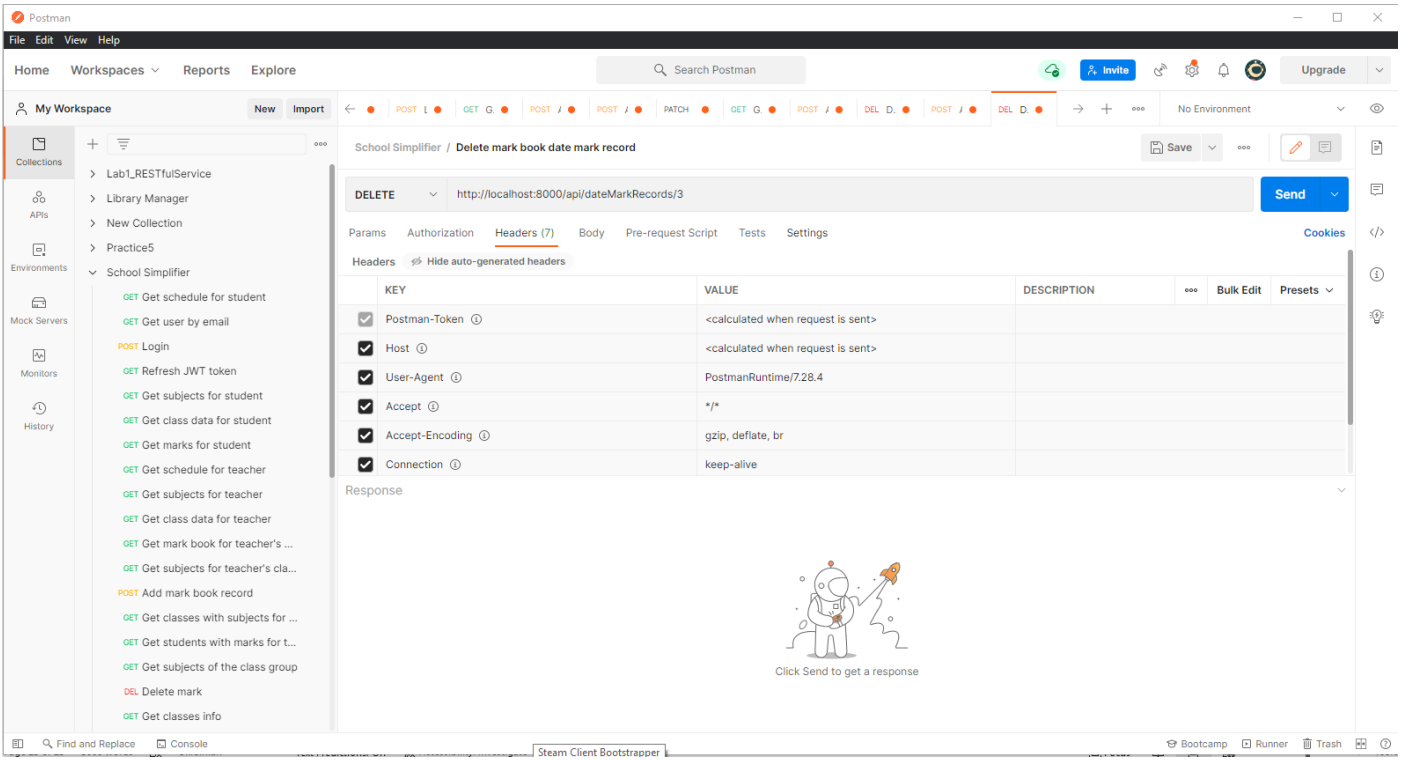


Рисунок 3.2.1.4.1. Інтерфейс програми Postman

3.2.2 Архітектура серверної частини

Класичним підходом до побудови застосунків, які передбачають взаємодію із користувачем, є архітектурний підхід MVC. Цей підхід передбачає розділення

застосунку на три взаємопов’язані частини – модель даних (model), представлення (view) та контролер (controller). Представлення відповідає за показ інформації користувачеві, модель – за збереження, додавання, модифікацію та будь-яку роботу із даними, а контролер – за передачу команд від користувача, які отримуються через представлення, до моделі, яка буде змінювати дані відповідно до отриманої команди. [10]

Основною метою використання цього підходу є створення гнучкої та масштабованої системи, компоненти якої є майже повністю незалежними один від одного. Модель є незалежною від представлення, оскільки реагує лише на команди, які отримує від контролера, а представлення, в свою чергу, не залежить від моделі, оскільки лише отримує команди і передає їх далі. Контролер виконує функцію моста між представленням та моделлю.

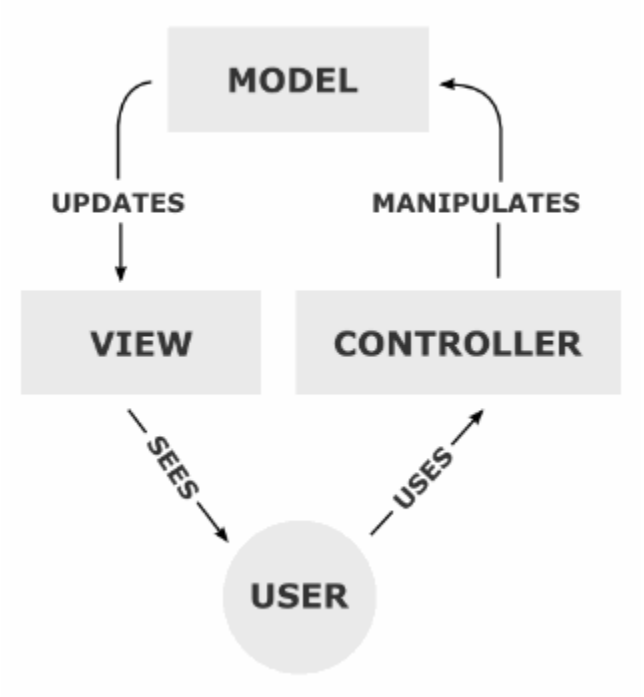


Рисунок 3.2.2.1 Ілюстрація підходу MVC

3.2.3 Реалізація серверної частини

3.2.3.1 Структура проекту

На рисунку 3.2.3.1.1 зображено файлову структуру проекту в IDE IntelliJ IDEA.

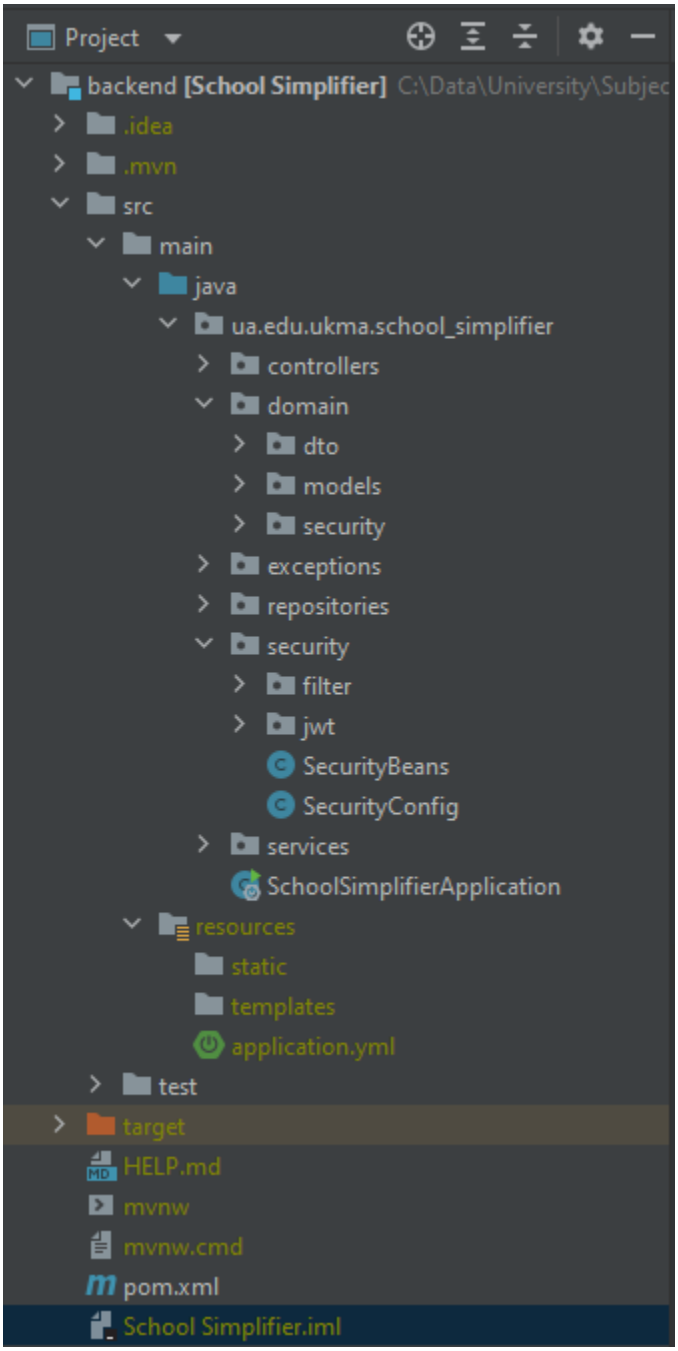


Рисунок 3.2.3.1.1 Файлова структура серверної частини проекту

Основними частинами, на які варто звернути увагу, є файл pom.xml та каталог src.

У файлі pom.xml знаходиться інформація про проект – версія Java, назва проекту, версія проекту, сторонні залежності (фреймворки та бібліотеки), які використовуються при розробці застосунку. Цей файл є основою для роботи Maven – програмного застосунку, який використовується для автоматизації збірки проектів.

У каталозі src знаходяться два основні підкаталоги – java та res. У каталозі res зберігається усі ресурси, які використовує застосунок – файли локалізації, шаблони html-сторінок та інші. Проте у цьому проекті важливим є лише один файл – application.yml, у якому знаходяться основні налаштування застосунку. Серед них – порт, на якому буде запущено сервер і посилання, логін та пароль для з’єднання із базою даних.

Вихідний код застосунку зберігається у каталозі java. Усі файли поділені за Java-пакетами – це механізм Java, який дозволяє групувати пов’язані файли із вихідним кодом в групи, щоб забезпечити логічне групування одиниць коду та уникнути проблеми накладання імен. Усі класи поділені за пакетами:

- controllers – тут зберігаються усі класи-контролери, які відповідають за відповідь на запити клієнта;
- domain – для класів, які представляють сутності предметної області;
- exceptions – тут зберігаються класи-виключення, які використовуються для повідомлення про помилки у роботі застосунку;
- repositores – для класів, які відповідають за доступ та роботу із базою даних;
- security – для компонентів, які відповідають за безпеку застосунку;
- services – тут зберігаються класи, які представляють основну бізнес-логіку застосунку;

- SchoolSimplifierApplication.java – файл, який знаходиться у кореневому пакеті ua.edu.ukma.school_simplifier і є точкою старту серверної частини застосунку.

3.2.3.2 Безпека застосунку

Основними принципами розробки захищеного застосунку є автентифікація користувачів та забезпечення надійного зберігання конфіденційної інформації користувачів. Також, оскільки у застосунку різні користувачі мають різні можливості, потрібно забезпечити авторизацію користувачів, щоб кожен користувач міг виконувати лише ті дії, які дозволено системою.

Першим кроком для забезпечення захисту системи є зберігання паролів у базі даних у захищеному вигляді. Для цього використовуються алгоритми шифрування, які перетворюють вхідний набір символів на інший набір символів. При розробці застосунку для шифрування був використаний алгоритм bcrypt, який є частиною Spring Security. Його перевагою є надійність та високий рівень захисту від класичних видів атак.

Для реалізації автентифікації та авторизації користувачів використовується JWT – це стандарт токена доступу на основі JSON. Він часто використовується для передачі даних та автентифікації у клієнт-серверних застосунках. Токени створюються сервером, підписуються секретним ключем і передається клієнту, який надалі використовує його для підтвердження своєї особи. [11] Зазвичай, для передачі токена від клієнта до сервера використовуються HTTP headers (заголовки протоколу HTTP) або Cookie. Кожен токен активний лише протягом певного періоду часу, який зберігається у самому токені і визначається сервером. Після закінчення цього періоду часу токен перестає прийматись сервером і користувач знову має пройти автентифікацію. Зазвичай, для уникнення необхідності

користувачу часто проходити автентифікацію використовуються 2 токени – access та refresh токени. Access токен має короткий термін активності тому потребує частого оновлення. Для його оновлення використовується refresh token – він надсилається клієнту при автентифікації разом із access токеном, проте має значно довший термін активності. Коли термін активності access токена завершується, клієнт надсилає на сервер refresh токен і отримує нову пару токенів. Перевагою JWT є зручність при роботі з RESTful API та можливість передавати додаткову інформацію про користувача. Ця можливість використовується у застосунку для авторизації користувача.

Авторизація користувачів побудована на основі ролей. Роль визначає певний перелік можливостей, які має користувач. Користувач може мати як одну, так і декілька ролей. У RESTful API роль визначає список ресурсів та дій з ними, які може виконувати користувач. Для передачі інформації про ролі користувача використовується поле roles у токені, в якому зберігається масив ролей користувача.

Для реалізації автентифікації користувачів використовується Spring Security. У ***Додатку А*** зображено клас SecurityConfig, який є основою для налаштування модуля Spring Security. Варто звернути увагу на метод configure(AuthenticationManagerBuilder), який дозволяє налаштувати об’єкт AuthenticationManager, який відповідає за автентифікацію користувачів. Йому для роботи надаються UserDetailsService – це інтерфейс, який має лише один метод loadUserByUsername(String username) для отримання користувача та BCryptPasswordEncoder – клас, який відповідає за шифрування паролів. Клас PrincipalServiceImpl реалізує інтерфейс UserDetailsService та завантажує інформацію про користувача із бази даних. Для автентифікації клієнт звертається до сервера за URL “/api/login”. Клас AuthenticationController отримує надісланий клієнтом логін та пароль, звертається до класу AuthenticationManager для

автентифікації. Якщо процес був успішним – цей же контролер повертає користувачу access та refresh токени.

Для забезпечення авторизації користувачів також використовується Spring Security. У класі SecurityConfig (***Додаток А***), у методі configure(HttpSecurity http) для кожного URL та кожного HTTP-метода можна задати дозволені ролі, з якими користувач може отримати доступ до ресурсу. Для отримання ролей використовується клас JwtAuthorizationFilter – це клас, який має один метод doFilterInternal(). У цьому класу отримується надісланий клієнтом токен, з нього отримається інформація про логін та ролі користувача та записуються у SecurityContext. На основі отриманих ролей та заданих у класі SecurityConfig налаштувань дозволених ролей для кожного URL відбувається надання доступу або заборона доступу до ресурсу.

3.2.3.3 Принципи роботи серверної частини застосунку

Сутності предметної області зберігаються у підпакеті models пакета domain. Ці сутності – звичайні Java-класи, які мають певні анотації для забезпечення роботи ORM, у даному випадку - Hibernate. У ***Додатку Б*** наведено приклад класу SchoolClass, який відповідає реляції school_class у базі даних. Клас має анотацію @Entity – вона означає, що клас є сутністю, яка зберігається у базі даних. Анотація @Table з атрибутом name вказує, якій таблиці у базі даних відповідає клас. Якщо поле класу є звичайним атрибутом у базі даних, воно позначається анотацією @Column із атрибутом name, який вказує, якій колонці у базі даних відповідає поле. Поле, яка є первинним ключем позначається анотацією @Id. Якщо первинний ключ генерується базою даних, то поле позначається анотацією @GeneratedValue, а атрибут strategy вказує на принцип генерації первинного ключа. Однією із найбільших переваг ORM є можливість отримувати із бази даних

для сутності не зовнішні ключі, які посилаються на інші сутності, а одразу об’єкти інших сутностей. Це забезпечується анотаціями, які вказують вид зв’язку між сутностями: @OneToOne, @OneToMany, @ManyToOne, @ManyToMany. Причому, зв’язок може бути як односторонній, так і двосторонній, тобто об’єкти можуть посилатись один на одного. Анотація @JoinColumn дозволяє вказати, за якою колонкою таблиці бази даних треба виконувати пошук пов’язаних сутностей. За допомогою усіх наведених вище анотацій Hibernate складає запити, які треба виконувати для роботи із об’єктами.

Для організації роботи застосунку використовується підхід controller-service-repository (контролер-сервіс-репозиторій). Він дозволяє розділити логіку отримання та відповіді на запити клієнта, виконання бізнес-логіки застосунку та отримання даних із бази даних, зробивши ці процеси незалежними один від одного. Відповідно, за виконання кожного етапу відповідає окремий об’єкт – контролер, сервіс, чи репозиторій. [12]

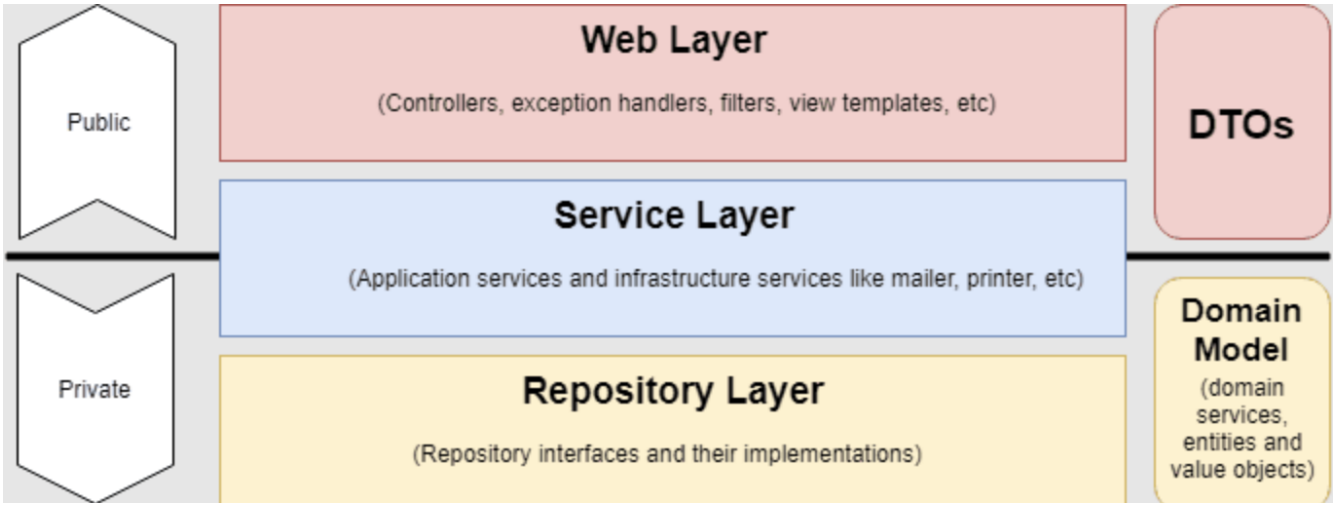


Рисунок 3.2.3.3.1. Ілюстрація підходу controller-service-repository

Репозиторій – об’єкт, який відповідає за отримання даних із бази даних. Він працює із об’єктами, які представляють сутності предметної області. Він відповідає за їх додавання у базу даних, оновлення, отримання та видалення.

