

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”

Завідувач кафедри ЦТЕ

_____ Наталія АУШЕВА

“ ” _____ 2025 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності 122 “Комп’ютерні науки”

на тему: Вебзастосунок для керування та моніторингу розкладу
занять в освітній установі

Виконав (-ла):

студент (-ка) IV курсу, групи ТР-13

_____ Костюк Владислав Сергійович

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник:

доктор технічних наук

_____ професор Сліпченко Володимир Георгійович

(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові)

_____ (підпис)

Рецензент:

_____ зав. каф. ТАЕ, д.т.н

_____ професор Черноусенко Ольга Юріївна

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові)

_____ (підпис)

Нормоконтролер: _____ асистент Волков Олександр Володимирович

_____ (підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент (-ка) _____

(підпис)

Київ – 2025

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

«__» _____ 2025 р.

**ЗАВДАННЯ
на дипломну роботу студенту**

Костюку Владиславу Сергійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи **Вебзастосунок для керування та моніторингу
розкладу занять в освітній установі**

керівник роботи **д.т.н., проф. Сліпченко Володимир Георгійович**

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 202__ р.
№ _____

2. Термін подання роботи студентом **07.06.2025 року**

3. Вихідні дані до роботи мова програмування Java, фреймворки Spring та
Hibernate, СКБД PostgreSQL, середовище розробки IntelliJ Idea,
інструмент для тестування Junit та Mockito

4. Перелік питань, які потрібно розробити

1) провести аналіз серед наявних аналогів

2) сформулювати вимоги до функціональності системи

3) аргументувати вибрані інструменти, технології та алгоритми для реалізації програмного забезпечення

4) побудувати архітектуру програмного забезпечення, баз даних та сховища файлів

5) створити прикладний програмний веб-інтерфейс

6) представити процес застосування веб-інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу: діаграми, що демонструють роботу алгоритмів веб-інтерфейсу, архітектуру системи, бази даних та сховища файлів, взаємодію користувачів із системою, класи програмного забезпечення; приклади роботи з програмним продуктом.

6. Дата видачі завдання «19» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.25	
3.	Розробка архітектури та загальної структури системи	21-25.04.25	
4.	Розробка частин окремих підсистем	25.04-02.05.25	
5.	Програмна реалізація системи	03-07.05.25	
6.	Оформлення пояснювальної записки	08-12.05.25	
7.	Захист програмного продукту	13-17.05.25	
8.	Передзахист	27-29.05.25	
9.	Подання готової роботи на кафедру	07.06.2025	
10.	Захист	17-21.06.2025	

Студент

(підпис)

Владислав КОСТЮК

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)

Володимир СЛІПЧЕНКО

(ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 68 сторінках, містить 22 рисунки, 1 додаток, 24 джерел в переліку посилань.

Мета роботи — створення вебзастосунку для автоматизованого управління розкладом занять у навчальному закладі, що забезпечує зручний доступ до актуального розкладу для студентів, викладачів та адміністрації.

У роботі використано такі методи та засоби: багаторівневий архітектурний шаблон, фреймворк Spring Boot для побудови серверної частини, Java як основна мова програмування, система управління базами даних PostgreSQL, ORM-бібліотека Hibernate для взаємодії з базою даних, фронтенд реалізовано з використанням HTML, CSS, JavaScript та шаблонізатора Thymeleaf.

У дипломній роботі проведено аналіз предметної області, досліджено існуючі програмні рішення, спроектовано архітектуру вебзастосунку та структуру бази даних. Реалізовано функціонал перегляду, фільтрації й оновлення розкладу занять відповідно до ролі користувача. Вебінтерфейс забезпечує адаптивне та інтуїтивно зрозуміле представлення інформації.

Розроблений застосунок може бути впроваджений у навчальних закладах для оптимізації процесу організації навчального процесу, зменшення навантаження на адміністрацію та покращення комунікації між учасниками освітнього процесу.

ABSTRACT

The diploma thesis is presented on 68 pages, contains 22 figures, 1 appendix, and 24 sources in the reference list.

The aim of the thesis is to develop a web application for the automated management of class schedules in an educational institution, providing convenient access to up-to-date schedules for students, teachers, and administrative staff.

The following methods and tools were used in the thesis: the client-server architectural pattern, the Spring Boot framework for building the backend, Java as the main programming language, the PostgreSQL database management system, the Hibernate ORM library for database interaction, and the frontend was implemented using HTML, CSS, JavaScript, and the Thymeleaf templating engine.

The thesis includes an analysis of the subject area, a review of existing software solutions, the design of the web application's architecture and the structure of the database. Functionality has been implemented to view, filter, and update the schedule according to the user's role. The web interface provides an adaptive and user-friendly presentation of information.

The developed application can be implemented in educational institutions to optimize the organization of the educational process, reduce the administrative workload, and improve communication among participants in the educational system.

ЗМІСТ

АНОТАЦІЯ.....	4
ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	8
ВСТУП.....	9
1 ПОСТАНОВКА ЗАДАЧІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ.....	12
1.1 Постановка задачі	13
1.2 Огляд методів розв’язання задачі	15
1.3 Огляд програмних засобів	16
1.3.1 Мова програмування.....	16
1.3.2 Spring Фреймворк	19
1.3.3 База даних	21
1.3.4 Інтерфейс користувача.....	23
1.3.5 Засоби розробки та керування проектом	24
2 МЕТОДИ ТА АЛГОРИТМИ ВИРІШЕННЯ ЗАДАЧІ	26
2.1 Валідація та контроль обмежень.....	27
2.2 Оптимізація взаємодії з базою даних.....	28
2.3 Забезпечення безпеки застосунку	29
3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ.....	31
3.1 Архітектура програмної системи	31
3.2 Модель представлення даних	34
3.3 Діаграма класів.....	37
4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ	41
4.1 Сценарій роботи користувача (студента або викладача).....	41
4.2 Сценарій роботи адміністратора	45
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56

ДОДАТОК А.....	59
----------------	----

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

1. MVC (Model-View-Controller) – шаблон проектування, що розділяє застосунок на три основні компоненти: модель (дані), уявлення (інтерфейс користувача) та контролер (обробка запитів і керування логікою).
2. DTO (Data Transfer Object) – об'єкт, призначений для перенесення даних між шарами застосунку, зазвичай між сервером і клієнтом або між сервісом і контролером.
3. DAO (Data Access Object) – об'єкт доступу до даних, який інкапсулює логіку доступу до бази даних.
4. ER-діаграма (Entity-Relationship diagram) – діаграма, що зображає сутності, їхні атрибути та зв'язки між ними.
5. ID (Identifier) – унікальний ідентифікатор сутності в базі даних.
6. UI (User Interface) – інтерфейс користувача, елементи, з якими взаємодіє користувач у програмі.
7. Service – шар логіки бізнес-операцій, який обробляє дані перед їх передачею до/від репозиторіїв.
8. Repository – шар взаємодії з базою даних, який реалізує CRUD-операції (створення, читання, оновлення, видалення).
9. Mapper – компонент, що перетворює об'єкти однієї структури (наприклад, Entity) в інші (наприклад, DTO) і навпаки.
10. Controller – компонент у MVC-архітектурі, що обробляє запити від користувача та повертає відповідь (наприклад, HTML-сторінку).

ВСТУП

У сучасних навчальних закладах усе більш актуальним стає питання ефективної організації розкладу занять [1]. Традиційні методи створення та поширення розкладу — у вигляді таблиць на папері або у вигляді статичних файлів — є застарілими та незручними як для адміністрації, так і для студентів і викладачів. В умовах цифровізації освіти виникає потреба у розробці динамічних, зручних у користуванні веб-застосунків, які автоматизують процеси управління розкладом і забезпечують доступ до нього в режимі реального часу.

Застосування сучасних інформаційних технологій у сфері освіти дозволяє значно підвищити ефективність адміністрування навчального процесу. Завдяки інтерактивним веб-інтерфейсам, дані про заняття можуть бути оновлені оперативно, із миттєвим відображенням змін для всіх користувачів. Це особливо важливо в умовах, коли розклад може змінюватися через різні обставини — наприклад, заміну викладача, перенесення занять або зміну навчальних планів.

На сьогоднішній день існують подібні системи — як університетські внутрішні сервіси, так і зовнішні комерційні рішення. Проте багато з них мають обмежений функціонал, складний або неадаптивний інтерфейс, або не враховують одночасно інтереси усіх категорій користувачів. Часто також відсутня підтримка зручного редагування та гнучкого відображення змін.

У зв'язку з цим, створення сучасного, адаптивного та функціонального веб-застосунку для управління розкладом навчального закладу є важливим і актуальним напрямом у сфері освітніх технологій.

Основною метою дипломної роботи є створення вебзастосунку для управління розкладом занять у навчальному закладі, який би забезпечував зручну, швидку та надійну взаємодію між адміністрацією, викладачами та студентами. У межах цього проєкту необхідно було реалізувати рішення, здатне замінити неефективні традиційні підходи до формування та поширення розкладів, а також врахувати актуальні вимоги цифровізації освітнього процесу.

Для досягнення цієї мети були поставлені наступні завдання:

- провести глибокий аналіз предметної області, що включає вивчення специфіки організації навчального процесу, типових потреб та очікувань кінцевих користувачів (адміністрації, студентів, викладачів). Врахувати як організаційні, так і технічні аспекти роботи розкладу;

- ознайомитися з існуючими аналогами систем, зокрема тими, які використовуються у вищих навчальних закладах. Здійснити порівняльний аналіз подібних вебзастосунків із метою виявлення їхніх переваг та недоліків, що дозволить уникнути поширених помилок при розробці власного рішення;

- розробити архітектуру системи, обравши багаторівневу модель побудови програмного забезпечення. Такий підхід дозволяє ефективно розділити логіку додатку на рівень взаємодії з користувачем, бізнес-логіку та рівень збереження даних, що підвищує масштабованість і підтримуваність системи;

- спроектувати структуру бази даних з урахуванням усіх необхідних об'єктів: навчальні групи, викладачі, дисципліни, типи занять, сітка навчального часу тощо. Важливо було досягти логічної зв'язності між таблицями, уникаючи надлишковості та дублювання даних;

- реалізувати серверну частину застосунку, використовуючи мову Java з фреймворками Spring Boot та Hibernate. Це забезпечує ефективну обробку запитів, безпечну роботу з базою даних і можливість масштабування проєкту в майбутньому;

- розробити інтерфейс користувача з використанням HTML, CSS, JavaScript і шаблонізатора Thymeleaf, з урахуванням адаптивності та зручності використання;

- реалізувати основні функціональні сценарії роботи користувача, зокрема перегляд розкладу, внесення змін (для адміністратора), оновлення власних занять (для викладача), та доступ до актуальної інформації (для студентів);

- забезпечити якісне тестування системи із застосуванням фреймворків JUnit та Mockito. Це дозволяє виявити та усунути помилки ще до розгортання, що позитивно впливає на стабільність і надійність системи;

– розробити супровідну документацію, яка містить опис архітектурних рішень, технічні деталі реалізації, інструкції з розгортання та приклади сценаріїв використання.

Усі зазначені завдання спрямовані на створення повноцінного функціонального продукту, що відповідає сучасним вимогам цифровізації навчального процесу, підвищує ефективність взаємодії між адміністрацією, викладачами та студентами, а також забезпечує зручність доступу до інформації в режимі реального часу.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

У даному розділі дипломної роботи розглядаються основні аспекти, пов'язані з формулюванням задачі, яку необхідно розв'язати в рамках розробки вебзастосунку для формування та управління розкладом занять у навчальному закладі.

Метою цього розділу є глибоке дослідження суті проблеми, яку вирішує дипломна робота, а також аналіз існуючих підходів до її розв'язання. Для досягнення цієї мети у розділі розглядаються три ключові напрями: опис предметної області та постановка задачі, огляд методів, які застосовуються у подібних розробках, а також аналіз програмних засобів, що можуть бути використані при реалізації програмного забезпечення.

У підрозділі 1.1 наведено детальний опис прикладної задачі, що постає перед розробником. Тут розглянуто особливості організації навчального процесу, специфіку побудови розкладів у навчальних закладах, категорії користувачів майбутньої системи, а також визначено вхідні та вихідні дані, які обробляються у рамках системи. Постановка задачі базується на реальних потребах адміністрації навчальних закладів, викладачів і студентів, що потребують зручного, гнучкого й автоматизованого способу роботи з розкладом.

У підрозділі 1.2 здійснюється огляд методів, що застосовуються у практиці автоматизованого створення та управління розкладом. Проаналізовано сучасні алгоритми та підходи. Визначається доцільність використання певних методів з огляду на поставлену задачу.

У підрозділі 1.3 проведено аналіз сучасних інструментів розробки програмного забезпечення, таких як мови програмування, фреймворки, бази даних, засоби побудови вебінтерфейсів, бібліотеки та сервіси, які можуть бути інтегровані у проєкт. Особливу увагу приділено порівнянню можливостей різних технологій та вибору оптимальних рішень для реалізації функціоналу системи, що відповідає критеріям надійності, масштабованості та зручності для кінцевого користувача.

1.1 Постановка задачі

У сучасному освітньому середовищі ефективне управління розкладом навчальних занять є однією з ключових задач для адміністративного персоналу навчальних закладів. Університети, коледжі та інші освітні установи стикаються з необхідністю врахування численних факторів під час створення розкладу: наявність викладачів, розподіл навчальних груп, доступність аудиторного фонду, особливості навчального плану, а також інші обмеження, які ускладнюють процес ручного складання розкладу.

Традиційні способи створення розкладу за допомогою таблиць у текстових редакторах чи електронних таблиць є застарілими та неефективними. Вони не передбачають автоматичної перевірки на конфлікти, не дають змоги зручно оновлювати або переглядати зміни, не забезпечують гнучкої взаємодії між усіма категоріями користувачів. У зв'язку з цим виникає потреба у розробці сучасного вебзастосунку, який автоматизує процес створення, редагування, зберігання та відображення розкладу занять.

Метою поставленої задачі є створення сучасної веборієнтованої інформаційної системи для управління розкладом занять у навчальному закладі. Розроблений застосунок має забезпечити комплексний підхід до організації навчального процесу, зручність у користуванні та доступність для всіх категорій користувачів.

Основними завданнями, які має реалізувати система, є:

- надання інструментів для гнучкого створення, редагування та оновлення розкладу занять з урахуванням великої кількості обмежень: зайнятості викладачів, навантаження груп, навчальної сітки тощо;
- забезпечення цілодобового доступу до актуального розкладу в онлайн-режимі для користувачів з будь-яких пристроїв, включаючи мобільні;
- реалізація адаптивного інтерфейсу, який автоматично підлаштовується під параметри екрану, що значно підвищує зручність користування;

– можливість чіткого розмежування ролей та функціоналу для адміністрації, викладачів і студентів, що дозволяє ефективно організувати роботу кожної категорії користувачів в межах єдиної системи.

Користувачі системи поділяються на кілька категорій:

- адміністратор (створює, редагує та затверджує розклад, додає нові дисципліни, викладачів, групи, аудиторії тощо);
- викладач (переглядає свій особистий розклад, отримує повідомлення про зміни, за потреби може ініціювати коригування);
- студент (переглядає розклад своєї навчальної групи, отримує сповіщення про зміни).

Вхідна інформація, що обробляється системою:

- список навчальних груп;
- перелік викладачів;
- набір навчальних дисциплін;
- розклад навчального часу (сітка);
- список доступних аудиторій;
- зв'язки між дисциплінами, групами, викладачами й приміщеннями.

Вихідна інформація, що генерується системою:

- повний розклад занять, який відображається у вигляді таблиць або інтерактивних вебсторінок;
- фільтрація розкладу за викладачем, навчальною групою або днем тижня;
- інформаційні повідомлення для користувачів про оновлення.

Обмеження, які враховуються при побудові розкладу:

- один викладач не може одночасно проводити кілька занять;
- одна група не може відвідувати кілька пар одночасно;
- один розклад не може містити більше ніж одне заняття на зазначений час.

Таким чином, задача полягає у розробці багатфункціонального вебзастосунку, який дозволить значно спростити та автоматизувати процес управління розкладом занять у навчальному закладі, підвищивши точність,

прозорість та оперативність оновлення інформації для всіх учасників освітнього процесу.

1.2 Огляд методів розв’язання задачі

Для ефективної розробки вебзастосунку, що автоматизує процес керування розкладом навчального закладу, необхідно визначити та обґрунтувати методи, які будуть використані для вирішення поставлених задач. У цьому підрозділі розглядаються підходи, що дозволяють забезпечити коректне формування розкладу, зручну взаємодію користувача з інтерфейсом та стабільну роботу системи на різних рівнях.

Передусім слід зазначити, що задача автоматизованого створення розкладу належить до класу задач з обмеженнями (constraint satisfaction problems — CSP). У ній необхідно врахувати значну кількість параметрів: навчальне навантаження викладачів, час занять, розклад за дисциплінами тощо.

Крім того, важливо забезпечити швидку обробку та збереження даних. Для цього застосовуються сучасні підходи до роботи з реляційними базами даних, зокрема використання ORM (Object-Relational Mapping) рішень. В межах цього проєкту було обрано Hibernate — один із найпоширеніших ORM-фреймворків для мови Java, який дозволяє ефективно реалізовувати зв’язки між сутностями, скорочуючи кількість коду для обробки запитів до БД.

Для створення стабільного вебінтерфейсу використовується MVC-архітектура (Model-View-Controller), що дозволяє розділити бізнес-логіку, представлення та керування даними в системі. Це значно спрощує супровід проєкту та полегшує тестування. Як рушій серверної частини виступає Spring Framework — потужний інструмент для розробки на Java, що підтримує модульність, інверсію управління, безпеку та масштабованість.

З боку користувача важливим є забезпечення інтуїтивного та зручного інтерфейсу. Для цього застосовуються технології веброботи: HTML, CSS,

JavaScript, а також шаблонізатор Thymeleaf, який дозволяє вбудовувати логіку представлення безпосередньо у HTML-файли. Це забезпечує узгодженість між даними з серверної частини та тим, як вони відображаються у браузері.

Таким чином, поєднання методів генерації розкладу, сучасних засобів обробки даних і підходів до побудови вебінтерфейсу дозволяє реалізувати комплексну програмну систему, що задовольняє вимоги всіх категорій користувачів і адаптується до умов функціонування реального навчального закладу.

1.3 Огляд програмних засобів

Для реалізації веборієнтованої системи управління розкладом занять навчального закладу було обрано набір сучасних та перевірених технологій, які забезпечують ефективність, масштабованість, безпеку та зручність у подальшій підтримці програмного продукту. У цьому підрозділі розглянемо основні програмні засоби, що використовуються на кожному з етапів розробки застосунку.

1.3.1 Мова програмування

У процесі розробки веборієнтованої системи для управління розкладом занять навчального закладу мова програмування відіграє ключову роль, адже від її вибору залежить не лише швидкість створення програмного забезпечення, а й його масштабованість, підтримуваність, безпечність і продуктивність у реальному середовищі експлуатації. У межах даного проєкту як основну мову програмування обрано Java — об'єктно-орієнтовану мову, яка вже протягом десятиліть залишається однією з найпопулярніших і найнадійніших мов у галузі розробки корпоративного програмного забезпечення.

Серед основних переваг Java [2], що стали вирішальними при її виборі для реалізації дипломного проєкту, можна виділити такі:

- платформна незалежність. Java-програми компілюються у байт-код, який виконується на віртуальній машині Java (JVM), що забезпечує можливість запуску застосунку на будь-якій операційній системі без необхідності перекомпіляції;
- широка екосистема та підтримка. Java має надзвичайно розвинену екосистему бібліотек, фреймворків та інструментів, які значно полегшують і прискорюють розробку. Окрім того, мільйони розробників по всьому світу створюють і підтримують open-source рішення, що дозволяє оперативно знаходити готові компоненти для інтеграції;
- безпечність. Мова має вбудовані механізми для забезпечення безпеки, зокрема обмеження доступу до ресурсів, контроль за виконанням коду, захист від переповнення буфера тощо;
- масштабованість і продуктивність. Java дозволяє створювати високонавантажені серверні застосунки, які стабільно функціонують у середовищі з великою кількістю одночасних користувачів, що є особливо важливим для систем із розподіленим доступом, таких як вебзастосунки для навчальних закладів;
- зворотна сумісність. Java підтримує сумісність між версіями, що дозволяє оновлювати бібліотеки та інструменти без необхідності радикальної переробки коду.

Використання Java є доцільним у даному проєкті. Розробка вебзастосунку для керування розкладом вимагає стабільного серверного середовища, здатного обробляти численні запити, взаємодіяти з базами даних, забезпечувати безпечну аутентифікацію користувачів та гнучке масштабування системи. Java ідеально відповідає цим вимогам, зокрема завдяки інтеграції зі Spring Framework, який дозволяє реалізувати архітектуру REST API, гнучку бізнес-логіку та зручне керування залежностями.

Окрім того, Java є мовою, що активно використовується в освітніх, урядових та корпоративних проєктах, а отже, вибір цієї мови забезпечує відповідність сучасним стандартам галузі.

Також варто зазначити, що Java є однією з найпопулярніших мов програмування у світі [3] та стабільно входить до трійки лідерів за рівнем попиту серед роботодавців (рисунок 1.3.1.1).

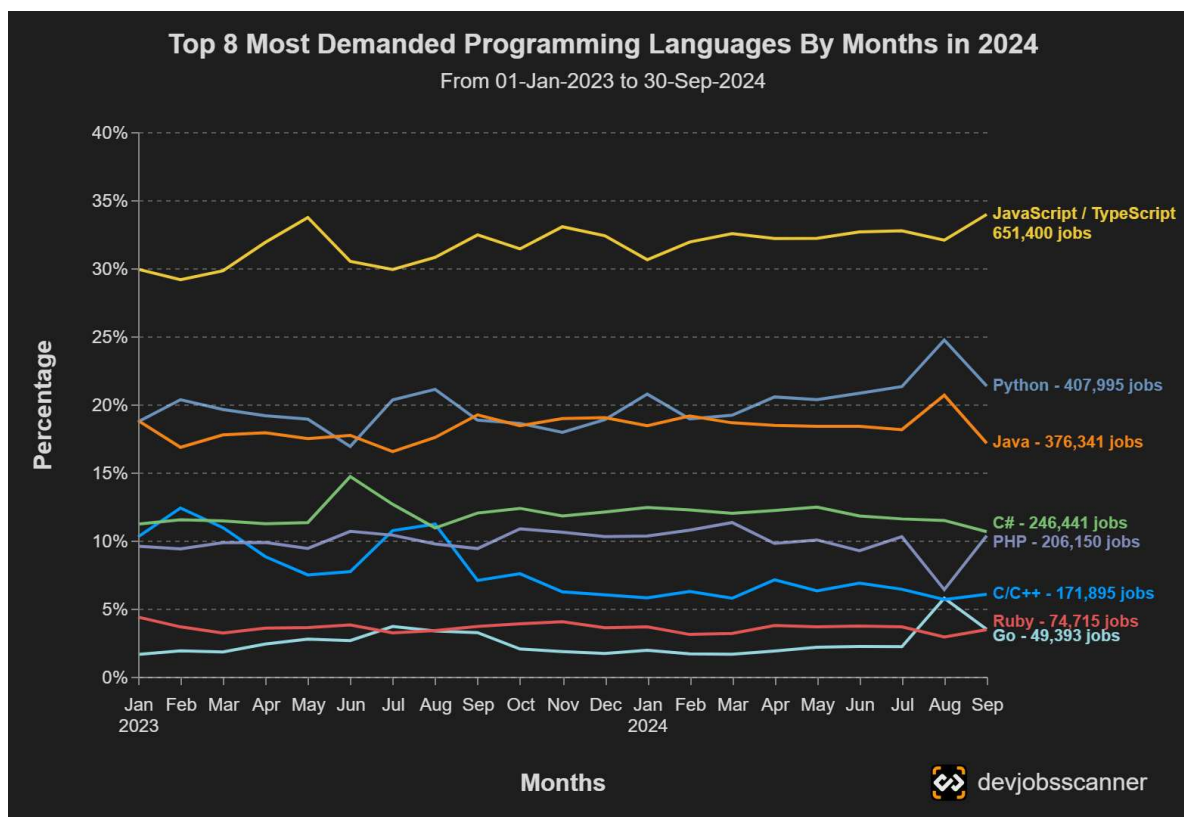


Рисунок 1.3.1.1 – Статистика попиту мов програмування в 2024 році

Порівняймо Java з кількома іншими популярними мовами програмування [4], які теоретично могли бути використані в даному проєкті:

- Python. Ця мова відзначається простотою синтаксису та високою швидкістю розробки. Проте у великих масштабованих системах Python часто поступається Java в плані продуктивності. Крім того, вбудовані механізми безпеки й обробки багатопотоковості в Python менш ефективні, що робить її менш бажаною для критичних вебзастосунків;

- JavaScript (Node.js). Node.js дозволяє писати серверний код JavaScript-ом і має свої переваги, особливо в системах з великою кількістю одночасних з'єднань. Однак Java пропонує значно стабільніше середовище з перевіреною практикою

розробки, а також є більш придатною для роботи з потужними фреймворками на кшталт Spring;

- C#. Ця мова дуже схожа на Java за стилем та архітектурною філософією. Проте C# тісно пов'язаний із платформою .NET і Windows, що ускладнює використання в кросплатформних середовищах. Java ж залишається універсальною і незалежною від ОС.