Часто, для кожної сутності існує окремий репозиторій, у якому зосереджені методи для роботи саме із об’єктами цієї сутності.

При розробці застосунку для створення репозиторіїв використовується модуль Spring Data. Приклад Spring Data репозиторію можна побачити у ***Додатку В***. Неважко помітити, що це не клас, а інтерфейс, який наслідується від інтерфейсу JpaRepository. Spring Data надає декілька стандартних видів репозиторіїв, які забезпечують найпростіші функції – пошук об’єкта за ідентифікатором, додавання нового об’єкта та видалення об’єкта, але деякі репозиторії розширюють ці можливості, наприклад, для забезпечення сортування чи пагінації. На прикладі репозиторію із ***Додатку В*** можна побачити анотацію @Query, а в ній код, схожий на SQL. Це HQL – мова запитів Hibernate, яка дозволяє писати запити, базуючись Java-класах, які відображають сутності предметної області і конфігуруються за допомогою анотацій. Hibernate перетворює запити із HQL в SQL і використовує їх для звернення до бази даних. Проте, у репозиторіях також можна писати запити і мовою SQL, але тоді розробнику треба буде подбати про конвертацію отриманих даних із запиту в об’єкти необхідних йому класів. Для кожного запиту можна передавати параметри за допомогою анотації @Param, у якій атрибут name вказує на ім’я, за яким треба буде звертатись до параметра при написанні запиту. При використанні Spring Data розробнику часто не треба писати реалізацію інтерфейсів, оскільки під’єднання до бази даних, перетворення запиту в SQL та конвертацію об’єктів виконує Spring Data та обрана бібліотека, яка реалізує ORM.

Сервіс – об’єкт, у якому розміщена основна бізнес-логіка застосунку. Він може отримувати, валідувати та повертати дані, і виконувати будь-які інші задачі. Зазвичай, сервіс отримує дані від репозиторіїв та комунікує з іншими сервісами для виконання бізнес-логіки. У ***Додатку Г*** можна побачити приклад класу-сервісу. Сервіси позначаються відповідною анотацією @Service, і вказують Spring, що ці класи можуть бути використані для DI. Анотація @Transactional вказує, що

всі методи сервісу виконуються як транзакції. Тобто, якщо в процесі виконання методу певний об’єкт був збережений у базу даних, але пізніше метод завершився аварійно, наприклад, виникло виключення, то зміни, внесені до бази даних, збережені не будуть.

У ***Додатку Г*** можна помітити, що сервіс часто повертає об’єкти класів, які не належать до предметної області. Замість них повертаються DTO – це Java-класи, які містять лише інформацію і не мають поведінки. Вони використовуються для розділення сутностей предметної області, які можуть мати велику кількість інформації. Такі класи допомагають отримати лише ту частину інформації, яка необхідна клієнту, щоб йому не потрібно було займатись отриманням необхідних даних від сутності. Сервіси часто отримують від репозиторіїв сутності предметної області, а результатом виконання методу повертають DTO. Також DTO часто використовуються при отриманні даних від клієнта контролером для передачі їх сервісу.

```
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class StudentSchoolClassDTO {

    private String schoolClassName;

    private TeacherSummaryDTO formTeacher;

    private List<StudentSummaryDTO> classStudents;
    private Map<Integer, List<StudentSummaryDTO>> groupStudents;
}
```

Рисунок 3.2.3.3.2 Приклад DTO

Контролер – об’єкт, який відповідає за отримання, обробку та відповідь на запити клієнта. Основні його функції – передача даних, отриманих від клієнта, до сервісів, отримання результатів виконання методів сервісів та повернення цих результатів клієнту. Принцип роботи контролера можна розглянути на прикладі ***Додатку Д***. Анотація @RestController вказує, що даний клас є веб-контролером, який повертає клієнту у відповідь не html-сторінку, а дані у певному форматі, наприклад, JSON чи XML. Ця анотація є більш вузькоспеціалізованою версією анотації @Controller, яка просто вказує, що клас є веб-контролером. Анотація @RequestMapping може застосовуватись на рівні класу та на рівні методу. На рівні методу вона вказує певний URL, запити за яким буде обробляти даний метод. На рівні класу вона вказує, що для будь-яких методів цього контролера, які обробляють запити клієнта, URL буде складатись із URL, вказаного на рівні класу + URL, вказаного на рівні методу. Існують варіації анотації @RequestMapping такі як @GetMapping, @PostMapping та інші, за іменами HTTP-методів. При використанні анотації @RequestMapping на рівні методу також треба вказувати, запити з якими HTTP-методами буде обробляти метод. А наведені варіації заміняють собою необхідність вказувати HTTP-методи для обробки запитів. Анотація @RequestBody вказує, що при отриманні запиту за цим URL Spring автоматично буде намагатись перетворити тіло HTTP-запиту на об’єкт зазначеного класу. Анотація @PathVariable дозволяє отримати змінну з URL, для цього треба зазначити ім’я змінної атрибутом name. Анотація @RequestParam дозволяє отримати HTTP-параметри з URL, для цього також треба зазначити ім’я параметру атрибутом name.

Значною перевагою Spring є те, що при роботі з RESTful API розробнику не треба вручну виконувати серіалізацію та десеріалізацію об’єктів у різні формати, оскільки Spring виконує її сам, розробнику достатньо повертати звичайні об’єкти Java-класів.

3.3 Розробка клієнтської частини

Клієнтську частину застосунку можна реалізувати декількома способами. Можна розробити комп’ютерний чи мобільний застосунок, проте, найкращим рішенням у даному випадку буде розробка веб-застосунку. По-перше, оскільки браузери існують для будь-якої операційної системи, то застосунок одразу стає доступним на багатьох платформах. По-друге, оскільки серверна частина застосунку побудована на основі REST-підходу, то можлива розробка комп’ютерного чи мобільного клієнта, оскільки клієнт лише отримує дані від сервера, а не представлення у вигляді html-сторінок, як, наприклад, при розробці веб-сайтів.

3.3.1 Інструменти розробки клієнтської частини

Для розробки клієнтської частини були використані мова розмітки HTML, мова для опису вигляду веб-сторінок CSS, мова програмування JavaScript, бібліотека React, бібліотека Redux та бібліотека React Router.

3.3.1.1 React

React – бібліотека для мови програмування JavaScript, основним призначенням якої є побудова зручного користувацького інтерфейсу. ЇЇ основними перевагами є декларативний стиль побудови інтерфейсу та використання маленьких частин, які можна використовувати багаторазово – компонентів. [13]

Основна ідея React – створення компонентів, кожен з яких є окремою функцією і має свій стан. React використовує Virtual DOM – спосіб представлення документ у вигляді дерева об’єктів, у якому зберігаються компоненти. Virtual DOM є копією реального DOM-дерева, яке обробляють браузери для відображення HTML-сторінок. Відповідно, при оновленні стану певного

компонента оновлюються відповідні вузли DOM-дерева, а потім ці зміни застосовуються до реального DOM.

Ще однією важливою частиною React є JSX – розширення синтаксису мови JavaScript, яке допомагає описувати React-компоненти. JSX за своїм виглядом не відрізняється від HTML, тому працювати із ним зручно будь-якому розробнику, який знайомий з HTML. JSX має можливості, які роблять побудову компонентів досить зручною. Серед них: умовне відображення – це механізм, який дозволяє показувати або ховати певні елементи на основі певних умов та можливість використовувати вирази мови JavaScript як елементи для відображення. Також, React-компоненти мають атрибути, які дозволяють робити їх досить гнучкими у використанні.

Третій важливий елемент React – React Hooks (хуки). Хуки – спеціальні функції, які використовуються для роботи зі станом компонентів. Різні хуки використовуються для обробки стану компонентів, для контролю за компонентами, з якими взаємодіє користувач, для збереження результатів певних важких обчислень, виконання побічних ефектів, тощо. React має стандартний набір хуків, проте розробник може створювати власні хуки.

3.3.1.2 Redux

Redux – JavaScript бібліотека, основна мета якої – забезпечити керований та стабільний контейнер стану застосунку. Її перевагами є передбачуваність та надійність стану застосунку, простота роботи та легкість інтеграції із будь-якими бібліотеками для побудови користувацьких інтерфейсів на JavaScript. [14]

Redux базується на ідеї view-action-state (представлення-дія-стан) та “one-way data flow” (потік даних рухається лише в одну сторону). Користувач взаємодіє

із представленням і генерує події. Ці події впливають на стан, а стан, в свою чергу, є джерелом інформації для представлення.

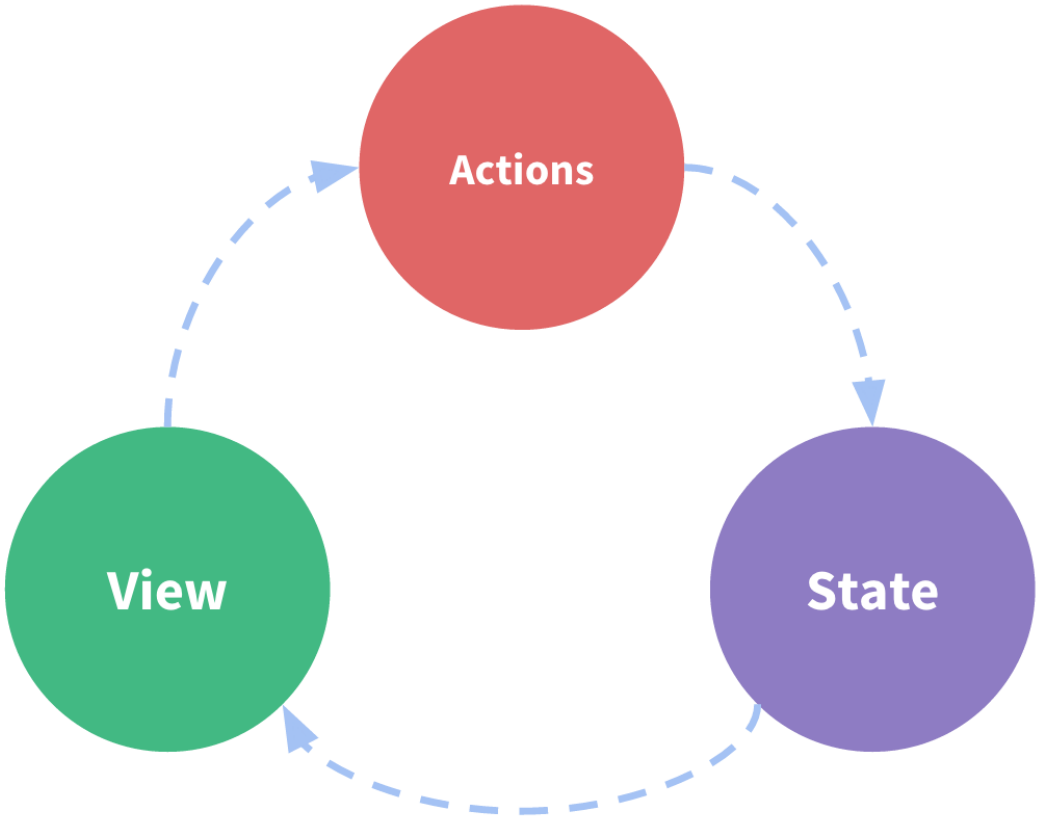


Рисунок 3.3.1.2.1. Ілюстрацію принципу view-action-state

Основним місцем збереження стану застосунку є store (сховище). Воно зберігає загальний стан усіх частин застосунку. Сховище має набір функцій, які називаються reducers (редуктори) і створюються розробником. Кожен редуктор отримує на вхід попередній стан застосунку та дію, яку він має застосувати. На основі цих дій редуктор виконує зміну стану застосунку. Робота зі станом відбувається через метод сховища dispatch(action), який приймає на вхід дію, яку хоче виконати користувач і передає її редукторам.

Також, для розробки клієнтською частини будуть використані ще дві невеликі бібліотеки: React-Redux, яка забезпечує інтеграцію Redux та React і

Redux-Toolkit, яка надає додаткові можливості для роботи з Redux та робить цю роботу зручнішою.

3.3.1.3 React Router

React Router – бібліотека, яка надає можливості для клієнтської маршрутизації у React застосунках. Вона невелика, проте має багато можливостей, серед яких: вкладені та відносні шляхи, історія відвідувань, URL-параметри та умовні шляхи. [15]

Основне призначення React Router – створення маршрутизації у SPA - односторінкових застосунках. Це дозволяє забезпечити можливість створення посилань на конкретну сторінку застосунку, як на сторінку веб-сайту, хоча весь застосунок і працює на одній html-сторінці, а також зручну історію відвідування сторінок, по якій можна рухатись назад або вперед.

3.3.2 Архітектура клієнтської частини

Для побудови веб-застосунку було обрано підхід SPA. SPA – принцип побудови веб-застосунків, коли увесь користувацький інтерфейс відображається лише на одній сторінці і змінюється шляхом динамічної зміни HTML, CSS та JavaScript. Сторінка ніколи не оновлюється, усі необхідні дані клієнт отримує із сервера у результаті певних дій користувача. Основна мета підходу SPA – наблизити браузерні застосунки до комп’ютерних чи мобільних. Зазвичай, застосунки, побудовані із застосуванням принципу SPA, працюють швидше аналогічних, побудованих на основі декількох сторінок, оскільки не потребують звертань до сервера кожен раз, коли треба завантажити нову сторінку. [16]

Підхід SPA досить добре поєднується із REST-підходом, оскільки REST-підхід передбачає лише отримання певних даних клієнтом та їх показ користувачеві.

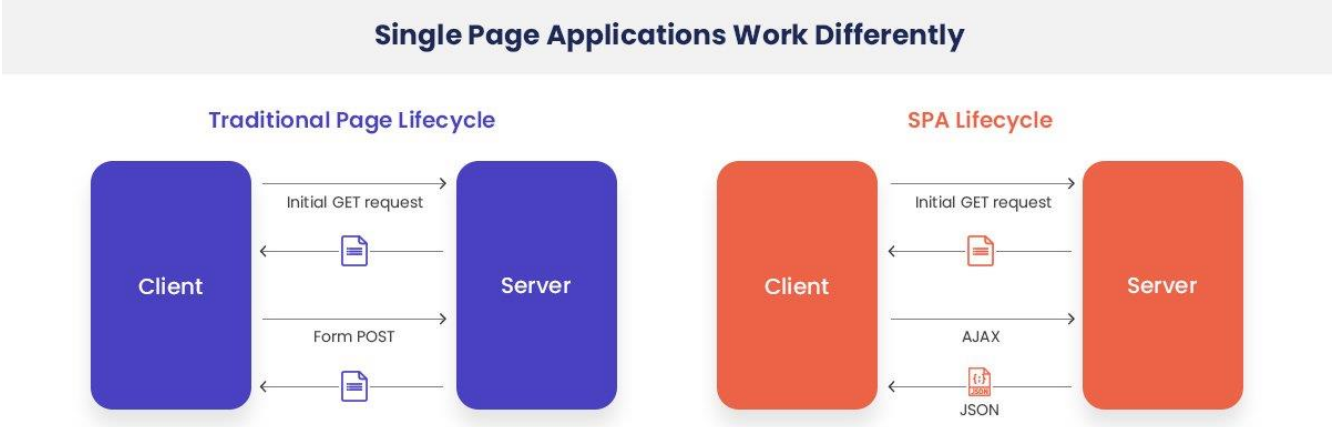


Рисунок 3.3.2.1 Порівняння традиційних веб-сайтів та SPA

3.3.3 Реалізація клієнтської частини застосунку

3.3.3.1 Структура проекту

На рисунку 3.3.3.1.1 зображено файлову структуру проекту в IDE Visual Studio Code. Основні файли та каталоги – package.json, public та src.

У файлі package.json зберігається інформація про застосунок – назва, версія та список залежностей (бібліотек), які необхідні для роботи застосунку. Цей файл створюється при виконанні команди “npx create-react-app” у менеджері пакетів npm. Ця команда завантажує шаблон звичайного React-застосунку та усі необхідні файли.

У каталозі public знаходиться html-сторінка, яка є основою для React-застосунку, та деякі ресурси – логотипи, інформація для пошукових роботів, тощо.

Файл index.js – точка старту React-застосунку, у якій відбувається рендеринг кореневого компоненту будь-якого React-застосунку – App.js.