Попри значну кількість переваг, Java має і певні недоліки [5]:

- вищий поріг входу. У порівнянні з Python чи JavaScript, Java вимагає більш глибокого розуміння об'єктно-орієнтованого підходу та структури додатків;
- відносно більший обсяг коду. Для реалізації простих функцій Java зазвичай потребує більше рядків коду, що іноді уповільнює розробку;
- використання пам'яті. JVM може споживати більше пам'яті у порівнянні з нативними рішеннями, особливо у невеликих проєктах.

Проте для задачі, пов'язаної з побудовою надійної, гнучкої та масштабованої інформаційної системи, зазначені недоліки є незначними і не впливають критично на кінцевий результат.

Враховуючи всі переваги, Java є оптимальним вибором для реалізації серверної частини дипломного проєкту. Її універсальність, безпека, масштабованість і широка підтримка в професійному середовищі роблять її ідеальним інструментом для створення сучасних вебсистем, що задовольняють потреби користувачів у сфері освіти.

1.3.2 Spring Фреймворк

Одним із ключових компонентів у реалізації дипломної роботи є використання Spring Framework [6] – потужного, гнучкого та широко розповсюдженого фреймворку для створення корпоративних Java-додатків. Його застосування у сучасній розробці веборієнтованих програмних систем є не лише виправданим, а й рекомендованим у професійному середовищі, зважаючи на численні переваги, які він надає розробнику.

Spring Framework базується на принципах інверсії керування [7] (Inversion of Control – IoC) та аспектно-орієнтованого програмування (Aspect-Oriented Programming – AOP), що забезпечує високу гнучкість, модульність і масштабованість програмного забезпечення. Однією з основних його переваг є контейнер IoC, який керує життєвим циклом об'єктів, сприяє слабкому зв'язуванню між компонентами системи та спрощує тестування і рефакторинг коду.

Spring має модульну структуру, яка дозволяє обрати лише ті компоненти, що дійсно потрібні для конкретного проєкту. У межах розробки вебзастосунку для формування та перегляду розкладу занять особливе значення мають наступні підмодулі Spring:

- Spring Boot. Спрощує налаштування та запуск застосунку, дозволяє швидко створювати production-ready рішення з мінімальною кількістю конфігурацій;

- Spring MVC. Забезпечує реалізацію шаблону Model-View-Controller, що дозволяє ефективно розділити бізнес-логіку, інтерфейс користувача та обробку запитів;

- Spring Data JPA. Спрощує взаємодію з базою даних за допомогою об'єктно-реляційного відображення (ORM), що у поєднанні з Hibernate дозволяє ефективно працювати з великою кількістю даних;

- Spring Security. Відповідає за автентифікацію та авторизацію користувачів, забезпечуючи безпеку системи.

Застосування Spring забезпечує швидкість розробки, уніфіковану структуру проєкту, зниження кількості помилок, а також підтримує інтеграцію з іншими популярними технологіями, зокрема, з базами даних (PostgreSQL), системами логування, сторонніми API тощо.

У порівнянні з іншими Java-фреймворками, таким як Jakarta EE (раніше Java EE, Spring має нижчий поріг входу, ширшу спільноту підтримки, кращу документацію, а також активно підтримується та розвивається великою кількістю ентузіастів і компаній-розробників. На відміну від монолітного підходу Jakarta EE, Spring дозволяє реалізувати більш гнучкі мікросервісні архітектури, що особливо цінно у масштабованих системах.

Недоліками Spring можна вважати доволі високий рівень абстракції, який може ускладнювати розуміння внутрішніх процесів новачками, а також можливе зростання складності конфігурації у великих проєктах. Проте, використання Spring Boot значною мірою нівелює ці недоліки, автоматизуючи більшість налаштувань.

Таким чином, вибір Spring Framework для реалізації вебзастосунку розкладу занять є технічно обґрунтованим. Завдяки своїй надійності, масштабованості та активній підтримці, він дозволяє створити ефективне, безпечне та зручне у підтримці програмне забезпечення.

1.3.3 База даних

У межах створення вебзастосунку для формування та перегляду розкладу занять особливу роль відіграє система керування базами даних (СКБД), яка відповідає за зберігання, обробку та доступ до структурованої інформації. У даній дипломній роботі для реалізації серверної частини застосунку було обрано PostgreSQL — потужну об'єктно-реляційну СКБД [8] з відкритим вихідним кодом, яка широко використовується в комерційних і наукових розробках завдяки своїй стабільності, гнучкості та широкому функціоналу.

PostgreSQL надає розвинені можливості роботи з реляційними даними, а також підтримує складні типи даних, збережені процедури, тригери, повнотекстовий пошук, JSON-типи та інші сучасні механізми, які роблять її конкурентоспроможною порівняно з іншими СКБД, як-от MySQL, Oracle чи Microsoft SQL Server.

Однією з ключових переваг PostgreSQL є строге дотримання стандарту SQL, що забезпечує високу переносимість і мінімізацію помилок при роботі з транзакціями, обмеженнями цілісності та складними запитами. У контексті даного проєкту це дозволяє ефективно моделювати структуру розкладу, що включає такі сутності, як: пари, викладачі, аудиторії, групи, дисципліни та інші пов'язані об'єкти.

Основні переваги використання PostgreSQL [9]:

- масштабованість. PostgreSQL підтримує роботу з великими обсягами даних, що дозволяє використовувати її як для невеликих систем, так і для проєктів, які мають потенціал до розширення;
- розширюваність. Система дозволяє створювати користувацькі типи даних, оператори та функції, що забезпечує високу гнучкість при реалізації специфічних бізнес-вимог;
- надійність і цілісність даних. Підтримка ACID-транзакцій (атомарність, узгодженість, ізолюваність, довговічність) гарантує правильність змін даних навіть при одночасному доступі кількох користувачів;
- безпека. PostgreSQL має розвинені механізми контролю доступу, шифрування, аутентифікації та аудиту, що дозволяє реалізувати розмежування прав користувачів (студенти, викладачі, адміністрація) у межах застосунку;
- активна спільнота та документація. PostgreSQL має широке ком'юніті, регулярні оновлення та високоякісну технічну документацію, що сприяє швидкому вирішенню проблем та оптимізації продуктивності.

У порівнянні з MySQL, PostgreSQL демонструє кращу підтримку складних SQL-операцій, таких як рекурсивні запити, віконні функції, CTE (Common Table Expressions), що може бути критично важливим при формуванні звітів або реалізації динамічного фільтрування розкладу. Крім того, PostgreSQL має краще впроваджену систему контролю транзакцій та дозволяє реалізовувати складну логіку на стороні бази даних без необхідності дублювання коду в бекенді.

Також важливим фактором стало повноцінне ORM-підтримання [10] PostgreSQL у Spring Data JPA, що дозволило органічно інтегрувати її у програмну архітектуру проєкту без зайвих налаштувань. Завдяки цьому сутності проєкту були легко спроектовані у вигляді JPA-об'єктів, які автоматично відображаються на відповідні таблиці у базі даних, що пришвидшило процес розробки.

Під час розробки також була побудована логічна модель бази даних яка охоплює всі ключові компоненти системи — користувачів, ролі, заняття, навчальні групи, викладачів та предмети. Зв'язки між таблицями реалізовані через зовнішні

ключі, що дозволяє забезпечити цілісність даних і підтримувати гнучку структуру, яку можна масштабувати у майбутньому.

Таким чином, вибір PostgreSQL як основної СКБД для веборієнтованого застосунку управління розкладом є цілком виправданим. Завдяки широким функціональним можливостям, високій продуктивності, надійності та сумісності з обраним технічним стеком, ця база даних забезпечує стабільну основу для ефективного функціонування всієї системи.

1.3.4 Інтерфейс користувача

Інтерфейс користувача (англ. User Interface, UI) є одним із ключових компонентів будь-якого сучасного вебзастосунку, оскільки саме через нього здійснюється взаємодія користувача із системою. У межах створення веборієнтованої системи для формування та перегляду розкладу занять особлива увага приділялася як функціональності інтерфейсу, так і його зручності, доступності та адаптивності. Від якісно реалізованого UI залежить загальне враження від використання програмного забезпечення, ефективність виконання користувацьких дій та рівень задоволеності цільової аудиторії.

Під час проєктування інтерфейсу було використано низку сучасних принципів:

- простота. Інтерфейс має мінімалістичний вигляд і не перевантажений зайвими елементами, що дозволяє користувачеві зосередитися на виконанні конкретного завдання;
- інтуїтивність. Розташування елементів, позначення кнопок і логіка переходів між сторінками відповідають очікуванням більшості користувачів;
- адаптивність. Завдяки використанню адаптивної верстки інтерфейс коректно відображається на різних типах пристроїв (ПК, планшетах, смартфонах);
- уніфікованість. Усі частини системи мають спільний стиль оформлення, що забезпечує цілісність UI і спрощує сприйняття інформації.

Для реалізації користувацького інтерфейсу застосовано HTML5, CSS3, JavaScript, а також фреймворк Bootstrap, що дозволяє швидко і ефективно створювати адаптивні інтерфейси з урахуванням сучасних вимог до дизайну.

У деяких частинах застосунку також використовуються можливості AJAX для асинхронної взаємодії з сервером, що дозволяє динамічно оновлювати дані без повного перезавантаження сторінки, покращуючи загальну продуктивність і зручність.

Для більш структурованої та масштабованої реалізації UI можливо подальше використання фреймворків на кшталт React або Angular, однак для цілей цієї роботи було вирішено обрати класичний підхід з використанням шаблонізатора, наприклад, Thymeleaf, який добре інтегрується із Spring та дозволяє будувати сторінки без надмірної складності.

1.3.5 Засоби розробки та керування проєктом

У процесі створення програмного забезпечення для автоматизованого формування та відображення розкладу занять важливу роль відіграють не лише мови програмування чи фреймворки, але й інструменти, які забезпечують ефективну розробку, тестування, налагодження та підтримку проєкту. До таких інструментів належать середовище розробки, системи керування версіями, засоби збірки та тестування, а також платформи для командної взаємодії. Використання сучасних засобів розробки дозволяє суттєво підвищити якість програмного продукту, пришвидшити розробку і забезпечити її прозорість на кожному етапі.

Основним середовищем розробки (IDE), що використовувалося у процесі роботи, стала IntelliJ IDEA – один із найпотужніших та найпопулярніших інструментів для розробки на мові Java. Ця IDE підтримує інтеграцію зі Spring Framework, має вбудовану підтримку роботи з Git, засоби налагодження, запуску тестів, автодоповнення коду, а також можливості для рефакторингу.

Для контролю версій використовувалась система Git [11]. Вона дозволяє ефективно відстежувати зміни у проєкті, створювати гілки для окремих функціональностей, а також забезпечує зручну співпрацю при колективній роботі над проєктом. У поєднанні з Git було використано хостинг GitHub, який надає вебінтерфейс для перегляду репозиторіїв, ведення документації, обговорення задач (issues), створення pull-запитів тощо.

Основні переваги використання Git:

- історія змін з можливістю відкотити попередні версії;
- паралельна робота над різними частинами проєкту;
- прозорість командної роботи;
- інтеграція з CI/CD інструментами.

Для автоматизації процесу збирання проєкту та управління зовнішніми залежностями використовувався Apache Maven [12] — одна з найпоширеніших систем збирання для Java-проєктів. Maven дозволяє централізовано описувати конфігурацію проєкту, автоматично завантажувати необхідні бібліотеки, керувати життєвим циклом збирання та виконання тестів.

Проєкт був структурований за принципами модульності та логічного поділу відповідальностей. Основні частини системи (бекенд, фронтенд, база даних) були організовані окремо з дотриманням загальноприйнятих архітектурних підходів (MVC). Код регулярно документувався, велась історія змін та контроль за коректною роботою всіх частин проєкту.

2 МЕТОДИ ТА АЛГОРИТМИ ВИРІШЕННЯ ЗАДАЧІ

У цьому розділі представлено сукупність методів, алгоритмів та технічних рішень, що були використані при реалізації вебзастосунку для формування та керування розкладом навчального процесу. Основна увага приділяється реалізації ключових функціональних частин системи — від побудови логіки розкладу до валідації введених даних, ефективної роботи з базою даних та відображення інформації на інтерфейсі користувача.

У підрозділі 2.1 здійснено опис механізмів перевірки коректності розкладу після його формування або редагування. Зокрема, реалізовано перевірку на предмет дублювання занять в один і той самий час у викладача, групи або аудиторії. Також здійснюється контроль відповідності тривалості занять встановленим нормам, дотримання часових рамок навчального дня та відсутність конфліктів між подіями. Валідація виконується як на стороні серверу (бекенду), так і частково на клієнтському рівні, що дозволяє запобігти помилкам ще до надсилання даних у базу.

У підрозділі 2.2 проведено аналіз структури запитів до бази даних та методів їх оптимізації. Розглядаються аспекти нормалізації таблиць, побудови ефективних індексів, використання зв'язків між сутностями через зовнішні ключі, а також застосування транзакцій для забезпечення узгодженості даних. Особлива увага приділена підходам до кешування та використанню проєкцій у Spring Data JPA для зменшення обсягу передаваних даних між базою та застосунком. У результаті реалізовано стабільну та продуктивну взаємодію з базою PostgreSQL, що забезпечує швидкий відгук системи навіть при великому обсязі даних.

У підрозділі 2.3 здійснюється аналіз заходів безпеки, реалізованих у вебзастосунку для захисту даних користувачів і стабільності системи. Наведено основні підходи до запобігання типових вебзагроз, таких як SQL-ін'єкції, міжсайтові скриптові атаки (XSS), міжсайтове підроблення запитів (CSRF). Також розглянуто використання шифрування паролів за допомогою алгоритму BCrypt, налаштування безпечної передачі даних через HTTPS, конфігурацію CORS та

забезпечення роботи з безпечними cookie. Застосування цих методів сприяє підвищенню загального рівня надійності та безпеки застосування.

2.1 Валідація та контроль обмежень

У даному підрозділі здійснюється детальний розгляд механізмів, які забезпечують перевірку коректності даних під час формування та редагування розкладу. Надійна валідація – це один з ключових етапів побудови програмного забезпечення, особливо коли йдеться про системи, що працюють з великою кількістю взаємопов'язаних елементів та обмежень, таких як навчальний розклад.

Основна мета валідації полягає у виявленні потенційних конфліктів, логічних суперечностей та порушень заданих обмежень ще на етапі створення або оновлення даних, щоб уникнути ситуацій, у яких кілька подій накладаються в часі або використовують одні й ті самі ресурси.

У процесі розробки вебзастосування було реалізовано багаторівневу систему контролю обмежень, яка працює як на клієнтській, так і на серверній стороні. Це дозволяє підвищити ефективність перевірки та зменшити навантаження на сервер, виконуючи частину валідацій одразу в браузері користувача. Серед основних перевірок, які виконує система:

- унікальність подій у межах одного часового слоту. Перевіряється, щоб в один і той самий час в одній не проходило більше одного заняття, а викладач або студентська група не були призначені одночасно на кілька подій;
- дотримання меж робочого часу. Заняття не можуть починатися раніше або закінчуватися пізніше за допустимі години роботи закладу;
- коректність посилань між сутностями. Перевіряється наявність та активний статус пов'язаних об'єктів (наприклад, чи існує викладач, до якого прив'язується пара).

Важливо, що всі перевірки реалізовані як через вбудовані інструменти Spring (наприклад, за допомогою Java Bean Validation API), так і через кастомні сервіси, що враховують специфіку предметної області.

У разі виявлення порушення обмежень система формує детальні повідомлення про помилки, що дозволяє користувачеві оперативно внести корективи. Таким чином, реалізована система валідації дозволяє не лише зберігати логічну цілісність даних, а й суттєво підвищує зручність роботи користувачів, сприяючи створенню надійного й точного розкладу.

2.2 Оптимізація взаємодії з базою даних

У цьому підрозділі розглядаються підходи до оптимізації взаємодії з базою даних, що використовувалися під час розробки веборієнтованого застосунку для управління розкладом. Ефективна організація доступу до даних є критично важливою складовою функціонування будь-якої інформаційної системи, особливо тієї, яка має справу з великою кількістю зв'язаних об'єктів, запитів та одночасних користувачів.

В основі проєкту використовується система управління базами даних PostgreSQL, яка обрана завдяки своїй високій продуктивності, стабільності та підтримці складних зв'язків між таблицями. Але навіть при використанні сучасного СУБД необхідно впроваджувати відповідні оптимізаційні практики для забезпечення швидкої обробки запитів, мінімізації затримок і раціонального використання ресурсів.

Основні аспекти оптимізації, реалізовані в межах проєкту:

- використання ORM-технології (Hibernate/JPA): Для абстракції взаємодії з базою даних у проєкті використовується ORM-бібліотека Hibernate разом з Spring Data JPA. Це дозволяє зменшити кількість SQL-запитів, що пишуться вручну, та забезпечити автоматичне керування транзакціями, кешуванням та зв'язками між сутностями;

- нормалізація структури БД: Схема бази даних проєкту розроблена з урахуванням принципів нормалізації [13]. Кожна сутність (наприклад, заняття, викладач або група студентів) представлена окремою таблицею з чітко визначеними зв'язками. Це дозволяє уникнути дублювання даних та спростити підтримку цілісності інформації;

- ліниве завантаження (Lazy Loading). Для зв'язаних об'єктів використовується підхід лінивого завантаження, який дозволяє уникати непотрібного витягування великих об'ємів даних [14] при виконанні запитів. Наприклад, під час отримання інформації про викладача система не витягує автоматично всі пов'язані заняття, якщо це не потрібно в даному контексті;

- кешування результатів. Для запитів, які часто повторюються (наприклад, отримання розкладу для конкретної групи), використовується механізм кешування на рівні сервісів, що дозволяє зменшити кількість звернень до бази даних та пришвидшити час відгуку інтерфейсу користувача;

- транзакційність. Для запобігання порушенням цілісності даних усі критичні операції (додавання, редагування, видалення) відбуваються в межах транзакцій [15]. Це гарантує, що навіть у разі помилки система повернеться у стабільний стан без втрати даних;

Результатом впровадження цих практик стало підвищення загальної продуктивності застосунку, зменшення часу відгуку на ключові запити та підвищення масштабованості системи, що особливо важливо в умовах зростання кількості користувачів або розширення функціоналу.

2.3 Забезпечення безпеки застосунку

Безпека є критично важливим аспектом для будь-якої інформаційної системи, особливо тієї, що працює в мережі Інтернет та зберігає персональні або службові дані користувачів. Під час розробки програмного забезпечення було

враховано низку потенційних загроз, характерних для вебдодатків, та реалізовано відповідні заходи захисту.

Однією з ключових реалізованих практик є захист від SQL-ін'єкцій [16]. Усі запити до бази даних виконуються з використанням підготовлених виразів (prepared statements), які дозволяють відокремити SQL-логіку від користувацьких даних, унеможливаючи ін'єкцію шкідливих конструкцій. Завдяки цьому забезпечується цілісність і захищеність даних у базі.

Забезпечення надійного зберігання облікових даних реалізовано шляхом шифрування паролів користувачів з використанням алгоритму BCrypt, який забезпечує захист від атак перебору (brute-force) завдяки адаптивності алгоритму хешування. Навіть у випадку компрометації бази даних, зловмисник не отримає доступ до справжніх паролів користувачів.

Для безпечної передачі інформації між клієнтом і сервером застосовується HTTPS-протокол, який забезпечує шифрування переданих даних та захист від атак.

Також було враховано налаштування CORS (Cross-Origin Resource Sharing) для забезпечення контрольованого доступу до ресурсів з інших доменів. Дозволено лише необхідні джерела, що мінімізує ризики пов'язані з небажаними міждоменними запитами.

Таким чином, у межах розробки застосунку було реалізовано низку комплексних заходів безпеки, спрямованих на захист даних, стабільність роботи системи та збереження довіри користувачів. Системний підхід до захисту інформації дозволяє зменшити ризики, пов'язані з найбільш поширеними загрозами у вебсередовищі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У третьому розділі наведено опис програмної реалізації вебзастосунку, розробленого в межах дипломної роботи. Розділ містить детальний розгляд архітектури системи, способу представлення та зберігання даних, а також внутрішньої логіки взаємодії об'єктів у вигляді діаграми класів. Такий підхід дозволяє чітко визначити взаємозв'язки між компонентами системи, забезпечити масштабованість, підтримуваність і розширюваність проєкту.

У підрозділі 3.1 наведено опис архітектурної моделі програмної системи, яка базується на принципах багаторівневої клієнт-серверної взаємодії.

У підрозділі 3.2 здійснюється опис логічної моделі даних, яка є основою для організації та зберігання інформації у застосунку. Основна увага приділена структурам сутностей, які відображають реальні об'єкти предметної області – користувачів, розклади, заняття, групи, викладачів тощо.

У підрозділі 3.3 наведено діаграму класів, яка ілюструє об'єктно-орієнтовану структуру застосунку, а також демонструє архітектуру застосунку та взаємозв'язки між його основними компонентами.

3.1 Архітектура програмної системи

Під час розробки програмного забезпечення для автоматизованого управління розкладом занять навчального закладу було реалізовано багаторівневу архітектуру [17], яка забезпечує розділення відповідальностей, гнучкість масштабування, зручність тестування та супроводу. Архітектура складається з таких основних рівнів:

- рівень представлення (Presentation Layer);
- рівень логіки додатка (Business Logic Layer);
- рівень доступу до даних (Data Access Layer).

Рівень представлення (Presentation Layer) — реалізований за допомогою HTML, CSS, JavaScript і серверного шаблонізатора Thymeleaf. Відповідає за взаємодію з користувачем, відображення даних і передачу запитів до серверної частини. Користувач бачить адаптивний вебінтерфейс, через який може переглядати розклад, шукати заняття за викладачем або групою, додавати нові дані (для адміністратора тощо).

Рівень логіки додатка (Business Logic Layer) — побудований з використанням Spring Framework. Саме тут зосереджено логіку обробки запитів користувача, перевірки правил (наприклад, коректність часу, доступність ресурсів) і взаємодії між сутностями (студенти, викладачі, предмети, групи).

Рівень доступу до даних (Data Access Layer) — реалізовано з використанням Hibernate, що дозволяє ефективно перетворювати Java-об'єкти на таблиці бази даних (PostgreSQL) і навпаки. Всі запити до бази даних інкапсульовані в репозиторіях (DAO), що дотримується принципу єдиного обов'язку.

Також слід звернути увагу на переваги та недоліки багаторівневої архітектури [18] і оцінити чи підходить ця архітектура для нашого застосунку.

Переваги обраної багаторівневої архітектури:

- чітке розділення відповідальностей між компонентами спрощує підтримку коду та дозволяє легко змінювати або розширювати окремі частини системи;
- гнучкість масштабування. Можна масштабувати окремі шари незалежно (наприклад, підключити frontend-фреймворк або відокремити базу даних);
- зручність тестування. Рівні логіки та даних легко піддаються юніт- та інтеграційному тестуванню з використанням JUnit і Mockito;
- безпека та контроль доступу. Spring дозволяє централізовано реалізувати обмеження прав доступу, автентифікацію та авторизацію.

Недоліки обраної багаторівневої архітектури:

- початкове ускладнення структури проєкту, особливо для малих застосунків, через необхідність підтримки додаткових рівнів;
- зниження продуктивності при надмірному поділі на шари, якщо не оптимізовано запити та кешування;

– підвищена складність розгортання в разі переходу на мікросервісну або розподілену модель.

Застосування багаторівневої архітектури у розробці даного застосунку є обґрунтованим, оскільки вона забезпечує чітке розділення відповідальностей, гнучкість масштабування та зручність тестування, що є критично важливим для підтримки та подальшого розвитку проєкту. Використання Spring Security також дозволило ефективно реалізувати механізми безпеки, такі як автентифікація та контроль доступу.