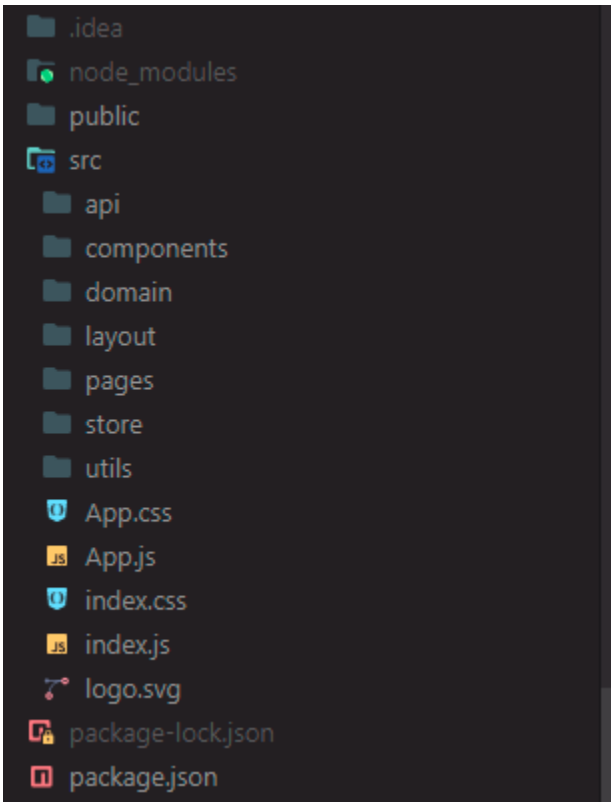


Рисунок 3.3.3.1.1. Файлова структура клієнтської частини проекту

Файл App.js – кореневий компонент, всередині якого будуть відображатись інші React-компоненти. У даному застосунку там знаходяться налаштування клієнтської маршрутизації, та загального шаблону веб-застосунку.

У каталозі src знаходиться вихідний код застосунку. Його вміст поділений на декілька підкаталогів:

- api – тут зберігаються файли, у яких містяться функції для надсилання запитів на сервер;
- components – містить усі React-компоненти, які були побудовані в ході розробки застосунку;
- domain – містить деякі константи, які відносяться до предметної області;
- layout – містить React-компоненти, які визначають шаблони сторінки;

- pages - містить React-компоненти, які можна вважати окремим сторінками у застосунку;
- store – містить конфігурацію та методи Redux-сховища;
- utils – містить різні допоміжні функції, які використовуються багатьма компонентами застосунку.

3.3.3.2 Принципи роботи клієнтської частини застосунку

Почнемо із огляду роботи зі станом застосунку. Для реалізації стану автентифікації користувача (автентифікований чи неавтентифікований) використовується Redux-сховище, частину якого, що відповідає за автентифікацію, можна побачити у ***Додатку Е***. Тут наявна назва цієї частини сховища - name, за якою до неї можна буде звертатись у застосунку, початковий стан – initial state, який визначає стан застосунку при старті та набір функцій редукторів – reducers, який визначає, які дії можна робити зі станом автентифікації. Редуктори дозволяють додати access та refresh токени, видалити їх та редуктор для завантаження токенів із локального сховища браузера.

Від того, автентифікований користувач чи ні, залежить, як буде виглядати інтерфейс та які шляхи будуть йому доступні. У ***Додатку Ж*** показано файл App.js, у якому визначаються шляхи та шаблони для автентифікованого користувача та неавтентифікованого – AuthLayout та PublicLayout відповідно. Неавтентифікованому користувачу доступні лише 2 сторінки – домашня сторінка та сторінка для автентифікації. Меню автентифікованого користувача залежить від його ролі. Для захисту певних шляхів від неавторизованих користувачів був розроблений компонент RequireAuth – йому можна передати список ролей, яким будуть доступні певні шляхи. Якщо користувач має потрібну роль – він буде

допущений до сторінки, якщо ні – йому буде показане повідомлення про відсутність доступу.

```
import { useSelector } from "react-redux";
import { Navigate, Outlet, useLocation } from "react-router-dom";
import Unauthorized from "../../pages/public/Unauthorized";

function RequireAuth({allowedRoles}) {
    const roles = useSelector(state => state.auth.roles);
    const location = useLocation();

    if(!roles) {
        return (
            <Navigate to='/login' state={{from: location}} replace />
        );
    }

    if(!roles.find(role => allowedRoles.includes(role))) {
        return (
            <Unauthorized />
        );
    }

    return (
        <Outlet />
    );
}

export default RequireAuth;
```

Рисунок 3.3.3.2.1. Компонент RequireAuth

Сторінка – це звичайний React-компонент, який є контейнером для інших React-компонентів. Основна задача сторінки – групування пов’язаних компонентів для їх відображення, отримання даних від сервера та їх передача іншим компонентам. Приклад сторінки можна побачити на рисунку 3.3.3.2.2.

Крім перегляду даних, деякі користувачі мають можливості для додавання чи редагування існуючих даних. Щоб забезпечити це на клієнтській частині

```jsx
import React, { useEffect, useState } from "react";
import { useSelector } from "react-redux";
import { getSubjectsForStudent } from "../../api/student";
import StudentSubjectsTable from "../../components/table/StudentSubjectsTable";

import classes from './StudentSubjects.module.css';

function StudentSubjects() {
    const accessToken = useSelector(state => state.auth.accessToken);
    const [subjects, setSubjects] = useState([]);

    useEffect(() => {
        const fetchData = async() => {
            try {
                const subjects = await getSubjectsForStudent(accessToken);
                setSubjects(subjects);
                console.log(JSON.stringify(subjects));
            }
            catch(er) {
                console.log(er);
            }
        }
        fetchData();
    }, [accessToken, setSubjects]);

    return (
        <div className={classes['page-container']}>
            <h2>Предмети</h2>
            <StudentSubjectsTable subjects={subjects} />
        </div>
    );
}

export default StudentSubjects;
```

Рисунок 3.3.3.2.2. Компонент StudentSubjects, який представляє сторінку, на якій учень бачить список своїх предметів

використовуються форми – вони надають розробнику елементи, які можна використовувати, наприклад, для введення даних із клавіатури. Таку форму можна зробити окремим React-компонентом і використати у декількох місцях застосунку. Зазвичай, форма лише отримує дані від користувача, проводить їх перевірку на правильність, а потім передає обробнику, який може використати ці дані для будь-якої мети: відправити запит на сервер, знайти якісь дані, що уже зберігаються на стороні клієнта, тощо. Така організація роботи з формами робить їх гнучкими та

дозволяє перевикористовувати їх. Приклад такої форми можна побачити на рисунку 3.3.3.2.3:

```
import { useRef, useState } from "react";

import classes from './AddMarkBookNamedTopicForm.module.css';

function AddMarkBookNamedTopicForm(props) {

    const enterNameRef = useRef();
    const [nameError, setNameError] = useState(false);
    const cantAddMarkBookNamedTopicError = props.cantAddMarkBookNamedTopicError;

    const submitFormHandler = (e) => {
        e.preventDefault();

        const enteredName = enterNameRef.current.value;
        const trimmedName = enteredName === null
                                ? null
                                : enteredName.trim();
        const validName = !!trimmedName;
        setNameError(!validName);

        if(validName) {
            props.onAddMarkBookNamedTopic(trimmedName);
        }
    }

    return (
        <form className={classes['form-layout']} onSubmit={submitFormHandler}>
            <label>Введіть назву теми:</label>
            <input type='text' required ref={enterNameRef}></input>
            {nameError && <p className={classes.error}>Назва теми не має бути порожньою</p>}

            <input type="submit" value="Додати тему" />
            {cantAddMarkBookNamedTopicError && <p className={classes.error}>{cantAddMarkBookNamedTopicError}</p>}
        </form>
    );
}

export default AddMarkBookNamedTopicForm;
```

Рисунок 3.3.3.2.3. Компонент AddMarkBookNamedTopicForm, який представляє форму, яку використовує вчитель для додавання теми до журналу оцінок

Для отримання даних від сервера використовується fetch API – це інтерфейс, який браузер надає JavaScript для отримання даних по мережі. Оскільки декілька компонентів можуть потребувати отримання одних і тих же даних, функції для доступу до API виносяться в окремі файли і вони можуть бути використані усіма компонентами, які їх потребують. При дотриманні REST-підходу, усі функції, які доступаються до певного ресурсу зазвичай поміщаються в один файл, оскільки

дозволяють зручно організувати логічно пов’язані одиниці коду. Приклад такого файлу із функціями можна побачити на рисунку 3.3.3.2.4:

```
import { DOMAIN_URL } from "../domain/constants";

const MARK_BOOK_NAMED_MARK_RECORDS = DOMAIN_URL + '/namedMarkRecords';

export async function addMarkBookNamedRecord(accessToken, studentId, markBookNamedTopicId, mark) {
    const response = await fetch(MARK_BOOK_NAMED_MARK_RECORDS, {
        method: 'POST',
        headers: {
            'Authorization': `Bearer ${accessToken}`,
            'Content-Type': 'application/json; charset=utf-8'
        },
        body: JSON.stringify({
            studentId,
            markBookNamedTopicId,
            mark
        })
    });
    if(!response.ok) {
        throw new Error("Mark book named mark record adding failed");
    }
}

export async function deleteMarkBookNamedTopicRecord(accessToken, markBookNamedTopicRecordId) {
    const response = await fetch(MARK_BOOK_NAMED_MARK_RECORDS + `/${markBookNamedTopicRecordId}`, {
        method: 'DELETE',
        headers: {
            'Authorization': `Bearer ${accessToken}`
        },
    });
    if(!response.ok) {
        throw new Error("Mark book named mark record deletion failed");
    }
}
```

Рисунок 3.3.3.2.4. Файл markBookNamedTopics.js, який містить функції, які відповідають за доступ до ресурсу "Тема журналу оцінок"

За описаними вище принципами та прикладами був розроблений користувацький інтерфейс, який надає користувачам усі можливості, описані при аналізі предметної області.

Розділ 4. Інструкція користувача

4.1 Можливості неавторизованого користувача

Коли користувач відкриває застосунок, він бачить вікно входу в систему. Для входу в систему йому потрібно ввести свої дані (електронну пошту та пароль) і натиснути кнопку “Увійти в систему”. Якщо введені дані правильні – користувач буде переміщений на певну сторінку, залежно від його ролі. Якщо ж дані неправильні – з’явиться повідомлення про неправильність введених даних.

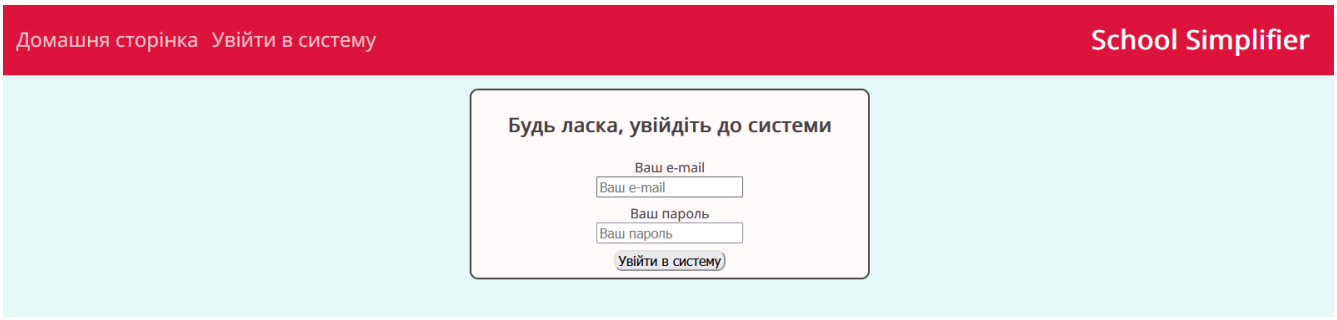


Рисунок 4.1.1. Сторінка входу в систему

Також, зі сторінки входу в систему користувач може перейти на домашню сторінку застосунку, використовуючи меню зверху. Вона не містить ніякої особливої інформації, лише запрошення користувачу для переходу на сторінку входу в систему.

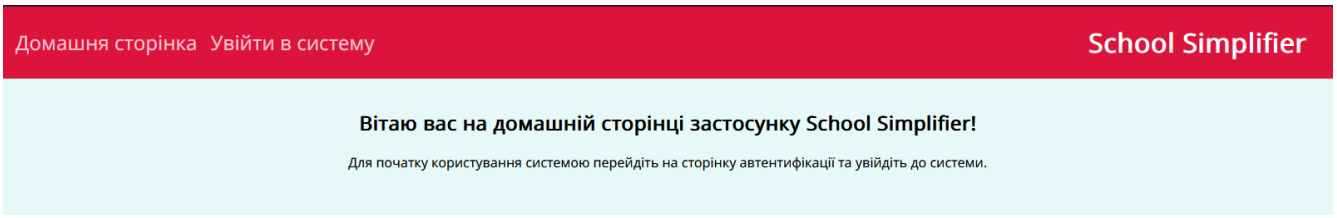


Рисунок 4.1.2. Домашня сторінка застосунку

4.1 Можливості учня

Після успішного входу в систему, учень потрапляє на сторінку зі своїм розкладом. На цій сторінці розміщено розклад на весь тиждень. Із цієї сторінки учень за допомогою меню зверху може перейти до списку своїх предметів, інформації про власний клас та власних оцінок.

Рисунок 4.2.1. Сторінка розкладу учня

На сторінці “Предмети” учень може побачити список предметів власного класу. Для кожного предмета вказано його назву, групу, яка його вивчає, та вчителя який викладає цей предмет.

Рисунок 4.2.2. Сторінка предметів учня

На сторінці “Клас” знаходиться інформація про усіх учнів класу учня та список учнів кожної групи класу. Також вказано назву класу та класного керівника.

Рисунок 4.2.3. Сторінка класу учня

На сторінці “Оцінки” знаходиться кнопка “Обрати предмет”, при натисненні якої з’являється форма, яка дозволяє учню обрати предмет, з якого він хоче побачити оцінки та групу, оскільки якщо предмет читається і для всього класу, і для кожної групи, то оцінки виставляються окремо за роботу на групових уроках та на уроках цілого класу. Після обрання учнем предмету та групи, на сторінці

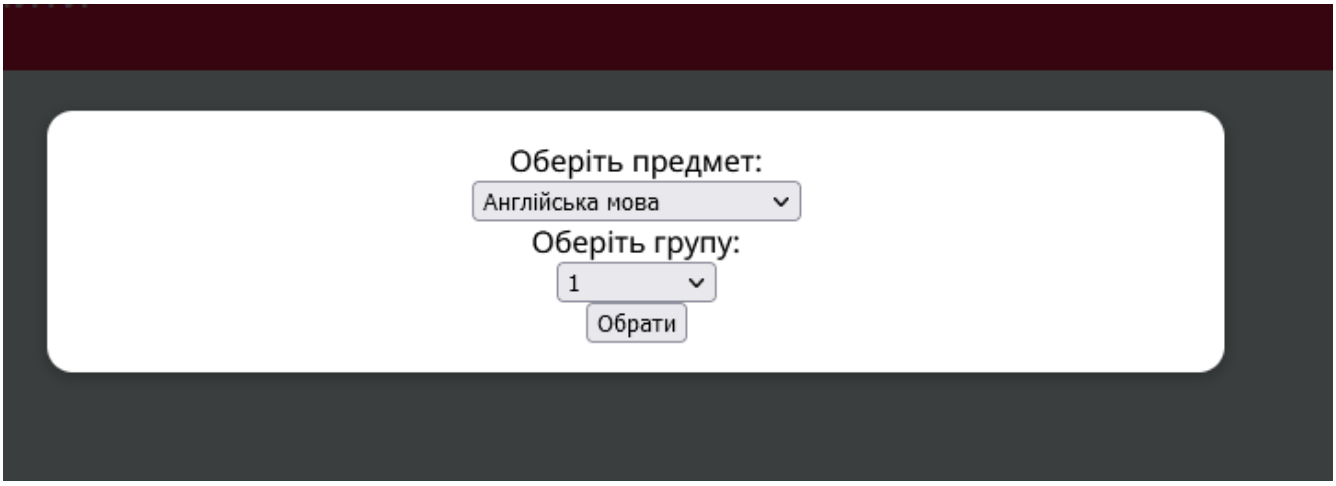


Рисунок 4.2.4. Форма для вибору учнем предмету та групи, для якої він хоче побачити оцінки

з’являється список його оцінок з предмету – окремо поточних та окремо тематичних.

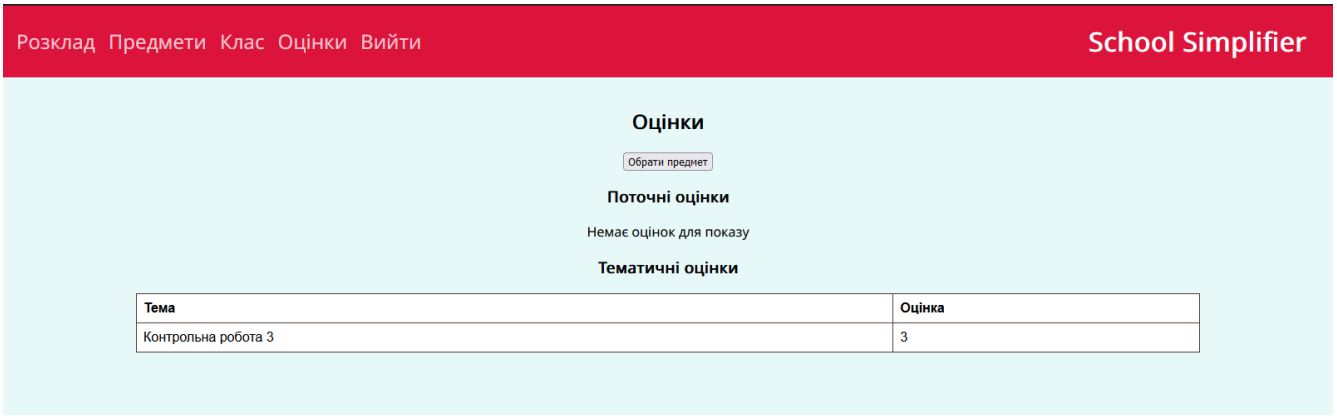


Рисунок 4.2.5. Сторінка оцінок учня після обрання предмету та групи

4.3 Можливості вчителя

Після успішного входу в систему, вчитель потрапляє на сторінку зі своїм розкладом. Він бачить номер та часу уроку, предмет, клас та групу, у якому проводиться урок. За допомогою меню зверху вчитель може перейти на сторінки “Предмети” та “Журнал оцінок” (не зазначені тут сторінки – це сторінки для класного керівника та заступника директора, проте вони відображені тут, оскільки обраний вчитель також є і класним керівником, і заступником директора).