Проте варто враховувати недоліки цього підходу, зокрема початкову складність структури та потенційне зниження продуктивності при надмірному розшаруванні. Для невеликих застосунків це може бути надлишковим, але у даному випадку, з огляду на можливість майбутнього розширення функціоналу, архітектура виправдовує себе.

Таким чином, обраний підхід є оптимальним, оскільки поєднує модульність, безпеку та легкість підтримки, що є ключовими вимогами до сучасного програмного забезпечення.

Також варто звернути увагу, що реалізація здійснена за принципами SOLID та чистої архітектури, з чітким розмежуванням між шарами застосунку. Для зменшення залежностей між компонентами активно застосовується інверсія керування (IoC) та впровадження залежностей (Dependency Injection), які реалізуються через анотації Spring. До того ж у застосунку реалізовано низку перевірених шаблонів проєктування:

- Model-View-Controller (MVC);
- Repository Pattern;
- DTO Pattern;
- Singleton;
- Factory / Builder.

Model-View-Controller (MVC) [19] – один із найбільш розповсюджених архітектурних шаблонів, який передбачає розділення застосунку на три основні компоненти: Model (модель – бізнес-логіка та дані), View (подання – інтерфейс

користувача) та Controller (контролер – обробка запитів і керування логікою взаємодії між View і Model). У Spring цей патерн реалізується завдяки механізмам контролерів, сервісів та шаблонів подання.

Repository Pattern – шаблон, який забезпечує абстракцію між бізнес-логікою та шаром доступу до даних. Репозиторії інкапсулюють усю взаємодію з базою даних і дозволяють сервісам працювати з об'єктами, не знаючи деталей SQL-запитів. У Spring цей патерн реалізується через інтерфейси JpaRepository чи CrudRepository.

DTO (Data Transfer Object) Pattern [20] – патерн, що дозволяє передавати тільки необхідні дані між різними шарами застосунку або між клієнтом і сервером. Це зменшує обсяг переданої інформації, підвищує безпеку (приховуючи зайві поля) і спрощує структуру запитів та відповідей.

Singleton [21] – шаблон, який гарантує, що клас матиме лише один екземпляр, і надає до нього глобальну точку доступу. У Spring цей патерн реалізується автоматично.

Factory / Builder [22] – шаблони, що використовуються для створення складних об'єктів. Factory дозволяє створювати об'єкти без прив'язки до конкретного класу, що зручно при реалізації залежностей через інтерфейси. Builder використовується для поетапної побудови об'єкта з великою кількістю параметрів, покращуючи читаність та керованість коду.

3.2 Модель представлення даних

Далі буде наведено опис моделі представлення даних, яка використовується у розробленому застосунку для організації, зберігання та обробки інформації. Структура бази даних реалізована на основі реляційної моделі з урахуванням нормалізації для уникнення надмірного дублювання даних, забезпечення логічної цілісності та підвищення ефективності запитів (рисунок 3.2.1).

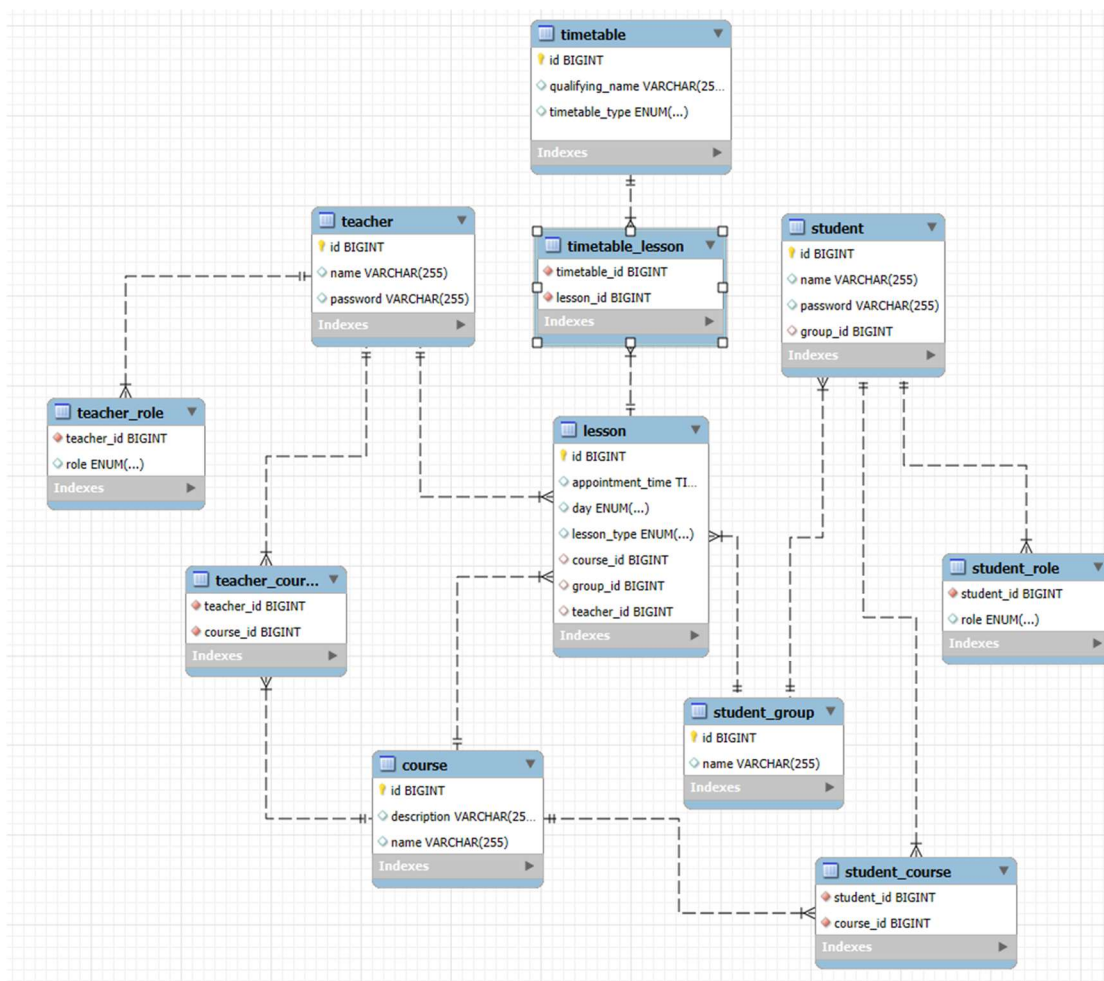


Рисунок 3.2.1 – Модель бази даних

Згідно з рисунком, який наведено вище, можна виділити основні сутності моделі:

- student;
- teacher;
- course;
- lesson;
- timetable;
- timetable_lesson;
- student_group;
- student_course;
- teacher_course;

– student_role і teacher_role.

Таблиця student містить інформацію про студентів. Вона включає такі поля, як name — ім'я користувача, password — зашифрований пароль для автентифікації, та зовнішній ключ group_id, який вказує на приналежність студента до певної академічної групи. Це дозволяє об'єднувати студентів у групи для формування розкладу та призначення занять.

Таблиця teacher призначена для зберігання даних про викладачів. Кожен запис містить ім'я викладача (name), пароль (password) та унікальний ідентифікатор (id). Ці дані використовуються для автентифікації викладача, а також для зв'язку з курсами та заняттями, які він проводить.

Таблиця course описує навчальні курси. Містить два основних поля: name — назва курсу, та description — короткий опис змісту курсу. Курс є основною одиницею навчального процесу, з яким пов'язані як студенти, так і викладачі.

Центральна таблиця lesson поєднує в собі курси, викладачів та студентські групи. Кожне заняття має фіксовані параметри: appointment_time — час проведення, day — день тижня, lesson_type — тип заняття (наприклад, лекція або практика), а також зовнішні ключі course_id, group_id та teacher_id, що дозволяє визначити, хто, з ким і коли проводить конкретне заняття.

Таблиця timetable описує розклади. Вона включає поле timetable_type — тип розкладу (наприклад, для викладача або групи) і поле qualifying_name — його умовна назва.

Таблиця timetable_lesson встановлює зв'язок між таблицями timetable та lesson. Кожен запис вказує, яке саме заняття включено до якого розкладу. Такий підхід дозволяє легко формувати різні варіанти розкладів для одного і того ж набору занять.

Таблиця student_group містить інформацію про академічні групи. Поле name дозволяє ідентифікувати групу, а первинний ключ використовується як зовнішній ключ у таблиці student та уроків. Групи є важливою одиницею при формуванні розкладу та організації навчального процесу.

Таблиця `student_course` реалізує зв'язок "багато-до-багатьох" між студентами та курсами. Кожен запис містить `student_id` та `course_id`, дозволяючи фіксувати, які студенти записані на які курси. Це дає змогу реалізувати кастомізоване навчання залежно від обраних курсів.

Таблиця `teacher_course` — аналогічна до `student_course`, яка встановлює зв'язок між викладачами та курсами. Містить поля `teacher_id` і `course_id`, що дозволяє системі знати, який викладач читає який курс, і використовувати цю інформацію при формуванні розкладу.

Таблиці `student_role` і `teacher_role` — окремі таблиці, призначені для зберігання ролей користувачів (наприклад, студент, староста, адміністратор). Вони містять поля `role` (у форматі ENUM) та зовнішній ключ (`student_id` або `teacher_id`). Такий підхід дає змогу гнучко керувати правами доступу та можливостями користувачів у межах системи.

Модель бази даних розроблено з дотриманням щонайменше третьої нормальної форми, що дозволяє уникнути аномалій оновлення, вставки і видалення. Кожна сутність має унікальний первинний ключ (`id`), а зв'язки між таблицями встановлюються за допомогою зовнішніх ключів, що забезпечує цілісність і узгодженість даних у базі. Завдяки такій структурі забезпечено гнучке управління навчальним процесом, можливість масштабування, адаптацію під різні сценарії використання та зручність реалізації валідації даних і запитів.

3.3 Діаграма класів

Нижче представлено діаграму класів, яка відображає архітектуру програмного застосунку. Оскільки повна система містить велику кількість сутностей, для збереження наочності у даній діаграмі зображено лише три ключові сутності — `Student` (Студент), `Teacher` (Викладач) та `Timetable` (Розклад). Це дозволяє продемонструвати загальні принципи побудови архітектури без перевантаження схеми (рисунок 3.3.1 та рисунок 3.3.2).

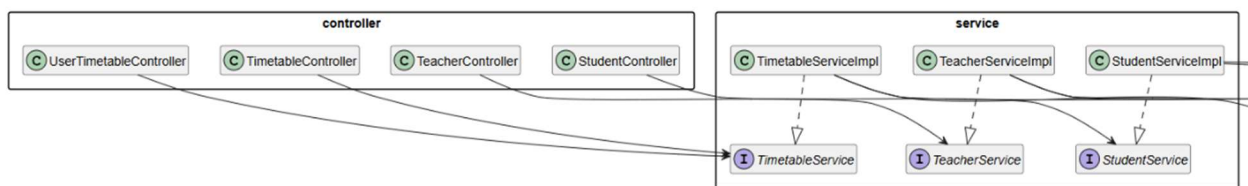


Рисунок 3.3.1 – Перша частина діаграми класів

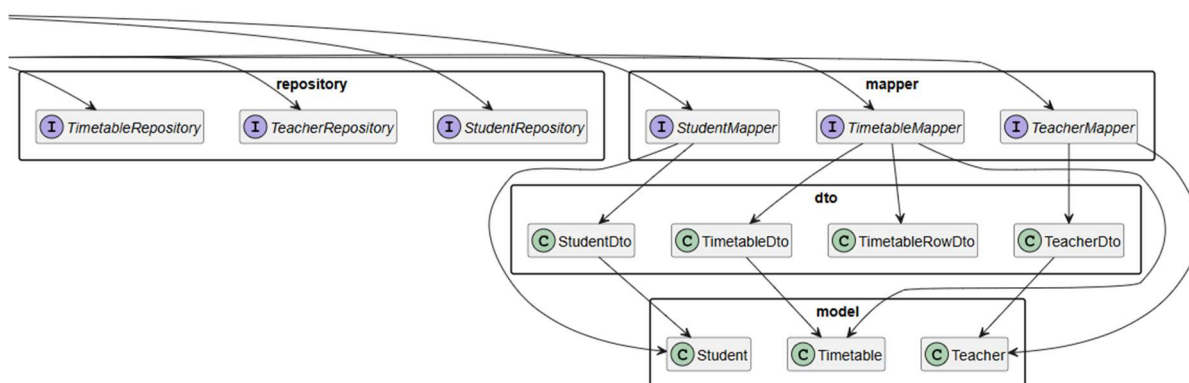


Рисунок 3.3.2 – Друга частина діаграми класів

Застосунок реалізовано відповідно до багаторівневої (layered) архітектури, яка розділяє відповідальності між різними компонентами системи. Такий підхід забезпечує гнучкість, розширюваність та спрощує тестування окремих модулів.