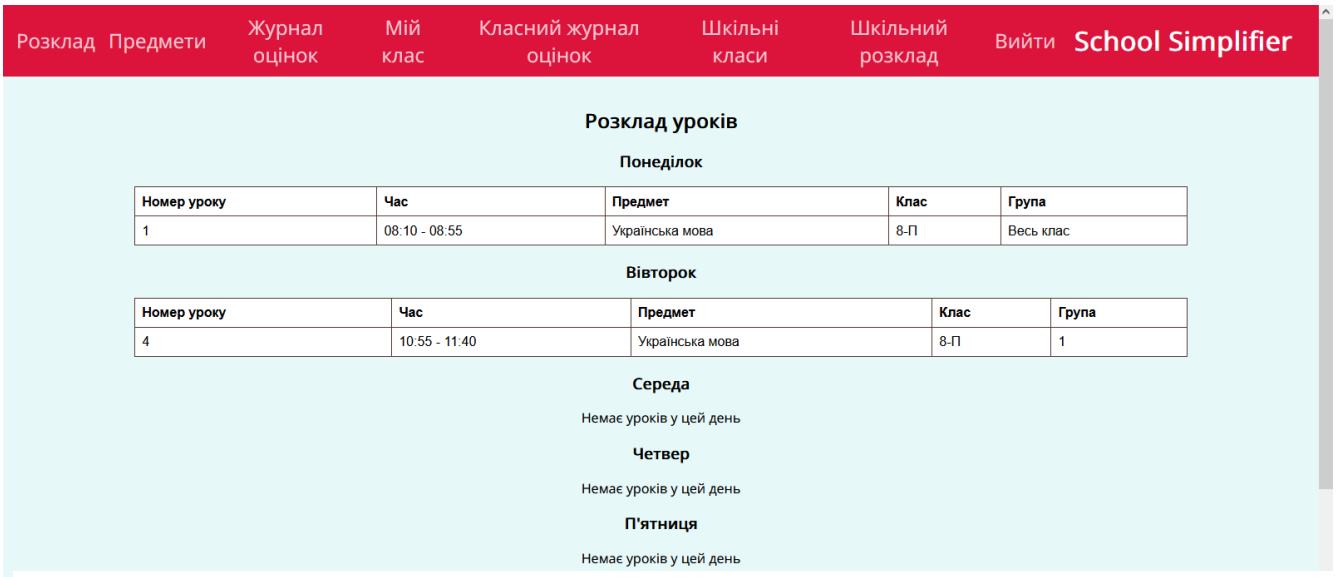


Рисунок 4.3.1. Сторінка розкладу вчителя

На сторінці “Предмети” вчитель бачить список усіх предметів, які він викладає. Для кожного предмету наявні назва предмету, клас, якому він викладається, та номер групи цього класу.

Рисунок 4.3.2. Сторінка предметів вчителя

На сторінці “Журнал оцінок” спочатку наявна лише кнопка “Обрати журнал”. При натисненні на неї, з’являється форма, яка дозволяє вчителю обрати предмет, клас та групу, для якої він хоче переглянути журнал оцінок.

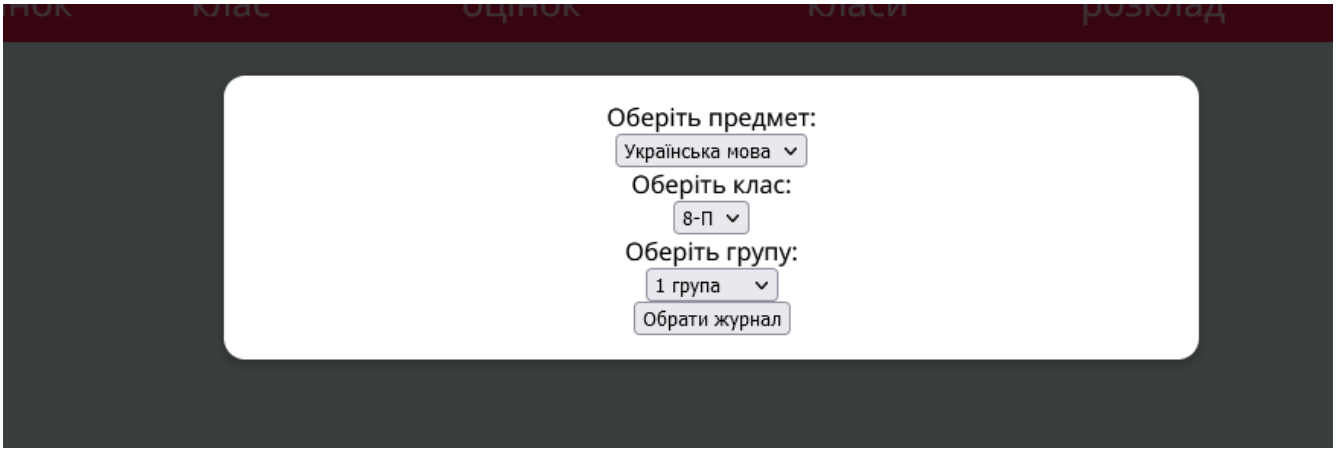


Рисунок 4.3.3. Форма для вибору журналу оцінок для вчителя

Після заповнення форми, на сторінці журналу з’являються кнопки для роботи із обраним журналом оцінок. Для роботи із поточними та тематичними

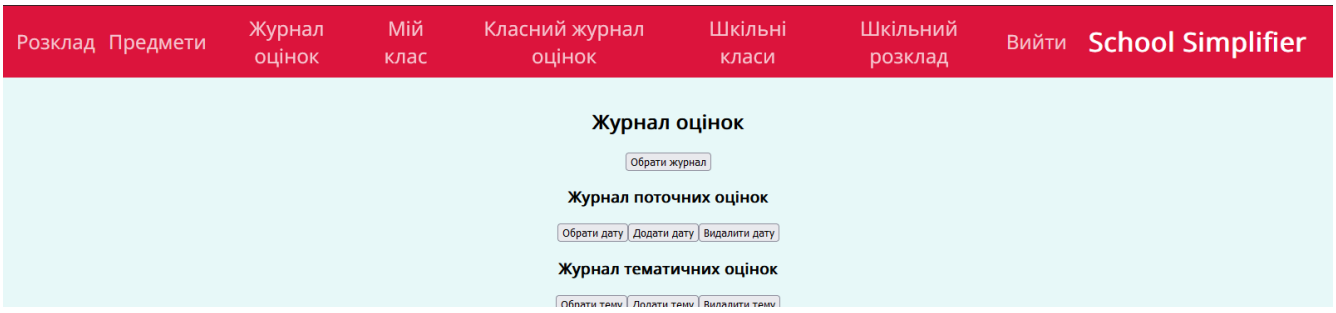


Рисунок 4.3.4. Сторінка журналу оцінок вчителя після вибору журналу

оцінками наявні по 3 кнопки – для того, щоб показати оцінки за обрану дату (або тему), для додавання нової дати та для видалення дати. Тут будуть описані 3 кнопки для роботи з поточними оцінками, оскільки робота із тематичними оцінками нічим не відрізняється.

При натисненні на кнопку “Додати дату” з’являється форма для додавання дати, за яку вчитель хоче виставити поточні оцінки та відмітити присутність учнів. Якщо така дата вже існує в журналі – виводиться повідомлення про помилку, якщо ж її немає – дата додається до журналу та стає доступною для перегляду.

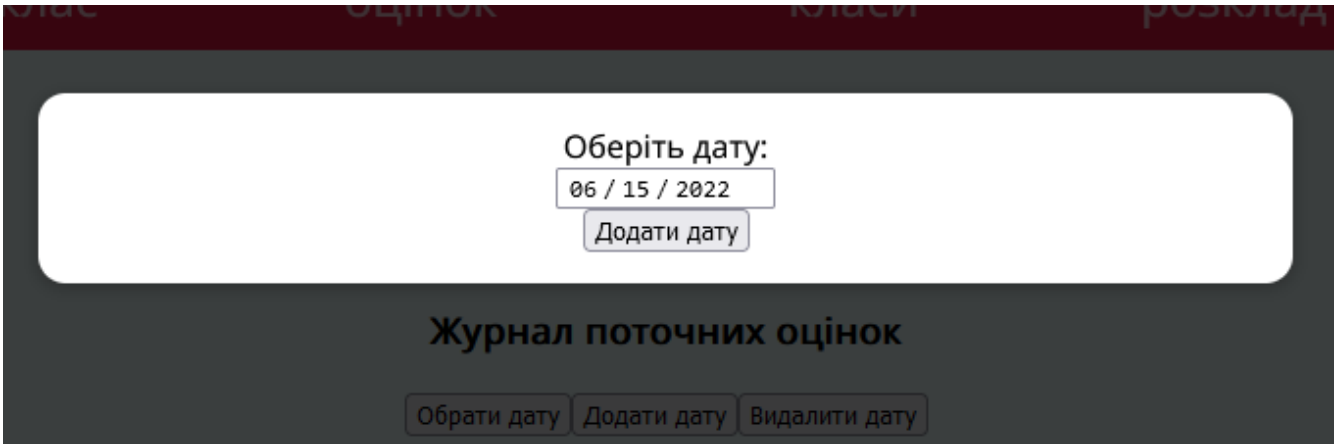


Рисунок 4.3.5. Форма для додавання нової дати до журналу оцінок вчителя

При натисненні на кнопку “Видалити дату” з’являється форма для вибору дати, яку вчитель хоче видалити. Якщо за дату немає жодних поточних оцінок – вона буде видалена, якщо ж є хоча б одна оцінка – виведеться повідомлення про помилку.

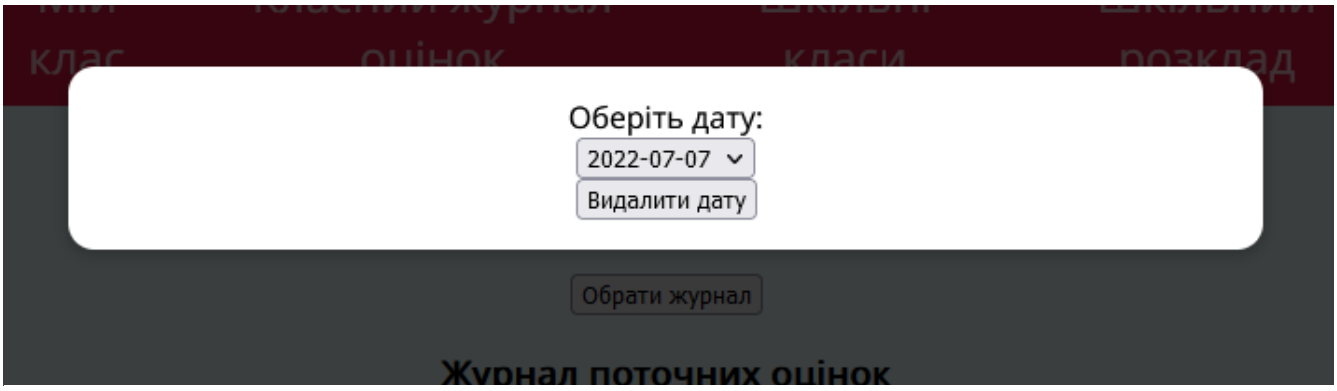


Рисунок 4.3.6. Форма для видалення дати із журналу оцінок для вчителя

При натисненні на кнопку “Обрати дату” з’являється форма для вибору дати, за яку вчитель хоче переглянути поточні оцінки.

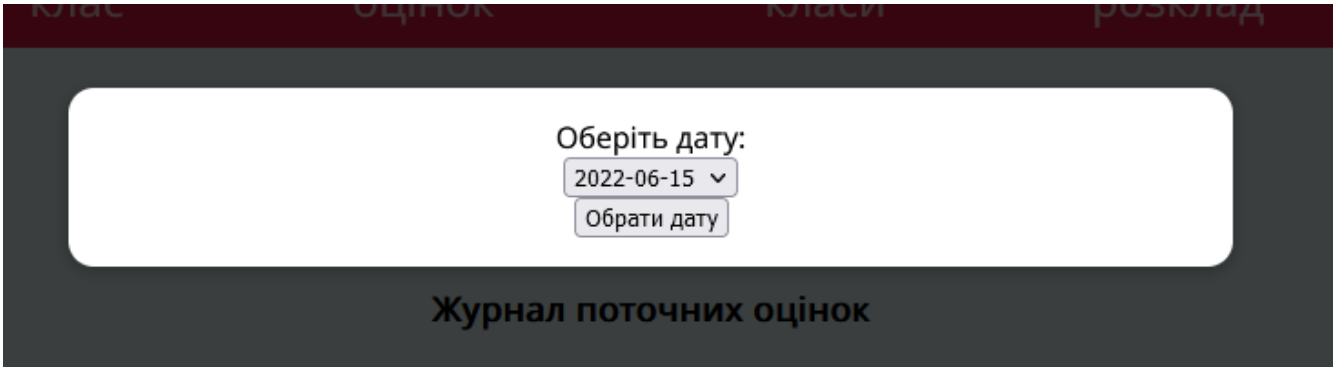


Рисунок 4.3.7. Форма для вибору дати, за яку вчитель хоче переглянути поточні оцінки

При виборі дати, з’являється список учнів з їх поточними оцінками. Для кожного учня може писати або “Оцінка не виставлена”, якщо вчитель ще не виставив оцінку, “Учень відсутній”, якщо вчитель відмітив учня як відсутнього, “Учень присутній”, якщо вчитель відмітив учня як присутнього, але учень не отримав оцінку, та “Учень присутній, оцінка”, якщо учень був присутній на уроці та отримав оцінку. Крім списку учнів з оцінками також з’являються ще 2 кнопки для додавання та видалення поточних оцінок.

Рисунок 4.3.8. Сторінка журналу оцінок після вибору вчителем дати для перегляду поточних оцінок

При натисненні на кнопку “Додати поточну оцінку” з’являється форма для додавання поточної оцінки учневі. У цій форми треба вибрати учня, позначити, чи присутній учень, якщо так – виставити отриману ним оцінку, або обрати варіант, що учень не отримав оцінку. Якщо учень позначений як відсутній, то у формі немає поля для введення оцінки, оскільки відсутній учень не може мати поточну оцінку за дату, коли він був відсутній.

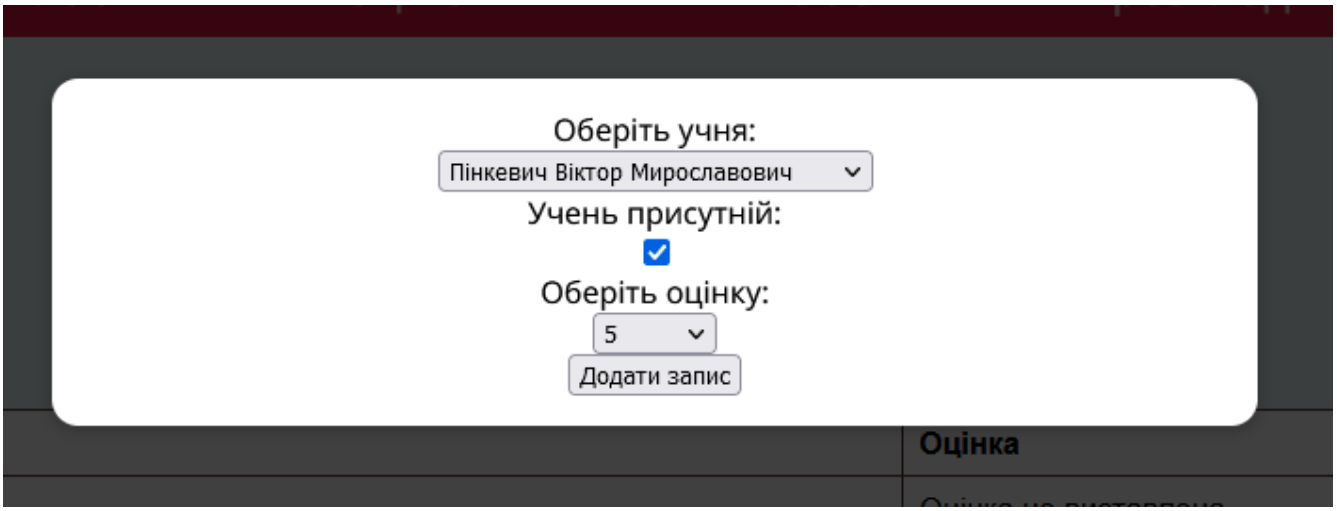


Рисунок 4.3.9. Форма для виставлення поточної оцінки

При натисненні на кнопку “Видалити поточну оцінку” з’являється форма для видалення поточної оцінки учневі. У цій формі треба вибрати учня, оцінку якого треба видалити. Вибрати треба саме учня, оскільки за одну дати учень може мати лише одну поточну оцінку.

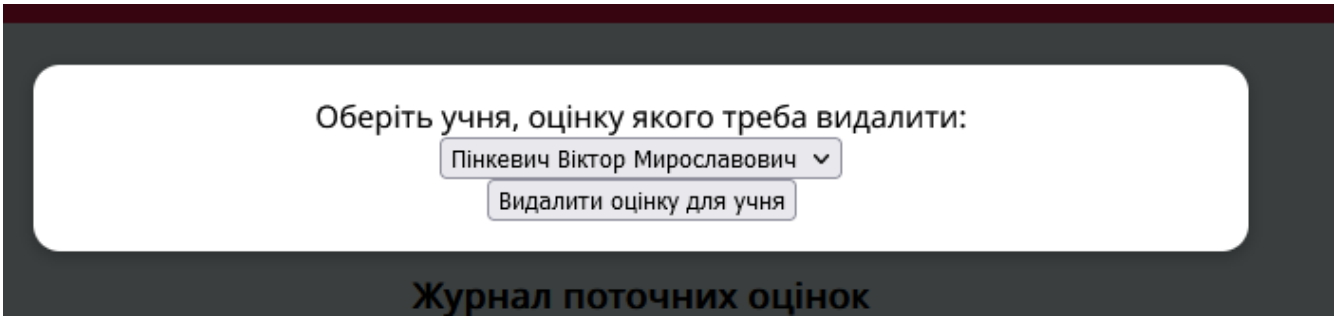


Рисунок 4.3.10. Форма для видалення поточної оцінки

Для тематичних оцінок робота із формами описуватись не буде, оскільки там усе дуже схоже на роботу із поточними оцінками. Єдина значна відмінність –

при виставленні оцінки за тему не треба відмічати присутність або відсутність учня, оскільки тематична оцінка має бути виставлена кожному учню обов’язково.

4.4 Можливості класного керівника

Після успішного входу в систему, класний керівник потрапляє на сторінку “Мій клас”, на якій міститься уся інформація про клас класного керівника. На цій сторінці можна побачити назву класу, список усіх учнів класу, список учнів кожної групи класу, список предметів класу та розклад класу. Усі елементи, крім назви класу, заховані під елементами-спойлерами, на які треба натиснути для показу відповідної інформації. Це було зроблено, оскільки показ одразу усіх елементів займає велику кількість місця і не є зручним для роботи.

Рисунок 4.4.1. Сторінка класу класного керівника одразу після входу в систему

Рисунок 4.4.2. Сторінка класу класного керівника після розгортання елементу “Список предметів”

Друга сторінка, на яку може перейти класний керівник, використовуючи верхнє меню – “Класний журнал оцінок”. На цій сторінці класний керівник може переглядати оцінки учнів свого класу із будь-якого предмету, який вивчається класом. Спочатку на цій сторінці наявна лише кнопка “Налаштування журналу”, при натисненні на яку з’являється форма для вибору журналу оцінок. У ній можна вибрати предмет та групу класу.

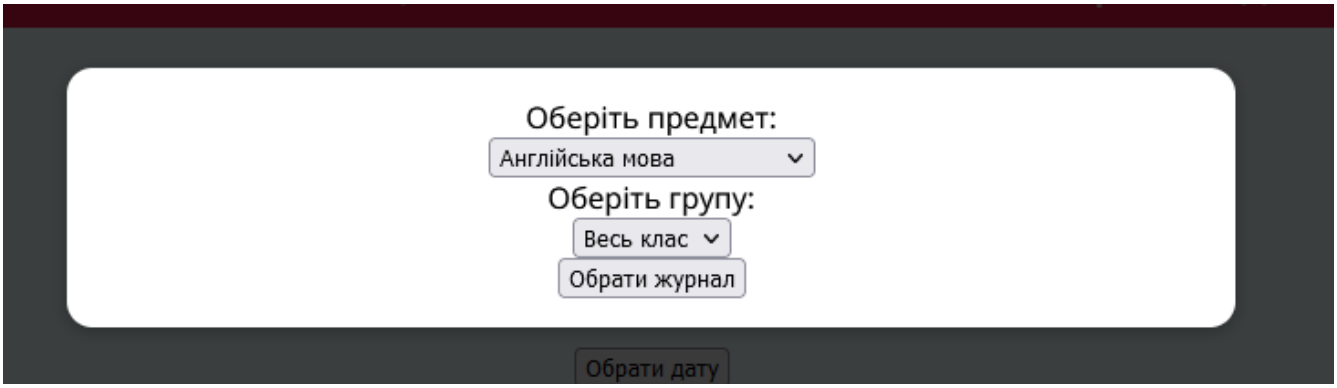


Рисунок 4.4.3. Форма для вибору журналу оцінок класним керівником

Після вибору журналу, на сторінці з’являються ще дві кнопки – для вибору дати, щоб переглянути поточні оцінки, і для вибору теми, щоб переглянути тематичні оцінки.

Рисунок 4.4.4. Сторінка класного журналу оцінок після вибору журналу

При натисненні на кнопку “Обрати дату” з’являється форма, яка дозволяє обрати дату для перегляду поточних оцінок учнів за обрану дату.

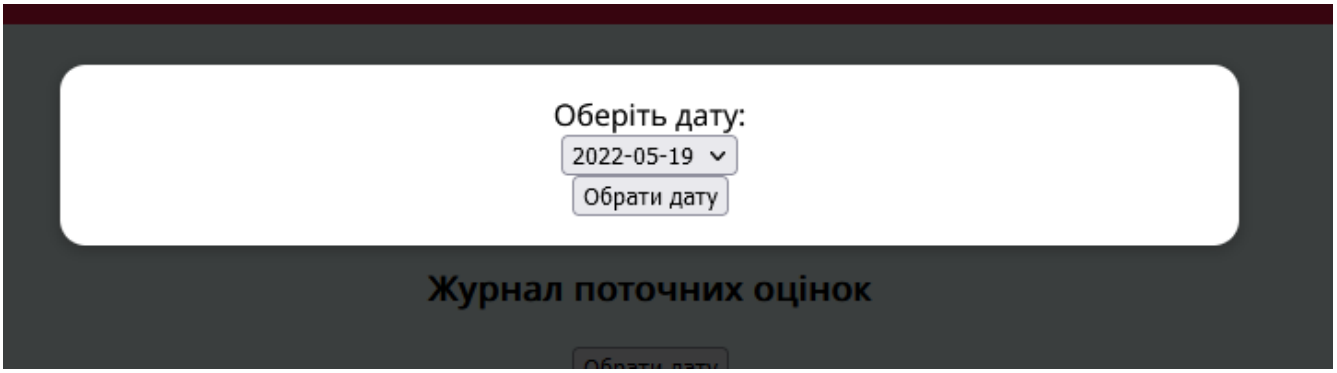


Рисунок 4.4.5. Форма для вибору дати для перегляду поточних оцінок класним керівником

Після вибору дати на сторінці з'являється список поточних оцінок учнів класу за обрану дату. Значення поля "Оцінка" можуть бути такими ж, як і описані при огляді можливостей вчителя та його сторінки "Журнал оцінок".

Рисунок 4.4.6. Сторінка класного журналу після вибору дати для перегляду поточних оцінок

При натисненні на кнопку "Обрати тему" з'являється форма, яка дозволяє обрати тему для перегляду оцінок учнів за обрану тему.

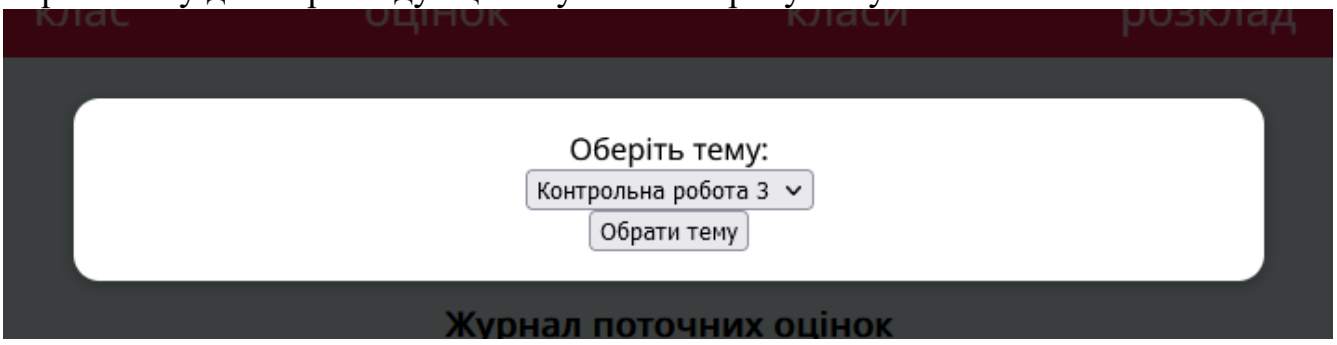


Рисунок 4.4.7. Форма для вибору теми для перегляду тематичних оцінок класним керівником

Після вибору теми на сторінці з’являється список тематичних оцінок учнів класу. Значення поля “Оцінка” можуть бути такими ж, як і описані при огляді можливостей вчителя та його сторінки “Журнал оцінок”.

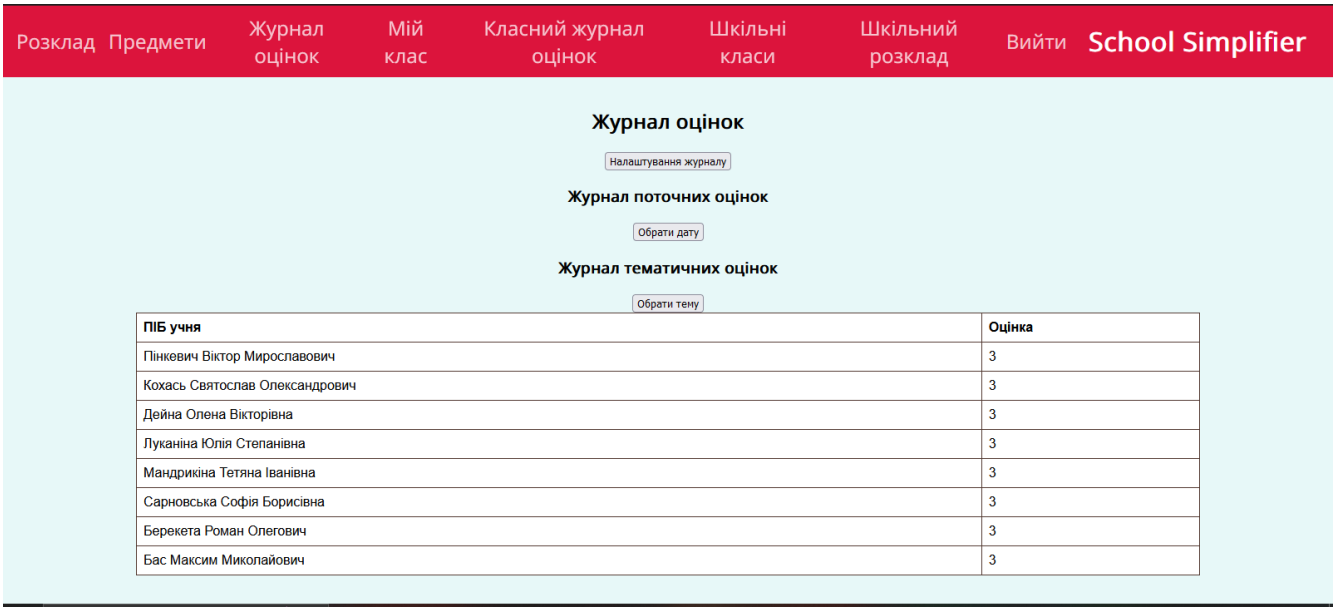


Рисунок 4.4.8. Сторінка класного журналу після вибору теми для перегляду тематичних оцінок

4.5 Можливості заступника директора

Після успішного входу в систему, заступник директора потрапляє на сторінку “Шкільний розклад”, на якій можна переглянути інформацію про розклад для будь-якого класу, додати та видалити можливий час проведення уроків а також додавати та видаляти записи у розкладі. На цій сторінці спочатку наявні лише 3 кнопки – “Обрати клас”, “Додати урок” та “Видалити урок”.

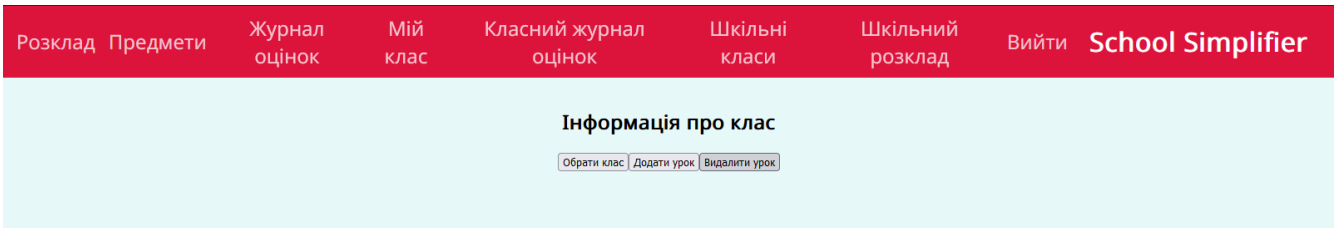


Рисунок 4.5.1. Сторінка шкільного розкладу одразу після входу в систему

При натисненні на кнопку “Додати урок” з’являється форма, яка дозволяє додати новий час проведення уроку. У ній потрібно ввести номер уроку, час його

початку та кінця. Якщо урок із такими параметрами уже існує – буде виведено повідомлення про помилку, якщо ж усе введено правильно – буде створено новий урок, який можна використовувати при створенні розкладу.

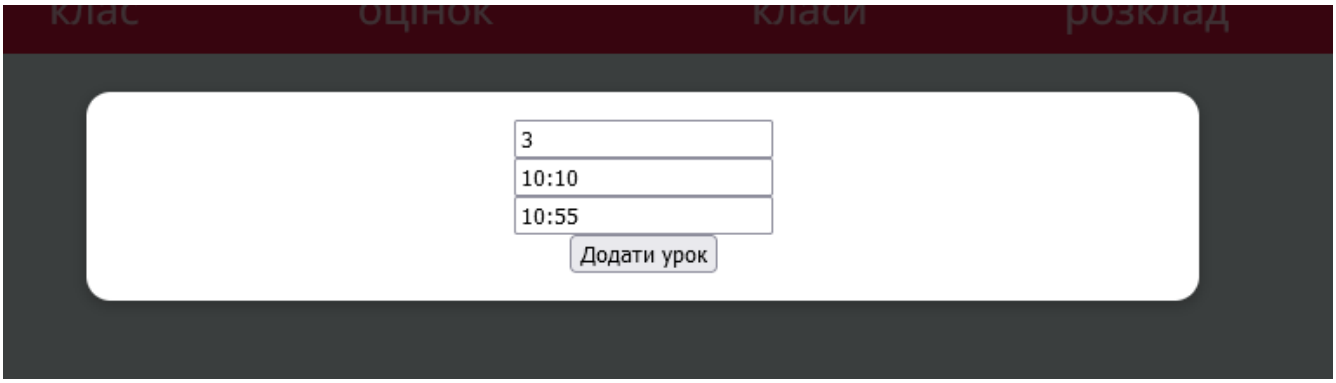


Рисунок 4.5.2. Форма для створення нового уроку

При натисненні на кнопку “Видалити урок” з’являється форма, яка дозволяє видалити час проведення уроку. Якщо обраний урок використовується хоча б в одному записі в розкладі, буде виведено повідомлення про помилку, якщо ж немає жодного запису в розкладі для цього уроку, його буде видалено.

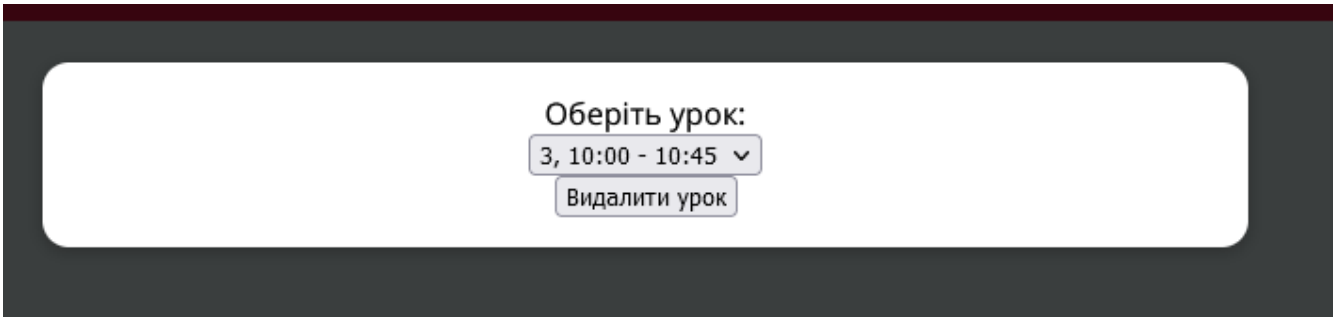


Рисунок 4.5.3. Форма для видалення уроку

При натисненні на кнопку “Обрати клас” з’являється форма, яка дозволяє обрати клас, розклад якого потрібно переглянути.

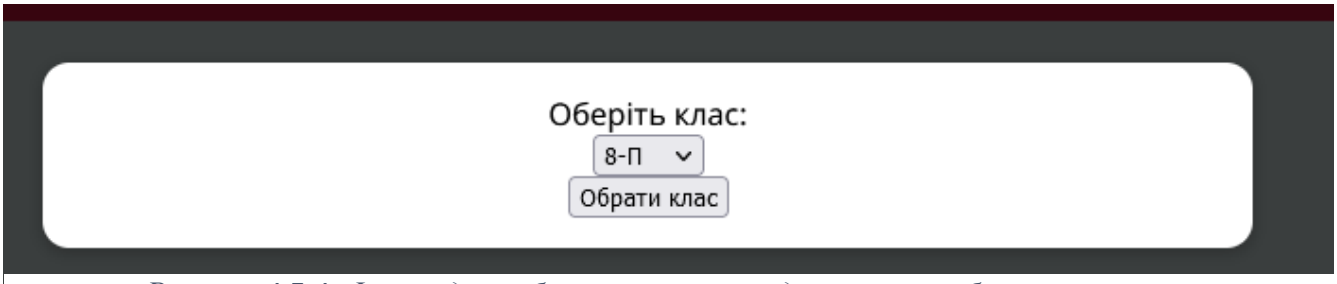


Рисунок 4.5.4. Форма для вибору класу, розклад якого потрібно переглянути

Після вибору класу, на сторінці з’явиться розклад обраного класу та 2 нові кнопки – “Додати запис у розклад” та “Видалити запис із розкладу”.