Основні рівні архітектури:

1. Model (Entity Layer)

На цьому рівні знаходяться сутності, які відповідають за структуру збереження даних у базі. Кожна сутність є окремим Java-класом, а її поля відображають колонки відповідної таблиці в базі даних. Наприклад:

- Student — містить поля id, name, password, group, roles, courses;
- Teacher — поля id, name, password, roles, courses;
- Timetable — поля id, timetableType, qualifyingName, lessons.

Кожна сутність має унікальний первинний ключ `id` та визначені зв'язки з іншими сутностями за допомогою анотацій ORM (наприклад, `@ManyToOne`, `@OneToMany`).

2. DTO (Data Transfer Object Layer)

DTO-класи використовуються для передачі даних між клієнтською частиною і сервером, а також між різними шарами логіки. Вони дозволяють приховати внутрішню структуру сутностей та контролювати, яка саме інформація буде передана назовні. Наприклад:

- `StudentDto` — спрощена версія `Student`, що може не містити чутливої інформації (наприклад, пароля);
- `TeacherDto`, `TimetableDto`, а також спеціалізований `TimetableRowDto` — для передачі окремих рядків розкладу.

В застосунку ці класи використовуються для передачі даних між сервісами та контролерами

3. Mapper (Mapping Layer)

Мапери [23] відповідають за конвертацію між `Entity` та DTO-об'єктами. Це забезпечує відділення представлення даних (DTO) від бізнес-логіки (`Entity`). Для кожної сутності створено окремий мапер: `StudentMapper`, `TeacherMapper`, `TimetableMapper` і так далі.

4. Repository (Data Access Layer)

DAO-рівень представлений інтерфейсами, які наслідують `JpaRepository`. Вони дозволяють виконувати операції з базою даних без явного написання SQL-запитів. Для кожної сутності створено окремий репозиторій: `StudentRepository`, `TeacherRepository`, `TimetableRepository` і так далі. Репозиторії працюють безпосередньо з `Entity`-класами, надаючи базові методи для збереження, пошуку, оновлення та видалення об'єктів.

5. Service (Business Logic Layer)

Сервіси реалізують бізнес-логіку застосунку. Для кожної сутності створено окремий сервісний інтерфейс та клас-реалізацію (наприклад `StudentService` і `StudentServiceImpl`). Інтерфейси визначають контракт — набір методів, які надає

сервіс. Реалізації виконують конкретні дії, наприклад, викликають репозиторії, обробляють DTO або працюють із маперами. Це дозволяє дотримуватись принципу інверсії залежностей (Dependency Inversion Principle) і полегшує модульне тестування.

6. Controller (Presentation Layer)

Контролери реалізують шаблон Model–View–Controller (MVC) [24] і відповідають за обробку HTTP-запитів від користувача та повернення відповідних HTML-сторінок. Кожен контролер пов'язаний із відповідною частиною функціональності застосунку. Вони приймають запити з інтерфейсу користувача, наприклад, через форми або переходи за посиланнями. Після цього викликають відповідні методи сервісного рівня для обробки даних, додають отримані дані до моделі (об'єкт Model) і в результаті повертають назву HTML-шаблону, який буде відрендерено та показано користувачу. Усі шаблони (HTML-файли) зберігаються в окремому каталозі (/templates), і можуть бути реалізовані з використанням шаблонізаторів, таких як Thymeleaf.

Наприклад:

- StudentController відповідає за перегляд, редагування або створення студентів;
- TeacherController — за роботу з викладачами;
- TimetableController — за перегляд або створення розкладу;
- UserTimetableController — за виведення персонального розкладу користувача.

Таким чином, контролери не лише спрямовують запити, але й керують тим, які дані й у якому вигляді відображаються на сторінці.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблена система управління розкладом занять є вебзастосунком, що не вимагає спеціального встановлення на стороні користувача. Для її використання достатньо мати доступ до веббраузера (наприклад, Google Chrome, Mozilla Firefox, Microsoft Edge) і підключення до мережі. Це значно спрощує розгортання й використання системи як для студентів, так і для викладачів та адміністративного персоналу.

4.1 Сценарій роботи користувача (студента або викладача)

Щойно користувач заходить на сайт, він потрапляє на головну сторінку (рисунок 4.1.1) з приємним адаптивним інтерфейсом та загальною інформацією про сайт або освітню установу.

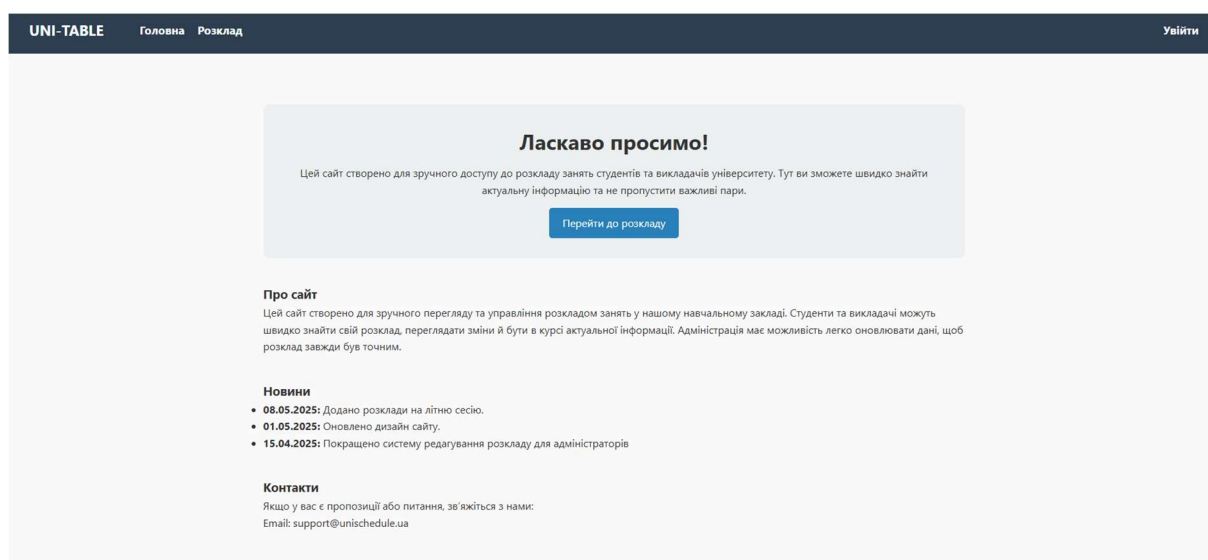
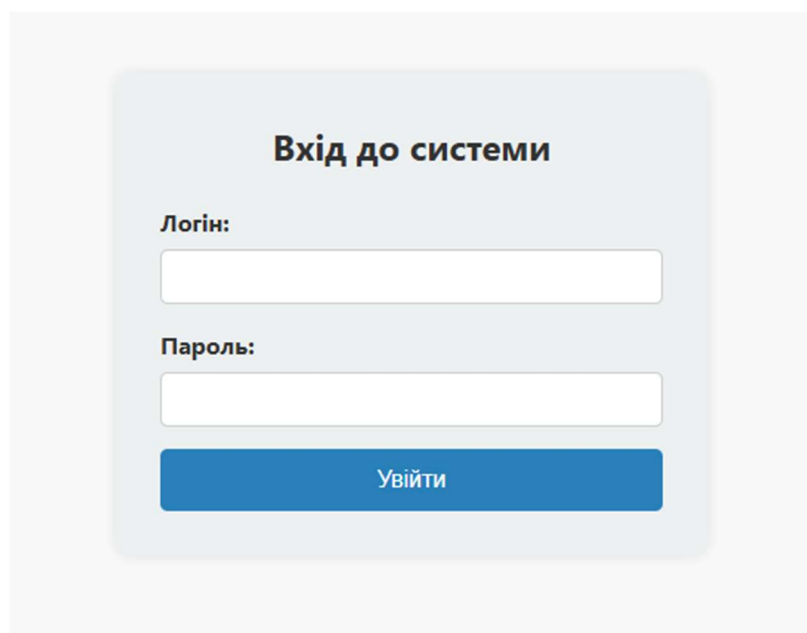


Рисунок 4.1.1 – Головна сторінка сайту

Щоб переглянути свій розклад, користувач може натиснути одну з двох кнопок: “Розклад” у верхній секції сайту або “Перейти до розкладу” на головній сторінці. Після цього він буде переадресований на сторінку входу, як зображено на рисунку 4.1.2. Важливо зауважити, що для того, щоб побачити свій розклад, треба обов’язково увійти у свій акаунт.



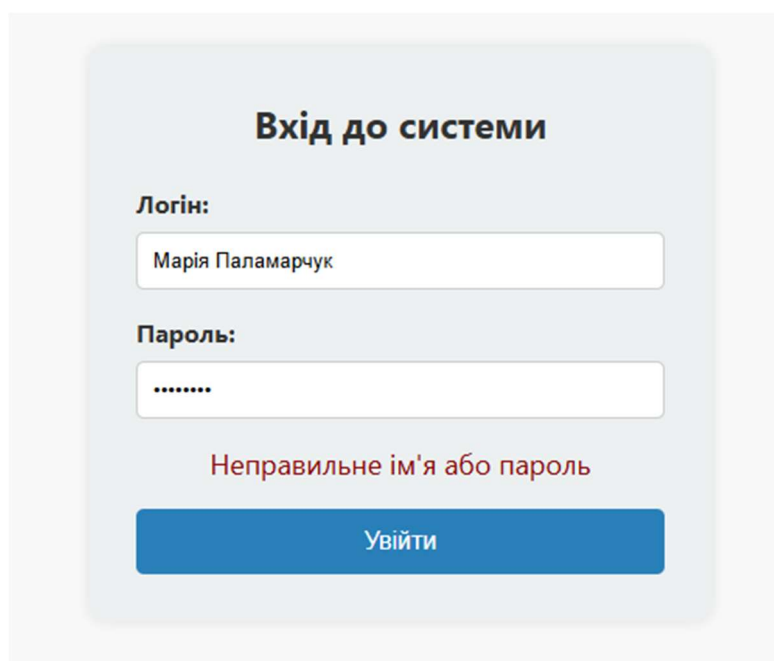
The image shows a login form with a light blue background. At the top, the title "Вхід до системи" is centered. Below it, there are two labels: "Логін:" and "Пароль:". Each label is followed by a white rectangular input field. At the bottom of the form is a solid blue button with the text "Увійти" in white.

Рисунок 4.1.2 – Сторінка входу

Після того як студент або викладач переходить на сторінку входу, йому необхідно ввести свої облікові дані. Система передбачає два поля для введення: логін та пароль. Важливо зазначити, що в ролі логіна використовується ім’я та прізвище користувача, які мають відповідати даним, наданим адміністрацією.

Слід звернути увагу, що звичайні користувачі, а саме студенти та викладачі, не мають можливості самостійно зареєструвати новий акаунт через сайт. Процес реєстрації повністю контролюється адміністраторами системи. Вони самостійно створюють облікові записи для кожного користувача, після чого надсилають логін і пароль будь-яким зручним способом — наприклад, електронною поштою, через месенджери або особисто під час навчального процесу.

У разі, якщо користувач вводить некоректні дані для входу — наприклад, помиляється в логіні або паролі — система відобразить відповідне повідомлення про помилку (рисунки 4.1.3). У повідомленні буде зазначено, що введені дані неправильні, і користувачу буде запропоновано повторити спробу входу. Поля для введення логіна та пароля залишаться активними, тож він зможе одразу внести правильну інформацію.



Вхід до системи

Логін:

Марія Паламарчук

Пароль:

.....

Неправильне ім'я або пароль

Увійти

Рисунок 4.1.3 – Приклад невдалої спроби входу

Після успішного входу в систему користувач автоматично перенаправляється на сторінку з розкладом (рисунки 4.1.4). Зміст цієї сторінки залежить від ролі користувача.