П'ятниця

Номер уроку	Час	Предмет	Група	Вчитель
1	08:10 - 08:55	Трудове навчання	Весь клас	Петренко Марія Іванівна
2	09:00 - 09:45	Трудове навчання	Весь клас	Петренко Марія Іванівна
3	10:00 - 10:45	Хімія	Весь клас	Воробйова Вікторія Степанівна
4	10:55 - 11:40	Історія України	Весь клас	Михайленко Степан Вікторович
5	12:00 - 12:45	Геометрія	1	Мельник Оксана Леонідівна
5	12:00 - 12:45	Інформатика	2	Мельниченко Оксана Миколаївна
6	12:55 - 13:40	Інформатика	1	Мельниченко Оксана Миколаївна
6	12:55 - 13:40	Геометрія	2	Мельник Оксана Леонідівна

Субота

Немає уроків у цей день

Неділя

Немає уроків у цей день

Додати запис у розклад | Видалити запис із розкладу

Рисунок 4.5.5. Нижня частина сторінки "Шкільний розклад" після вибору класу, розклад якого потрібно переглянути

При натисненні на кнопку “Додати запис у розклад” з’являється форма для додавання нового запису до розкладу. Для її заповнення треба обрати день тижня, урок, предмет, вчителя та групу класу, для якої треба додати запис у розклад. Після її заповнення, до розкладу класу буде додано новий запис, який одразу можна побачити в інтерфейсі користувача.

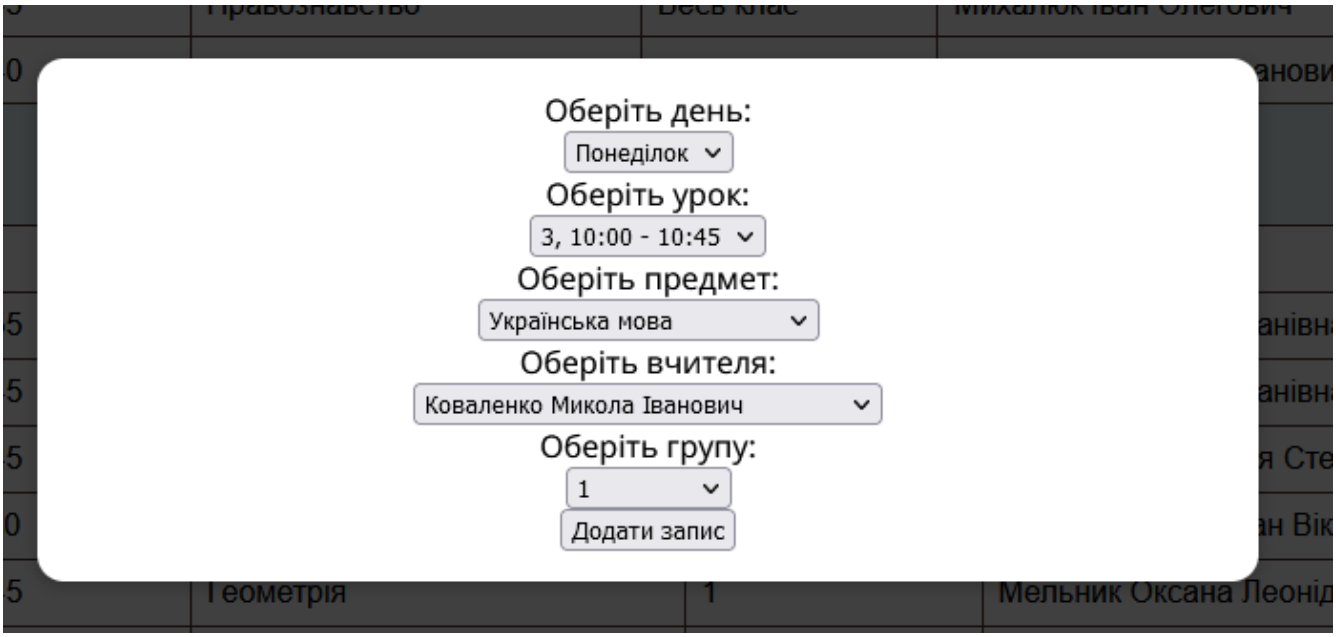


Рисунок 4.5.6. Форма для додавання запису у розклад

При натисненні на кнопку “Видалити запис із розкладу” з’являється форма для видалення запису з розкладу. Для того, щоб обрати правильний запис, при виборі відображається максимальна кількість інформації про запис.

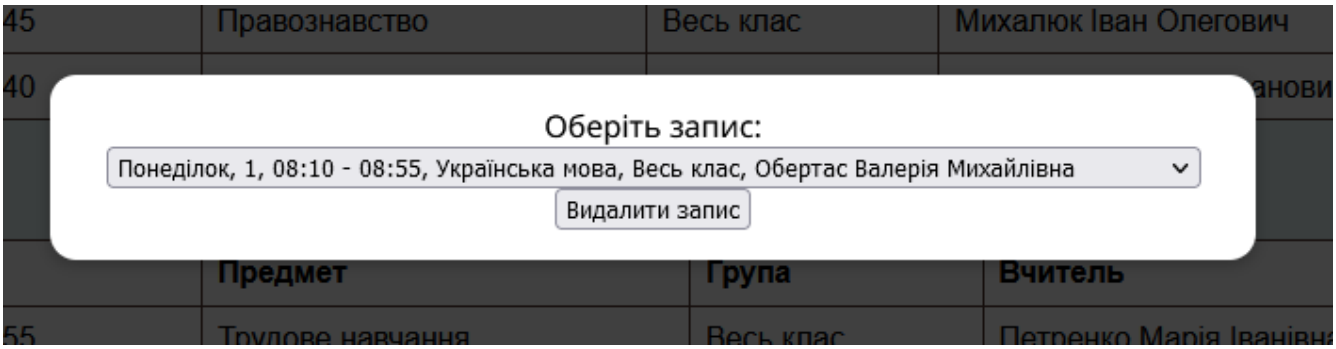


Рисунок 4.5.7. Форма для видалення запису з розкладу

Друга сторінка, на яку може перейти заступник директора, використовуючи верхнє меню – “Шкільні класи”. На цій сторінці можна додавати та видаляти класи, переглядати інформацію про будь-який клас, додавати та видаляти групи для класу, змінювати класи та групи для учнів. Спочатку на цій сторінці наявні лише 3 кнопки – “Обрати клас”, “Додати клас” та “Видалити клас”.

При натисканні на кнопку “Додати клас” з’являється форма, яка дозволяє додати новий клас. Для додавання класу треба ввести його назву, обрати класного керівника та ввести кількість груп. Якщо якісь дані будуть неправильними (наприклад, клас із введеною назвою уже існує) виведеться повідомлення про помилку. Якщо всі дані буде введено правильно – буде створено новий клас.

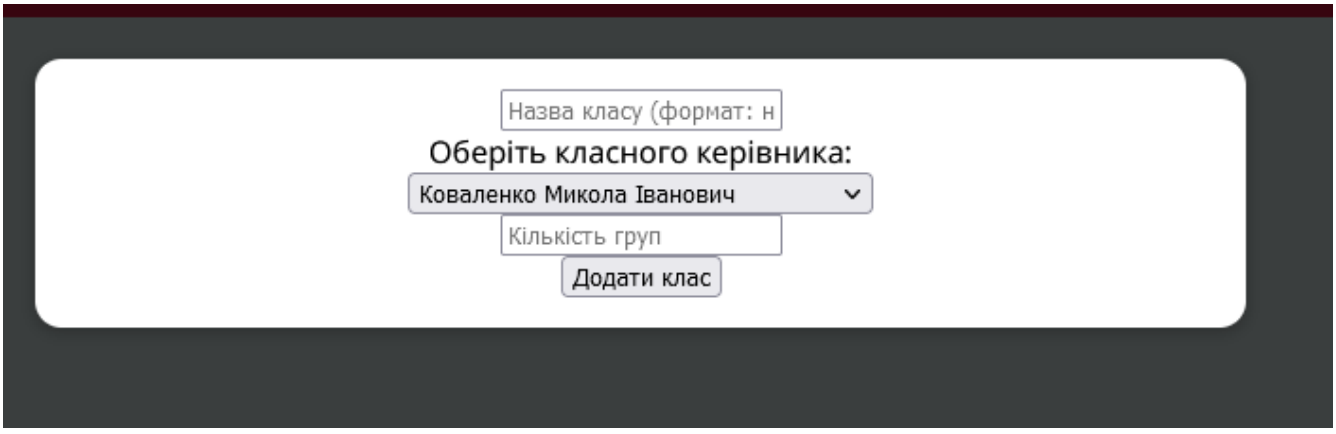


Рисунок 4.5.8. Форма для додавання класу

При натисканні на кнопку “Видалити клас” з’являється форма, яка дозволяє видалити клас. Якщо у класі ще є учні чи групи – буде виведено повідомлення про помилку, інакше – клас буде видалено.

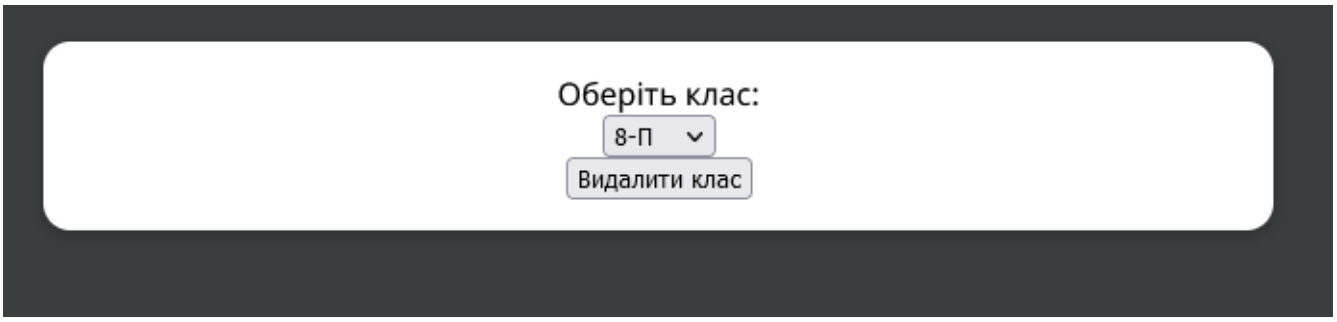


Рисунок 4.5.9. Форма для видалення класу

При натисканні на кнопку “Обрати клас” з’являється форма, яка дозволяє обрати клас, інформацію про який потрібно переглянути.

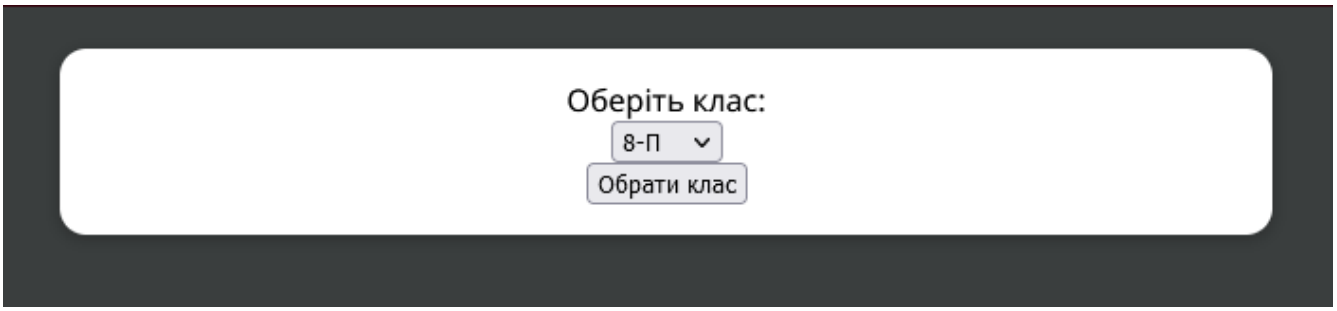


Рисунок 4.5.10. Форма для вибору класу, інформацію про який потрібно переглянути

Після вибору класу на сторінці з’явиться інформація, схожа на інформацію на сторінці “Мій клас” класного керівника – назва класу та 3 елементи-спойлери, які приховують інформацію про усіх учнів класу, список предметів, які вивчаються класом, та розклад класу. Проте наявні також кнопки – “Додати групу для класу”, “Видалити групу для класу” та “Змінити групу і клас для учня”.

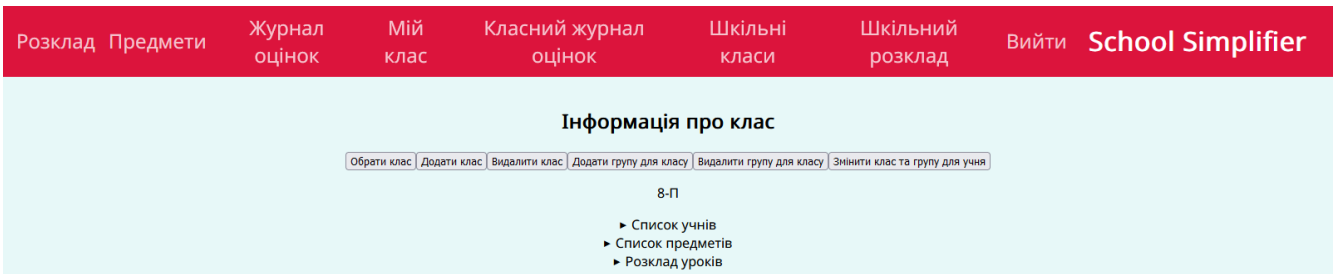


Рисунок 4.5.11. Сторінка шкільних класів після вибору класу, інформацію про який потрібно переглянути

При натисненні на кнопку “Додати групу для класу” з’являється форма для додавання нової групи до класу. Для додавання групи достатньо ввести лише її номер. Якщо група із таким номером уже існує – буде виведено повідомлення про помилку, інакше – група буде додана до класу.

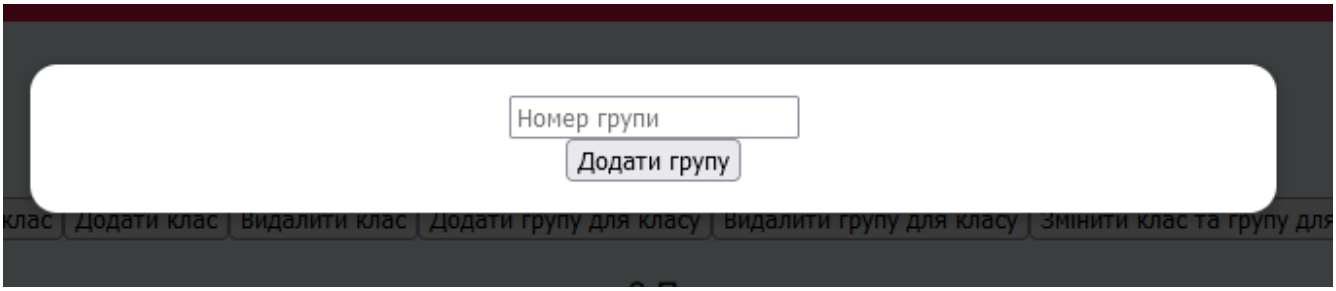


Рисунок 4.5.12. Форма для додавання групи до класу

При натисканні на кнопку “Видалити групу для класу” з’являється форма для видалення групи класу. Якщо у групі є хоча б один учень – буде виведено повідомлення про помилку, інакше – група буде видалена.

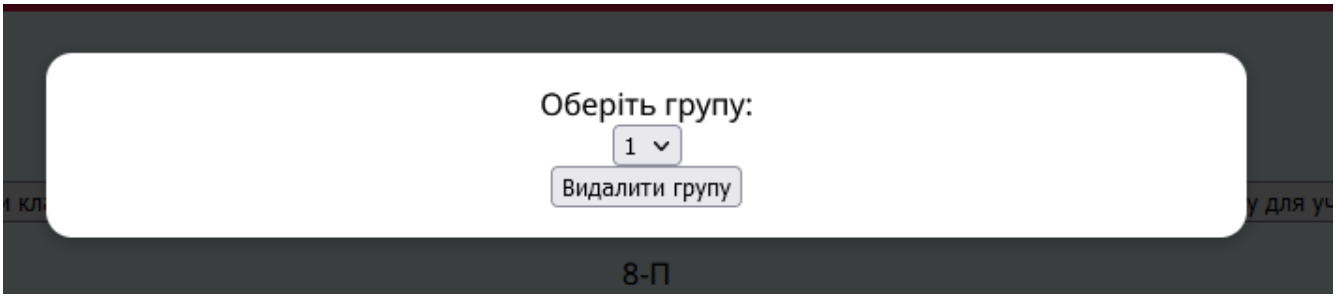


Рисунок 4.5.13. Форма для видалення групи класу

При натисканні на кнопку “Змінити групу і клас для учня” буде виведено форму, яка дозволяє задати нову групу для будь-якого учня. Обраний учень буде переведений до класу, який був обраний раніше та до обраної групи.

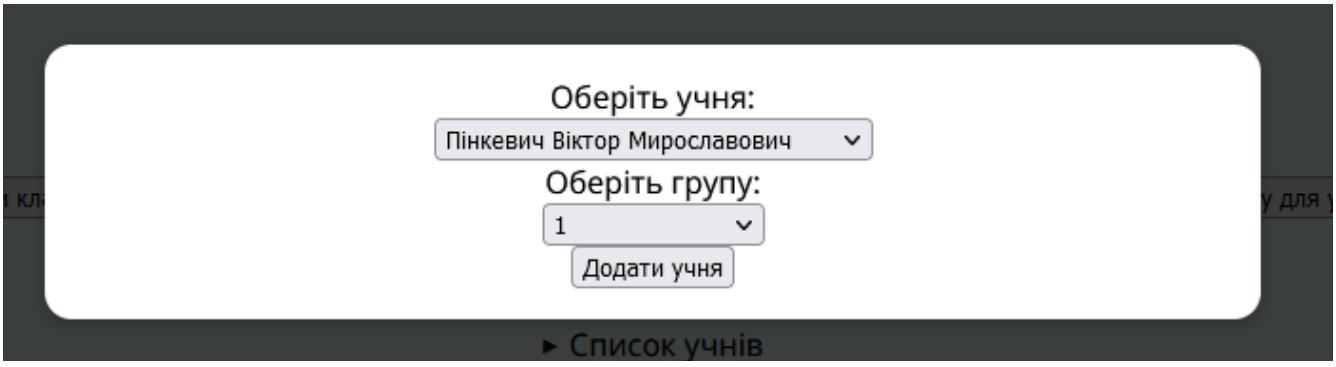


Рисунок 4.5.14. Форма для зміни класу та групи учня

4.6 Можливості розширення та покращення застосунку

Розроблений застосунок має усі можливості, які були описані у технічних вимогах. На даний момент, він орієнтований на цифровізацію базових речей освітнього процесу, проте, з урахуванням сучасних тенденцій, доречним буде додати також можливості для організації дистанційного навчання. Для подальшого розвитку застосунку можна виділити такі основні напрямки:

- розвиток можливостей дистанційного навчання;
- засоби для додавання вчителями домашніх завдань, їх перевірки та оцінювання;
- розробка власної системи онлайн-тестувань або інтеграція з уже існуючими;
- можливості завантаження файлів із виконаними завданнями;
- покращення уже розробленого функціоналу, користувацького інтерфейсу та користувацького досвіду;
- розробка клієнтських застосунків для смартфонів та комп’ютерів.

4.7 Висновки

В ході виконання роботи було проведено аналіз сучасного стану цифровізації шкільних процесів. На основі цього аналізу було виявлено, що сучасні технології дозволяють покращити та зробити приємнішим процес шкільного навчання. Для вирішення деяких задач було розроблено перший прототип веб-застосунку, який у подальшому може стати базою для більш масштабних розробок в галузі.

При розробці застосунку було отримано досвід розробки усіх частин веб-застосування – як клієнтської, так і серверної. Використання на практиці принципів REST, MVC, SPA та інших дозволило покращити знання із побудови

архітектури застосунку та застосувати кращі практики, які значно спрощують процес розробки. Ці знання можуть у майбутньому бути застосовані для створення більш масштабних систем.

Важливою частиною курсової роботи було ознайомлення із клієнтськими бібліотеками React, Redux, React Router і серверними фреймворками та бібліотеками Spring, Hibernate. Наведені вище засоби сьогодні є стандартами розробки сучасних застосунків та дозволяють отримати досвід, який буде корисним при опануванні інших, схожих інструментів розробки, або при створенні власних таких інструментів.

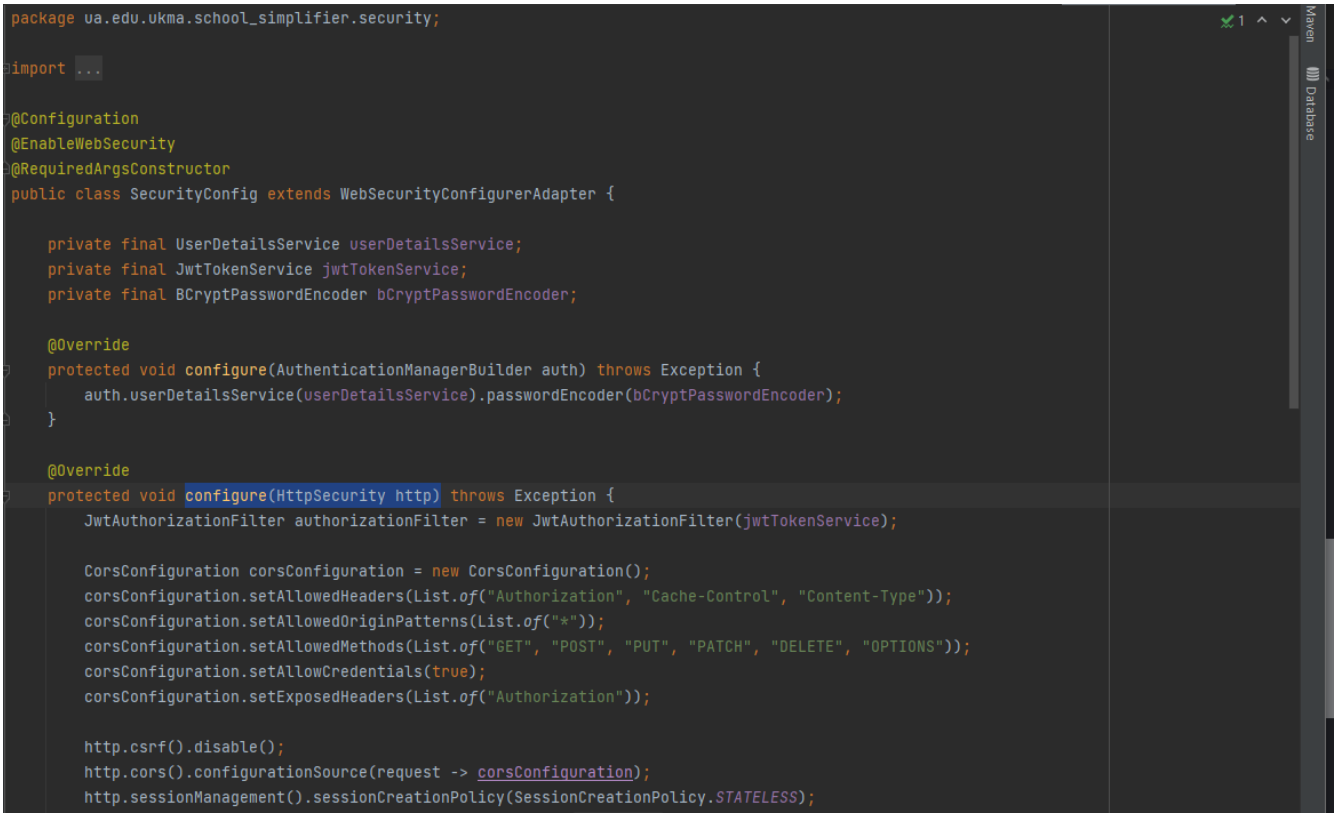
Список використаних джерел

1. What is ER Modeling? Learn with Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/er-modeling.html.
2. Relation Data Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/dbms/relational_data_model.htm.
3. Документація СКБД PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/.
4. What is Client Server Architecture? [Електронний ресурс] – Режим доступу до ресурсу: https://intellipaat.com/blog/what-is-client-server-architecture/#no1.
5. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/.
6. Документація Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/why-spring.
7. Inversion of Control and Dependency Injection with Spring [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring.
8. Документація Hibernate [Електронний ресурс] – Режим доступу до ресурсу: https://hibernate.org/orm/.
9. What is Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/product/what-is-postman/.
10. The Model View Controller Pattern [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/.
11. JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/.
12. Controller-Service-Repository [Електронний ресурс] – Режим доступу до ресурсу: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5.

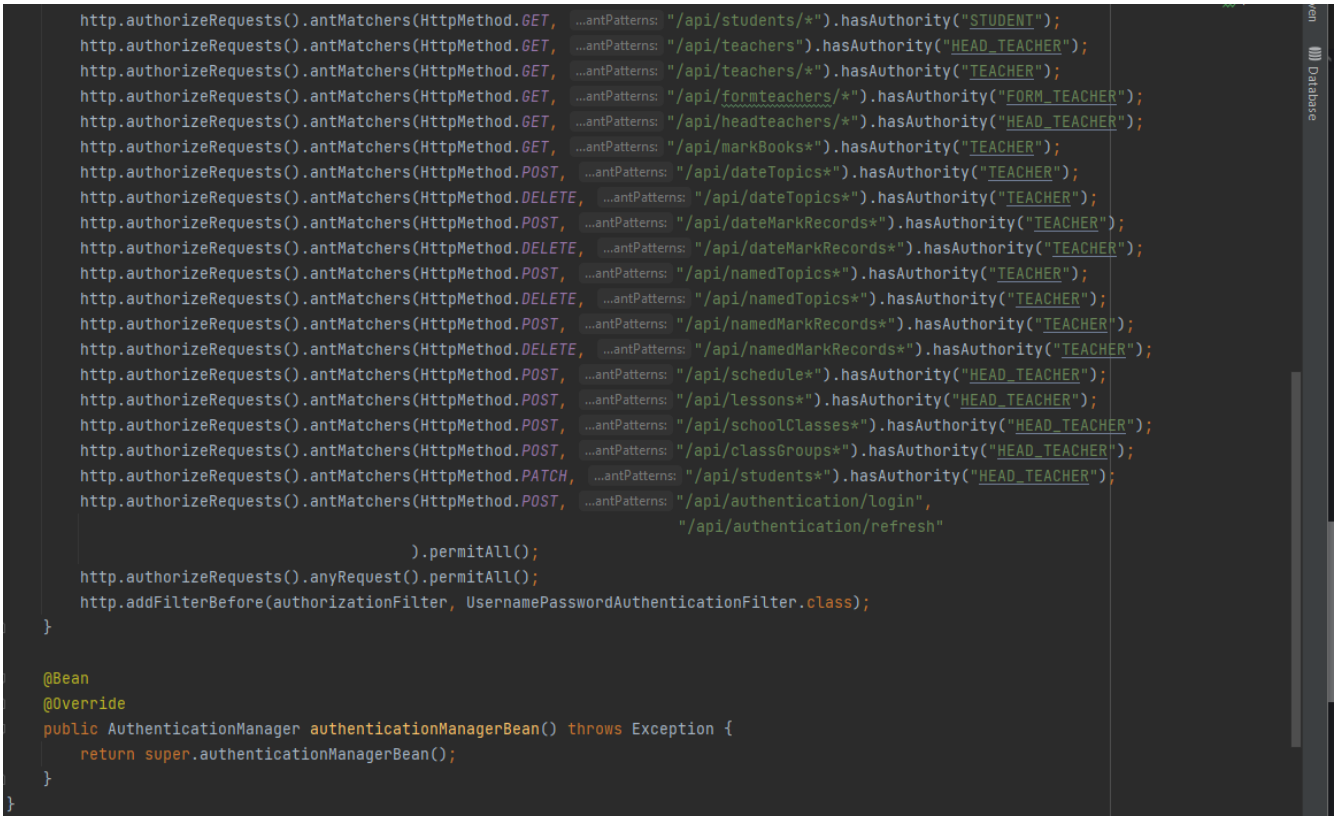
13. Документація React.js [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/.
14. Документація Redux [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/.
15. Документація React Router [Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/.
16. Single-page application vs. multiple-page application [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

Додатки

Додаток А. Клас SecurityConfig

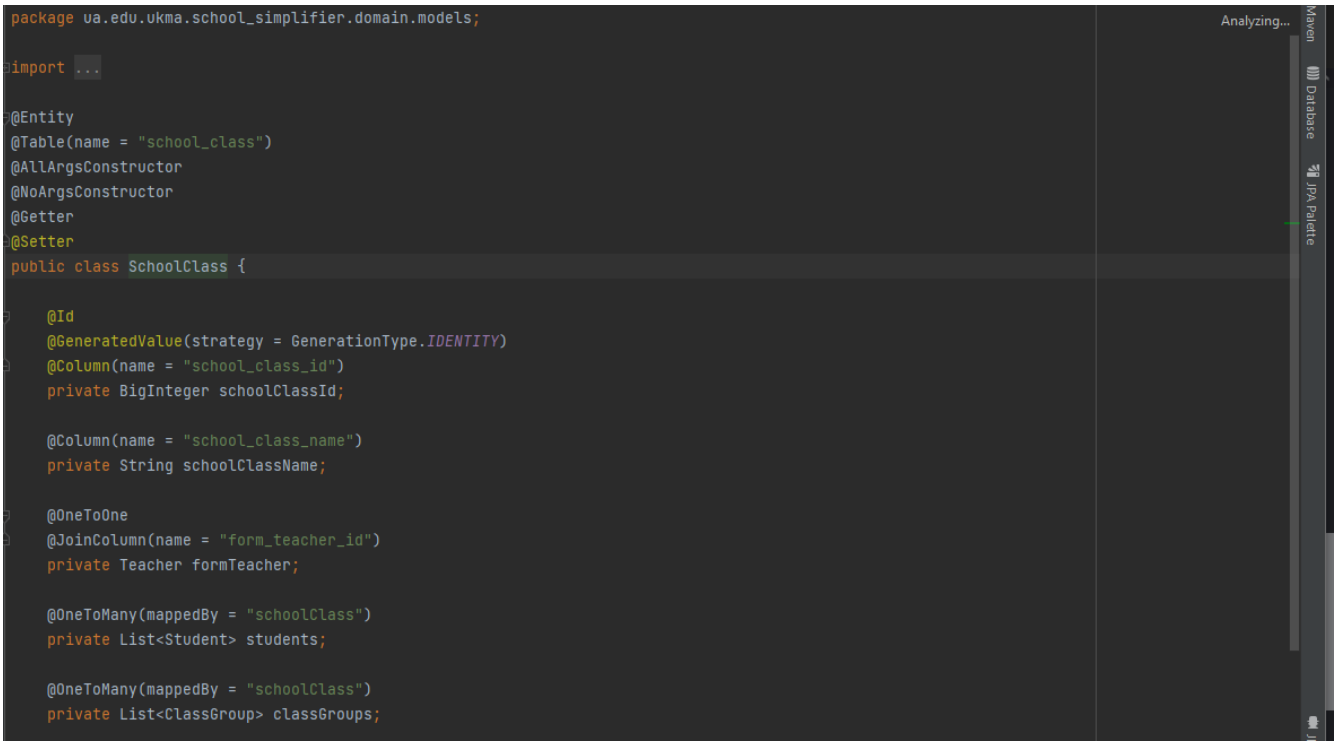


```
package ua.edu.ukma.school_simplifier.security;

import ...

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;
    private final JwtTokenService jwtTokenService;
    private final BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        JwtAuthorizationFilter authorizationFilter = new JwtAuthorizationFilter(jwtTokenService);

        CorsConfiguration corsConfiguration = new CorsConfiguration();
        corsConfiguration.setAllowedHeaders(List.of("Authorization", "Cache-Control", "Content-Type"));
        corsConfiguration.setAllowedOriginPatterns(List.of("*"));
        corsConfiguration.setAllowedMethods(List.of("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"));
        corsConfiguration.setAllowCredentials(true);
        corsConfiguration.setExposedHeaders(List.of("Authorization"));

        http.csrf().disable();
        http.cors().configurationSource(request -> corsConfiguration);
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
```



```
        http.authorizeRequests().antMatchers(HttpMethod.GET, ...antPatterns: "/api/students/*").hasAuthority("STUDENT");
        http.authorizeRequests().antMatchers(HttpMethod.GET, ...antPatterns: "/api/teachers").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.GET, ...antPatterns: "/api/teachers/*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.GET, ...antPatterns: "/api/formteachers/*").hasAuthority("FORM_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.GET, ...antPatterns: "/api/headteachers/*").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.GET, ...antPatterns: "/api/markBooks*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/dateTopics*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.DELETE, ...antPatterns: "/api/dateTopics*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/dateMarkRecords*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.DELETE, ...antPatterns: "/api/dateMarkRecords*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/namedTopics*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.DELETE, ...antPatterns: "/api/namedTopics*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/namedMarkRecords*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.DELETE, ...antPatterns: "/api/namedMarkRecords*").hasAuthority("TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/schedule*").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/lessons*").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/schoolClasses*").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/classGroups*").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.PATCH, ...antPatterns: "/api/students*").hasAuthority("HEAD_TEACHER");
        http.authorizeRequests().antMatchers(HttpMethod.POST, ...antPatterns: "/api/authentication/login",
                                                               "/api/authentication/refresh"
                                    ).permitAll();
        http.authorizeRequests().anyRequest().permitAll();
        http.addFilterBefore(authorizationFilter, UsernamePasswordAuthenticationFilter.class);
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

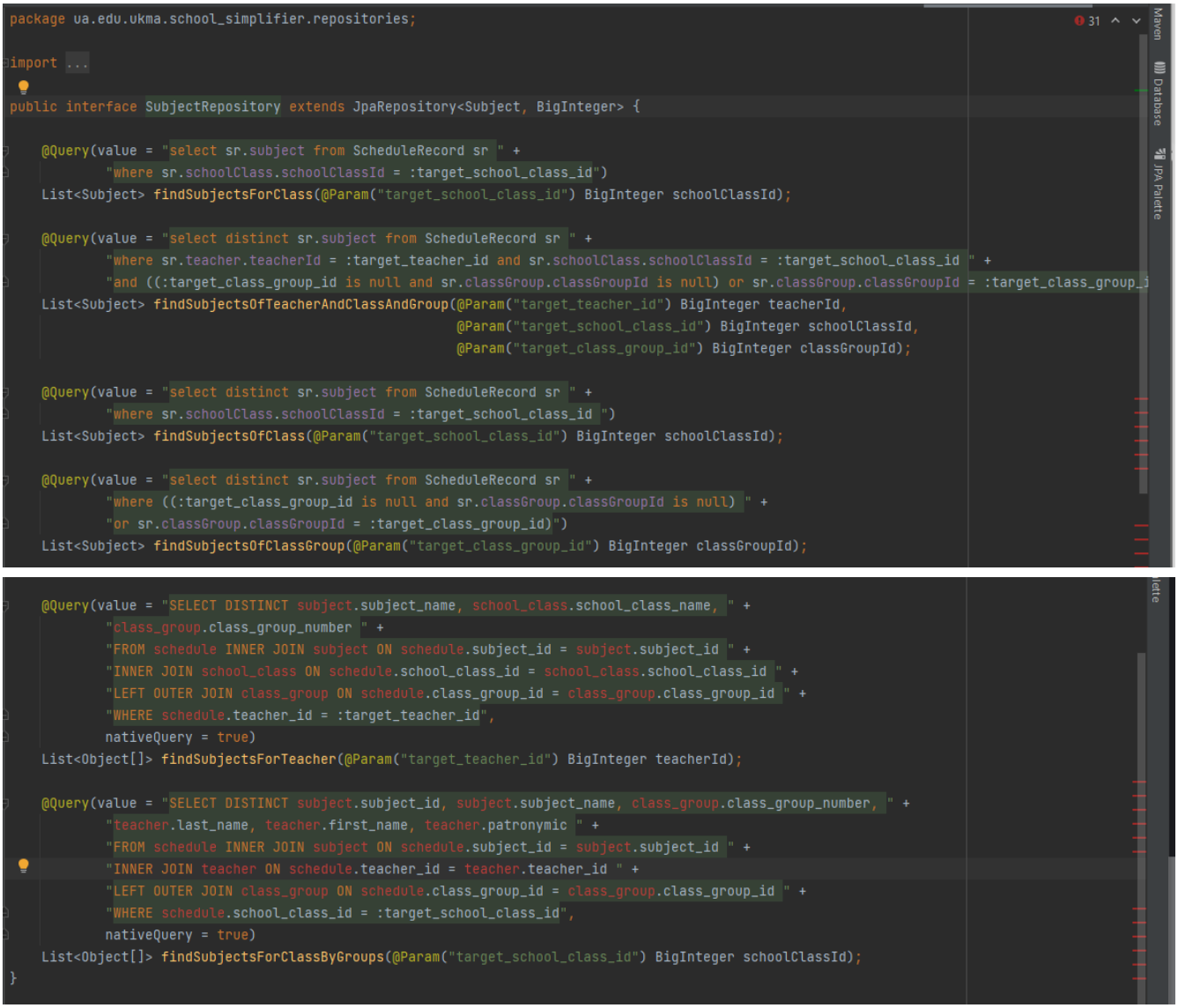
Додаток Б. Клас SchoolClass