Якщо користувач є студентом, система автоматично визначає його навчальну групу та відображає розклад занять, актуальний саме для цієї групи. У розкладі можна побачити дні та години проведення пар, назви дисциплін та прізвища викладачів.

У випадку, якщо користувач є викладачем, система формує індивідуальний розклад, що містить перелік усіх занять, у яких він бере участь як викладач. Такий розклад також містить інформацію про час, предмет та групу студентів.

Розклад занять						
Для Марія Паламарчук						
	Понеділок	Вівторок	Середа	Четвер	П'ятниця	Субота
08:30		Предмет: Економіка Група: рж-79 Викладач: Марія Паламарчук				
10:25		Предмет: Економіка Група: бж-13 Викладач: Марія Паламарчук		Предмет: Економіка Група: сж-62 Викладач: Марія Паламарчук		
12:20	Предмет: Математика Група: еж-31 Викладач: Марія Паламарчук					
14:15				Предмет: Математика Група: те-79 Викладач: Марія Паламарчук		
16:10	Предмет: Математика Група: те-79 Викладач: Марія Паламарчук		Предмет: Математика Група: не-86 Викладач: Марія Паламарчук	Предмет: Математика Група: бж-13 Викладач: Марія Паламарчук		

Рисунок 4.1.4 – Сторінка з розкладом

Розклад містить усю необхідну інформацію як для студентів, так і для викладачів. Кожен користувач бачить повний список занять, відповідно до своєї ролі.

Окрему увагу варто звернути на позначення типів занять. Поруч із кожним записом у розкладі розміщено спеціальний значок, що візуально відрізняє лекції від практичних занять (рисунок 4.1.5 та рисунок 4.1.6). Це дозволяє користувачам швидко орієнтуватися в структурі розкладу й розуміти, який формат заняття їх очікує.

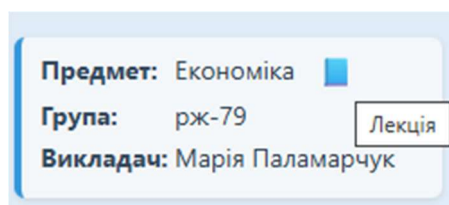


Рисунок 4.1.5 – Демонстрація позначення лекції

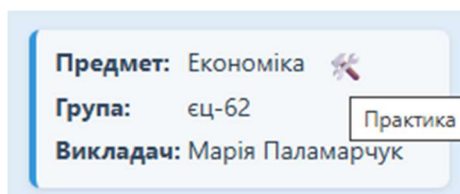


Рисунок 4.1.6 – Демонстрація позначення практики

Як зазначено в цьому сценарії, викладачі та студенти мають змогу у зручний для себе спосіб користуватися застосунком з метою отримання повної та актуальної інформації, пов'язаної з навчальним розкладом, що сприяє ефективному плануванню навчального процесу.

4.2 Сценарій роботи адміністратора

Тепер розглянемо, як працюють з системою користувачі з правами адміністратора. Основна відмінність адміністраторів від звичайних користувачів полягає в тому, що вони мають доступ до адміністративної панелі управління.

Адмін-панель — це спеціальний розділ сайту, який відкривається лише для користувачів з відповідними правами доступу. Вона автоматично з'являється у верхній секції сайту (рисунок 4.2.1) після входу в систему, якщо обліковий запис належить адміністратору.

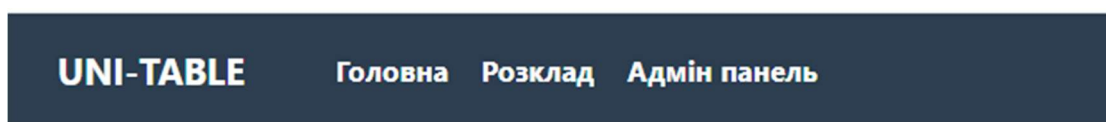


Рисунок 4.2.1 – Демонстрація доступу до адмін-панелі привілейованим користувачам

Цей інтерфейс дозволяє адміністраторам виконувати низку важливих дій — наприклад, керувати користувачами, створювати або редагувати розклади, призначати викладачів до занять, а також оновлювати інформацію про навчальні групи чи дисципліни. Адмін-панель спроектована таким чином, щоб забезпечити швидкий доступ до ключових функцій без зайвих переходів по сайту. Переглянути як виглядає адмін-панель можна на рисунку 4.2.2

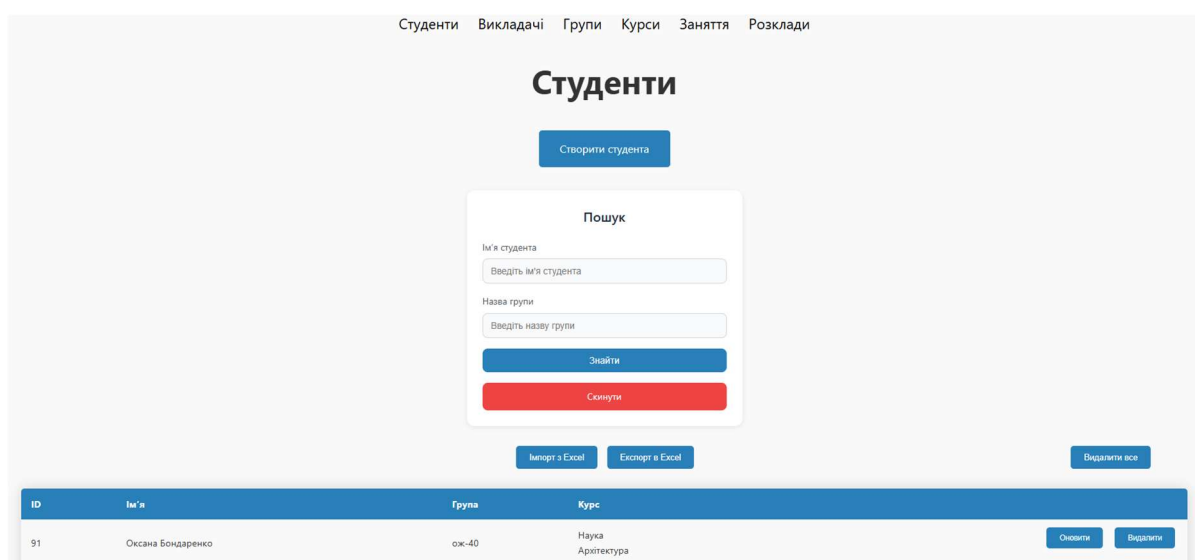


Рисунок 4.2.2 – Адмін-панель

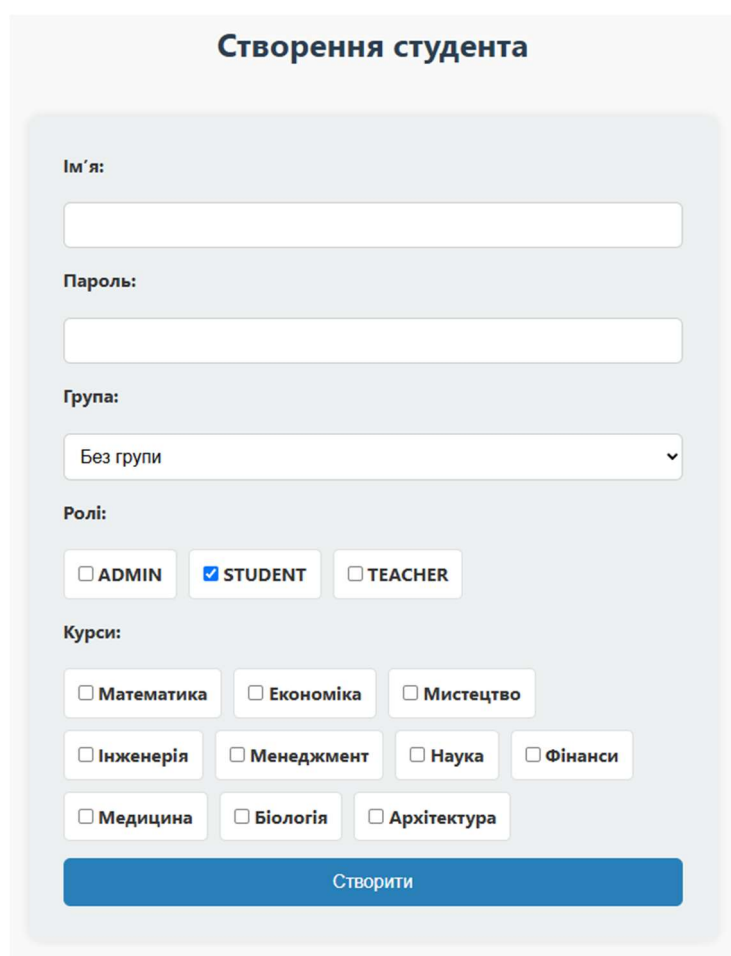
В адмін-панелі адміністратор може перемикатися між різними розділами, які відповідають ключовим елементам системи, зокрема: студентами, викладачами, навчальними групами, заняттями та розкладами.

Кожен із цих розділів представлений у вигляді таблиці, що містить усі відповідні об'єкти. Над таблицею розташовано зручну пошукову панель, яка дозволяє швидко знайти потрібну інформацію за ключовими параметрами — наприклад, ім'ям, назвою групи чи дисципліни.

Крім перегляду, адміністратору доступні базові операції з об'єктами: створення нових записів, редагування існуючих та їхнє видалення. Причому система дозволяє як вилучати окремі елементи, так і застосовувати масові дії до кількох вибраних об'єктів одночасно.

Також передбачено функціональність для експорту всіх даних до файлу у форматі Excel. Це може бути корисно для збереження резервних копій, аналізу або звітності. У свою чергу, доступна і зворотна дія — імпорт даних з Excel-файлу, що значно пришвидшує процес масового додавання інформації до системи

Якщо адміністратор бажає створити нового користувача, він має натиснути відповідну кнопку. Після цього система автоматично перенаправить його на спеціальну сторінку створення користувача (рисунок 4.2.3).



Сторінка створення студента. Формуляція заголовка: **Створення студента**. Поля для введення: **Ім'я:** (текстове поле), **Пароль:** (текстове поле). Вибір групи: **Група:** (список вибору, значення: Без групи). Вибір ролі: **Ролі:** (кнопки: ADMIN, STUDENT (вибрано), TEACHER). Вибір курсу: **Курси:** (кнопки: Математика, Економіка, Мистецтво, Інженерія, Менеджмент, Наука, Фінанси, Медицина, Біологія, Архітектура). Кнопка **Створити**.

Рисунок 4.2.3 – Сторінка створення студента

Як показано на рисунку, адміністратор має можливість задати всі необхідні дані для створення нового студента. У формі передбачені поля для введення імені, прізвища, логіна, навчальної групи та інших параметрів, необхідних для реєстрації користувача в системі.

Важливо зауважити, що форма створення нового студента включає валідацію введених даних. Зокрема, якщо адміністратор намагається створити користувача з ім'ям або логіном, які вже існують у базі даних, система не дозволить завершити операцію. У такому випадку на екрані з'явиться повідомлення про помилку (рисунок 4.2.4).

Створення студента

Це ім'я користувача вже зайняте.

Ім'я:
Оксана Бондаренко

Пароль:
.....

Група:
Без групи

Ролі:
☐ ADMIN ☒ STUDENT ☐ TEACHER

Курси:
☒ Математика ☒ Економіка ☐ Мистецтво
☒ Інженерія ☐ Менеджмент ☐ Наука ☐ Фінанси
☐ Медицина ☐ Біологія ☐ Архітектура

Створити

Рисунок 4.2.4 – Приклад невдалої спроби створити студента

Якщо адміністратор бажає оновити або видалити наявного студента, він може скористатися відповідними кнопками (рисунок 4.2.5), які розміщені у таблиці поруч з кожним записом. Натиснувши кнопку “Видалити”, адміністратор ініціює процес видалення обраного студента з бази даних. Після підтвердження дії запис буде остаточно видалено, і він зникне з таблиці. Якщо ж адміністратор натисне

кнопку “Оновити”, система автоматично перенаправить його на сторінку редагування профілю студента.



Рисунок 4.2.5 – Кнопки для видалення та оновлення сутності

Сторінка оновлення (рисунок 4.2.6) запису за своєю структурою практично не відрізняється від сторінки створення нового об'єкта. Єдина принципова відмінність полягає в тому, що всі поля форми автоматично заповнюються поточними значеннями, взятими з відповідного запису в базі даних. Це дозволяє адміністратору швидко вносити зміни, не заповнюючи форму з нуля.