```java
package ua.edu.ukma.school_simplifier.domain.models;

import ...

@Entity
@Table(name = "school_class")
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class SchoolClass {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "school_class_id")
    private BigInteger schoolClassId;

    @Column(name = "school_class_name")
    private String schoolClassName;

    @OneToOne
    @JoinColumn(name = "form_teacher_id")
    private Teacher formTeacher;

    @OneToMany(mappedBy = "schoolClass")
    private List<Student> students;

    @OneToMany(mappedBy = "schoolClass")
    private List<ClassGroup> classGroups;
```

```java
    @OneToMany(mappedBy = "schoolClass")
    private List<ScheduleRecord> scheduleRecords;

    @OneToMany(mappedBy = "schoolClass")
    private List<MarkBook> markBooks;
}
```

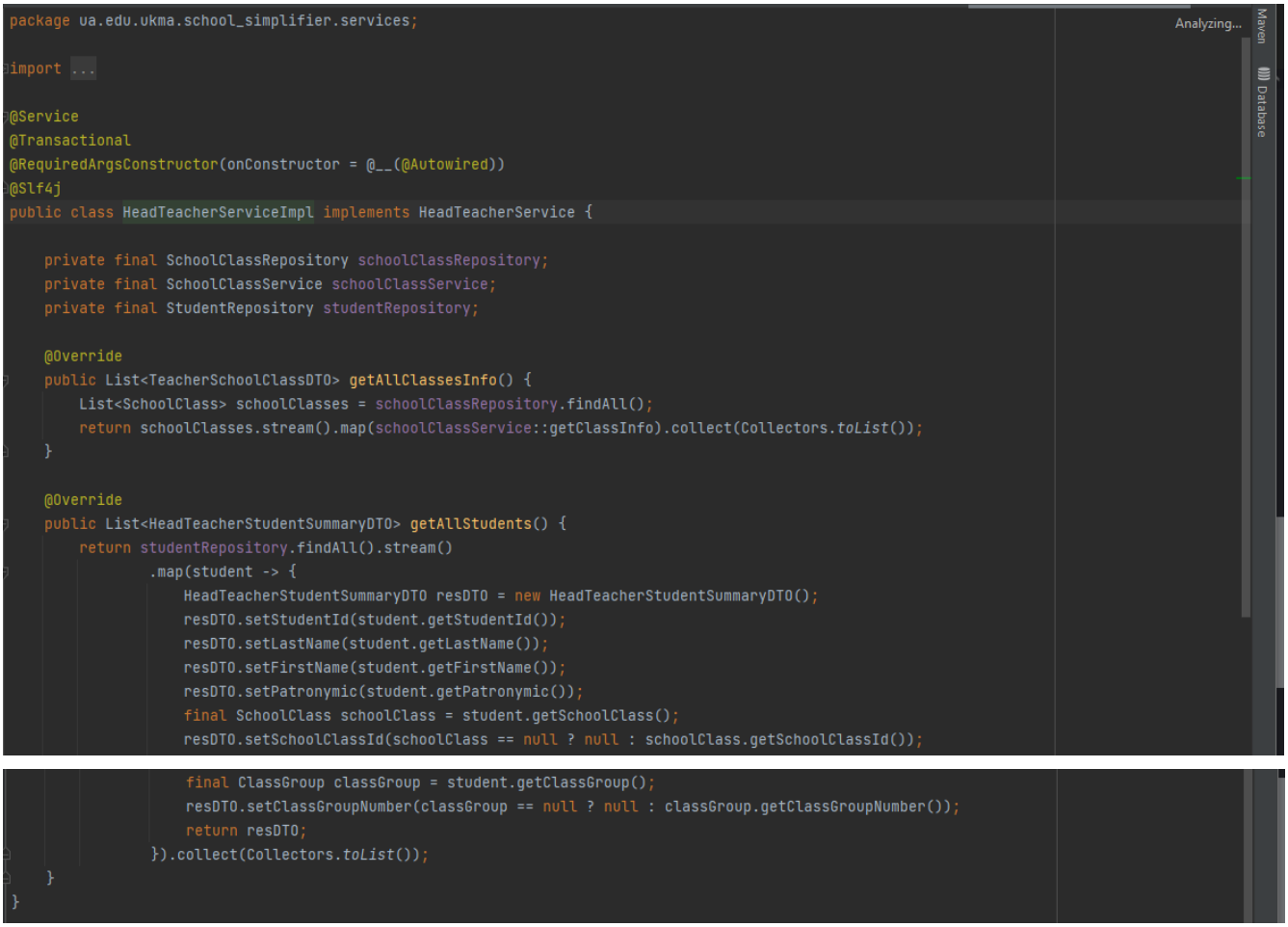
Додаток В. Клас SubjectRepository



```
package ua.edu.ukma.school_simplifier.repositories;

import ...

public interface SubjectRepository extends JpaRepository<Subject, BigInteger> {

    @Query(value = "select sr.subject from ScheduleRecord sr " +
            "where sr.schoolClass.schoolClassId = :target_school_class_id")
    List<Subject> findSubjectsForClass(@Param("target_school_class_id") BigInteger schoolClassId);

    @Query(value = "select distinct sr.subject from ScheduleRecord sr " +
            "where sr.teacher.teacherId = :target_teacher_id and sr.schoolClass.schoolClassId = :target_school_class_id " +
            "and ((:target_class_group_id is null and sr.classGroup.classGroupId is null) or sr.classGroup.classGroupId = :target_class_group_i
    List<Subject> findSubjectsOfTeacherAndClassAndGroup(@Param("target_teacher_id") BigInteger teacherId,
                                                        @Param("target_school_class_id") BigInteger schoolClassId,
                                                        @Param("target_class_group_id") BigInteger classGroupId);

    @Query(value = "select distinct sr.subject from ScheduleRecord sr " +
            "where sr.schoolClass.schoolClassId = :target_school_class_id ")
    List<Subject> findSubjectsOfClass(@Param("target_school_class_id") BigInteger schoolClassId);

    @Query(value = "select distinct sr.subject from ScheduleRecord sr " +
            "where ((:target_class_group_id is null and sr.classGroup.classGroupId is null) " +
            "or sr.classGroup.classGroupId = :target_class_group_id)")
    List<Subject> findSubjectsOfClassGroup(@Param("target_class_group_id") BigInteger classGroupId);
```

```
    @Query(value = "SELECT DISTINCT subject.subject_name, school_class.school_class_name, " +
            "class_group.class_group_number " +
            "FROM schedule INNER JOIN subject ON schedule.subject_id = subject.subject_id " +
            "INNER JOIN school_class ON schedule.school_class_id = school_class.school_class_id " +
            "LEFT OUTER JOIN class_group ON schedule.class_group_id = class_group.class_group_id " +
            "WHERE schedule.teacher_id = :target_teacher_id",
            nativeQuery = true)
    List<Object[]> findSubjectsForTeacher(@Param("target_teacher_id") BigInteger teacherId);

    @Query(value = "SELECT DISTINCT subject.subject_id, subject.subject_name, class_group.class_group_number, " +
            "teacher.last_name, teacher.first_name, teacher.patronymic " +
            "FROM schedule INNER JOIN subject ON schedule.subject_id = subject.subject_id " +
            "INNER JOIN teacher ON schedule.teacher_id = teacher.teacher_id " +
            "LEFT OUTER JOIN class_group ON schedule.class_group_id = class_group.class_group_id " +
            "WHERE schedule.school_class_id = :target_school_class_id",
            nativeQuery = true)
    List<Object[]> findSubjectsForClassByGroups(@Param("target_school_class_id") BigInteger schoolClassId);
}
```

Додаток Г. Клас HeadTeacherServiceImpl



```java
package ua.edu.ukma.school_simplifier.services;

import ...

@Service
@Transactional
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@Slf4j
public class HeadTeacherServiceImpl implements HeadTeacherService {

    private final SchoolClassRepository schoolClassRepository;
    private final SchoolClassService schoolClassService;
    private final StudentRepository studentRepository;

    @Override
    public List<TeacherSchoolClassDTO> getAllClassesInfo() {
        List<SchoolClass> schoolClasses = schoolClassRepository.findAll();
        return schoolClasses.stream().map(schoolClassService::getClassInfo).collect(Collectors.toList());
    }

    @Override
    public List<HeadTeacherStudentSummaryDTO> getAllStudents() {
        return studentRepository.findAll().stream()
                .map(student -> {
                    HeadTeacherStudentSummaryDTO resDTO = new HeadTeacherStudentSummaryDTO();
                    resDTO.setStudentId(student.getStudentId());
                    resDTO.setLastName(student.getLastName());
                    resDTO.setFirstName(student.getFirstName());
                    resDTO.setPatronymic(student.getPatronymic());
                    final SchoolClass schoolClass = student.getSchoolClass();
                    resDTO.setSchoolClassId(schoolClass == null ? null : schoolClass.getSchoolClassId());
                    final ClassGroup classGroup = student.getClassGroup();
                    resDTO.setClassGroupNumber(classGroup == null ? null : classGroup.getClassGroupNumber());
                    return resDTO;
                }).collect(Collectors.toList());
    }
}
```

Додаток Д. Клас SchoolClassController

```java
package ua.edu.ukma.school_simplifier.controllers;

import ...

@RestController
@RequestMapping("/api/schoolClasses")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@Slf4j
public class SchoolClassController {

    private final SchoolClassService schoolClassService;

    @PostMapping("")
    public ResponseEntity<Object> addSchoolClass(@RequestBody AddSchoolClassDTO addSchoolClassDTO) {
        try {
            schoolClassService.addSchoolClass(addSchoolClassDTO);
            return ResponseEntity.status(HttpStatus.CREATED).build();
        } catch(InvalidParameterException ex) {
            final ErrorResponse errorResponse = new ErrorResponse(HttpStatus.BAD_REQUEST.value(), ex.getMessage());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errorResponse);
        }
    }
```

```java
    @DeleteMapping("/{schoolClassId}")
    public ResponseEntity<Object> deleteMarkRecord(@PathVariable(name = "schoolClassId") BigInteger schoolClassId) {
        try {
            schoolClassService.deleteSchoolClass(schoolClassId);
            return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
        } catch(InvalidParameterException ex) {
            final ErrorResponse errorResponse = new ErrorResponse(HttpStatus.BAD_REQUEST.value(), ex.getMessage());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errorResponse);
        }
    }
}
```

Додаток Е. Файл auth-slice.js

```
import { createSlice } from "@reduxjs/toolkit";
import jwtDecode from "jwt-decode";

const authSlice = createSlice({
    name: 'auth',
    initialState: {
        accessToken: null,
        refreshToken: null,
        roles: null,
    },
    reducers: {
        setAccessToken(state, action) {
            const newToken = action.payload.accessToken;
            state.accessToken = newToken;
            localStorage.setItem('accessToken', newToken);
            state.roles = jwtDecode(newToken).roles;
        },

        setRefreshToken(state, action) {
            const newToken = action.payload.refreshToken;
            state.refreshToken = newToken;
            localStorage.setItem('refreshToken', newToken);
        },

        deleteAccessToken(state, action) {
            state.accessToken = null;
            localStorage.removeItem('accessToken');
            state.roles = null;
        },
```

```
        deleteRefreshToken(state, action) {
            state.refreshToken = null;
            localStorage.removeItem('refreshToken');
        },

        loadTokensFromStorage(state, action) {
            const loadedAccessToken = localStorage.getItem('accessToken');
            const loadedRefreshToken = localStorage.getItem('refreshToken');
            if(loadedAccessToken) {
                state.accessToken = loadedAccessToken;
                state.roles = jwtDecode(loadedAccessToken).roles;
            }
            if(loadedRefreshToken) {
                state.refreshToken = loadedRefreshToken;
            }
        },

        logout(state, action) {
            state.accessToken = null;
            localStorage.removeItem('accessToken');
            state.refreshToken = null;
            localStorage.removeItem('refreshToken');
            state.roles = null;
        }
    }
});

export const authActions = authSlice.actions;
export default authSlice;
```

Додаток Ж. Файл App.js

```
import React from 'react';
import { Navigate, Route, Routes } from 'react-router-dom';
import Login from './pages/public/Login';
import NotFound from './pages/public/NotFound';
import './App.css';
import { useSelector } from 'react-redux';
import { getHomePageForUser } from './utils/navigation';
import RequireAuth from './components/security/RequireAuth';
import StudentSchedule from './pages/student/StudentSchedule';
import TeacherSchedule from './pages/teacher/TeacherSchedule';
import { Roles } from './domain/constants';
import StudentSubjects from './pages/student/StudentSubjects';
import StudentClass from './pages/student/StudentClass';
import StudentMarks from './pages/student/StudentMarks';
import TeacherSubjects from './pages/teacher/TeacherSubjects';
import FormTeacherClass from './pages/formteacher/FormTeacherClass';
import FormTeacherClassMarkBook from './pages/formteacher/FormTeacherClassMarkBook';
import TeacherMarkBook from './pages/teacher/TeacherMarkBook';
import AuthLayout from './layout/AuthLayout';
import PublicLayout from './layout/PublicLayout';
import Home from './pages/public/Home';
import HeadTeacherClass from './pages/headteacher/HeadTeacherClass';
import HeadTeacherSchedule from './pages/headteacher/HeadTeacherSchedule';

function App() {
  const roles = useSelector(state => state.auth.roles);

  return (
    <div className="App">
      <Routes>
        <Route element={<PublicLayout />}>
          <Route path='/' exact element={<Navigate redirect to={getHomePageForUser(roles)} />}></Route>
          <Route path='/home' exact element={<Home />}></Route>
          <Route path='/login' exact element={<Login />}></Route>
          <Route path='*' element={<NotFound />}></Route>
        </Route>

        <Route element={<AuthLayout />}>
          <Route path='/student' element={<RequireAuth allowedRoles={[Roles.STUDENT]} />}>
              <Route path='schedule' element={<StudentSchedule />}></Route>
              <Route path='subjects' element={<StudentSubjects />}></Route>
              <Route path='class' element={<StudentClass />}></Route>
              <Route path='marks' element={<StudentMarks />}></Route>
          </Route>

          <Route path='/teacher' element={<RequireAuth allowedRoles={[Roles.TEACHER]} />}>
              <Route path='schedule' element={<TeacherSchedule />}></Route>
              <Route path='subjects' element={<TeacherSubjects />}></Route>
              <Route path='markBook' element={<TeacherMarkBook />}></Route>
            </Route>

          <Route path='/formteacher' element={<RequireAuth allowedRoles={[Roles.FORMTEACHER]} />}>
              <Route path='class' element={<FormTeacherClass />}></Route>
              <Route path='classMarkBook' element={<FormTeacherClassMarkBook />}></Route>
          </Route>

          <Route path='/headteacher' element={<RequireAuth allowedRoles={[Roles.HEADTEACHER]} />}>
              <Route path='class' element={<HeadTeacherClass />}></Route>
              <Route path='schedule' element={<HeadTeacherSchedule />}></Route>
          </Route>
        </Route>
      </Routes>
    </div>
  );
}

export default App;
```