Варто звернути увагу, що під час редагування студента або викладача поле для пароля залишається порожнім. Це зроблено з міркувань безпеки: паролі користувачів зберігаються в зашифрованому вигляді, і система не передає їх у відкритому вигляді на HTML-сторінку. Таким чином, уникнуто ризику витоку конфіденційної інформації.

Якщо адміністратор бажає змінити пароль користувача, він може ввести нове значення вручну. Якщо ж поле залишити порожнім, поточний пароль залишиться без змін.

Оновити студента

Ім'я
Тимофій Захарко

Пароль

Група
не-86

Ролі
☐ ADMIN
 ☒ STUDENT
 ☐ TEACHER

Курси
☐ Математика
 ☐ Економіка
 ☐ Мистецтво
 ☐ Інженерія
 ☐ Менеджмент
 ☐ Наука
 ☒ Фінанси
 ☐ Медицина
 ☐ Біологія
 ☒ Архітектура

Оновити

Рисунок 4.2.6 – Сторінка оновлення студента

Як уже зазначалося раніше, для кожної таблиці з даними в адміністративній панелі передбачена можливість експорту та імпорту інформації у форматі Excel. Під пошуковою секцією кожного розділу розміщено відповідні кнопки — “Експорт в Excel” та “Імпорт з Excel” (рисунок 4.2.7).

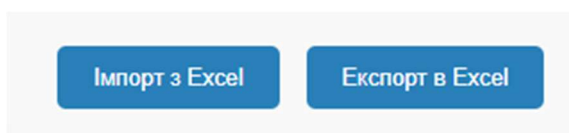


Рисунок 4.2.7 – Кнопки для імпорту та експорту

Після натискання на кнопку “Експорт в Excel” відкривається діалогове вікно, в якому користувач може обрати місце на своєму пристрої для збереження Excel-

файлу з даними. Програма автоматично формує файл (рисунок 4.2.8), який містить актуальну інформацію з відповідної таблиці, та пропонує зберегти його у вказаному каталозі.

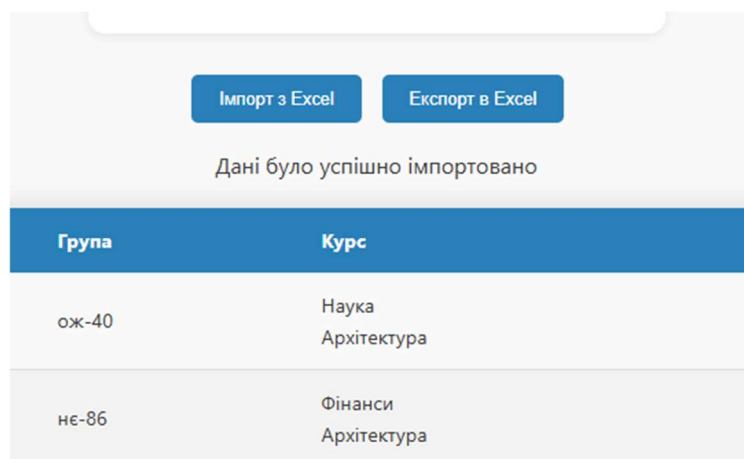
	A	B	C	D	E	F	G	H	I
1	ID	Ім'я	Пароль	Ролі	Група	Курси			
2	1	Оксана Бондаренко	\$2a\$10\$Rdg.Uli	STUDENT	ож-40	Наука, Архітектура			
3	2	Тимофій Захарко	\$2a\$10\$/RTzAM	STUDENT	не-86	Фінанси, Архітектура			
4	3	Денис Іващенко	\$2a\$10\$T.QSO	STUDENT	єц-62	Економіка, Біологія			
5	4	Егор Ткаченко	\$2a\$10\$W39FV	STUDENT	нп-02	Біологія			
6	5	Олег Паламарчук	\$2a\$10\$JTy0iup	STUDENT	нп-02	Наука, Фінанси			
7	6	Денис Гнатенко	\$2a\$10\$ENBwF	STUDENT	єц-62	Інженерія, Фінанси, Медицина			
8	7	Катерина Паламарчук	\$2a\$10\$V5815J	STUDENT	бх-13	Менеджмент, Наука			
9	8	Оксана Коваленко	\$2a\$10\$LsXJaw	STUDENT	за-29	Економіка, Мистецтво			
10	9	Максим Семененко	\$2a\$10\$0fVgqe	STUDENT	не-86	Економіка			
11	10	Катерина Романенко	\$2a\$10\$f5bcTl	STUDENT	єг-31	Інженерія, Менеджмент, Архітектура			
12	11	Єва Захарко	\$2a\$10\$gACCTR	STUDENT	рж-79	Мистецтво, Менеджмент			
13	12	Владислав Ткаченко	\$2a\$10\$dIn.l1q	STUDENT	єг-31	Інженерія, Фінанси, Медицина			
14	13	Егор Данилко	\$2a\$10\$9iLSsA	STUDENT	нп-02	Математика, Економіка, Архітектура			
15	14	Егор Кравченко	\$2a\$10\$D9mPs	STUDENT	бх-13	Фінанси			
16	15	Артем Семененко	\$2a\$10\$Ohh8ys	STUDENT	бх-13	Мистецтво			
17	16	Артем Литвиненко	\$2a\$10\$bgQQb	STUDENT	єг-31	Біологія			
18	17	Людмила Хоменко	\$2a\$10\$N7V3B	STUDENT	єц-62	Наука			
19	18	Ілля Бондаренко	\$2a\$10\$gb.OL.S	STUDENT	кт-27	Мистецтво, Наука, Фінанси			
20	19	Людмила Гнатенко	\$2a\$10\$fsTvw	STUDENT	ож-40	Інженерія, Біологія			
21	20	Тимофій Ткаченко	\$2a\$10\$R5.h.Fs	STUDENT	за-29	Інженерія			
22	21	Андрій Данилко	\$2a\$10\$xNTVp	STUDENT	єц-62	Медицина			
23	22	Андрій Ключко	\$2a\$10\$dLk.OD	STUDENT	рж-79	Економіка, Інженерія, Медицина			
24	23	Владислав Кравченко	\$2a\$10\$f1Gsum	STUDENT	єг-31	Наука, Фінанси, Архітектура			

Рисунок 4.2.8 – Приклад збереженого Excel файлу

Щоб імпортувати дані, адміністратору необхідно натиснути на кнопку “Імпорт з Excel”. Після цього відкриється діалогове вікно, в якому потрібно вибрати Excel-файл, що містить підготовлені дані для додавання до системи.

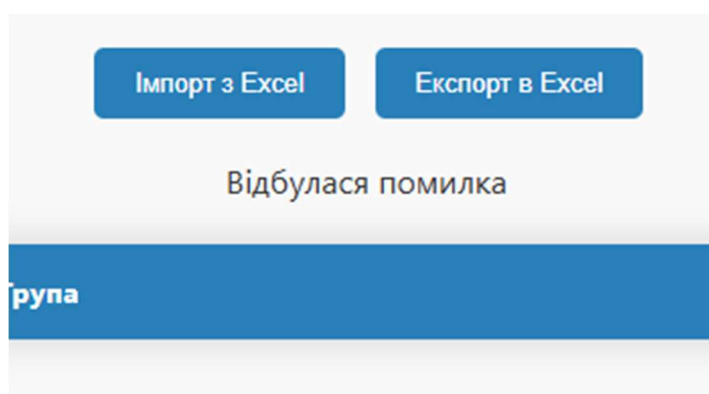
Після вибору файлу система зчитує його вміст і намагається додати або оновити записи у відповідній таблиці бази даних. Якщо імпорт проходить успішно (рисунок 4.2.9), адміністратор побачить повідомлення про успішне завершення операції, а нові або оновлені записи одразу з’являться в таблиці.

У разі виникнення помилок — наприклад, через некоректну структуру файлу, дублікати або відсутність обов’язкових полів — над таблицею з’явиться відповідне повідомлення (рисунок 4.2.10).



Група	Курс
ож-40	Наука Архітектура
нє-86	Фінанси Архітектура

Рисунок 4.2.9 – Приклад успішного імпорту



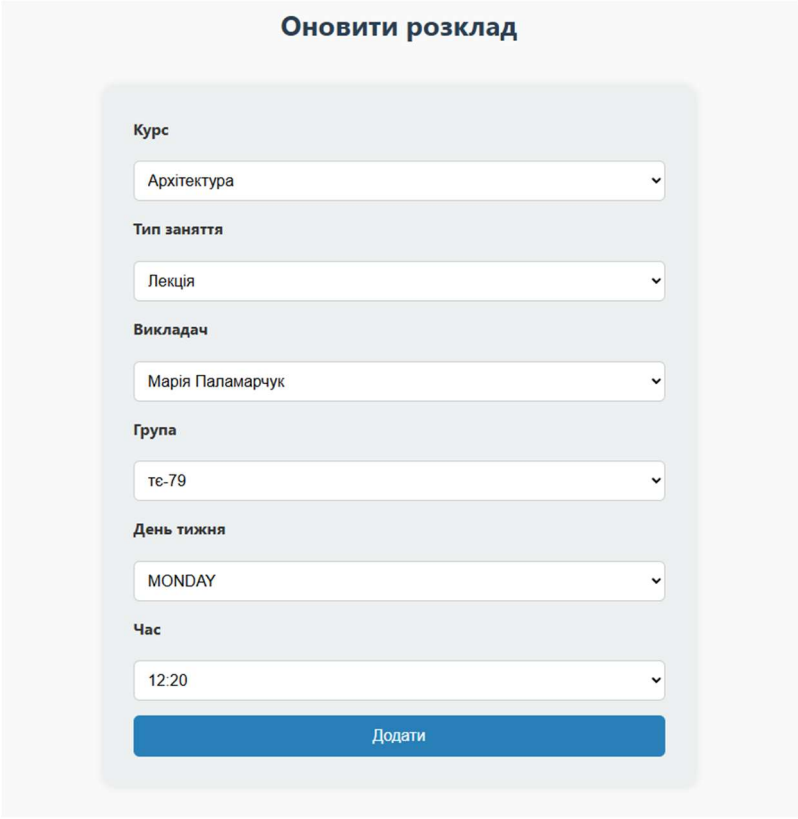
Група

Рисунок 4.2.10 – Приклад невдалого імпорту

Усі описані операції було продемонстровано на прикладі роботи з таблицею студентів, однак важливо зауважити, що аналогічний функціонал доступний і для інших таблиць у системі — таких як викладачі, навчальні групи, заняття та розклади. У кожному з цих розділів адміністратор має можливість переглядати, створювати, редагувати, видаляти записи, а також здійснювати імпорт та експорт даних за допомогою Excel. Завдяки єдиному підходу до інтерфейсу, робота з усіма сутностями є логічною, послідовною та інтуїтивно зрозумілою.

Окремо варто продемонструвати, як адміністратор може редагувати розклад для конкретного викладача або навчальної групи. Сторінка оновлення розкладу має унікальну структуру й складається з двох основних частин.

У першій частині сторінки (рисунок 4.2.11) адміністратор має змогу додати нове заняття до розкладу. Для цього потрібно заповнити всі необхідні параметри пари — зокрема день тижня, час початку та завершення, назву предмета, тип заняття (лекція або практика), а також вибрати викладача й навчальну групу. Після заповнення форми й підтвердження дії заняття буде додане до розкладу.



Оновити розклад

Курс
Архітектура ▼

Тип заняття
Лекція ▼

Викладач
Марія Папамарчук ▼

Група
те-79 ▼

День тижня
MONDAY ▼

Час
12:20 ▼

Додати

Рисунок 4.2.11 – Перша частина сторінки оновлення розкладу

Друга частина сторінки (рисунок 4.2.12) відображає поточний розклад — усі заняття, які вже закріплені за цією групою або викладачем. Для зручності перегляду заняття відсортовано за днями тижня — від початку понеділка до завершення суботи. Кожен запис містить основну інформацію про пару, а також кнопку “Видалити“, за допомогою якої адміністратор може швидко прибрати конкретне заняття з розкладу.

Система також передбачає базову перевірку на конфлікти в часі. Якщо адміністратор намагається додати заняття на той самий день і час, коли вже

заплановано інше, система не дозволить виконати цю дію й виведе відповідне повідомлення про помилку. Це дозволяє уникнути дублювання та забезпечує логічну послідовність у формуванні розкладу.

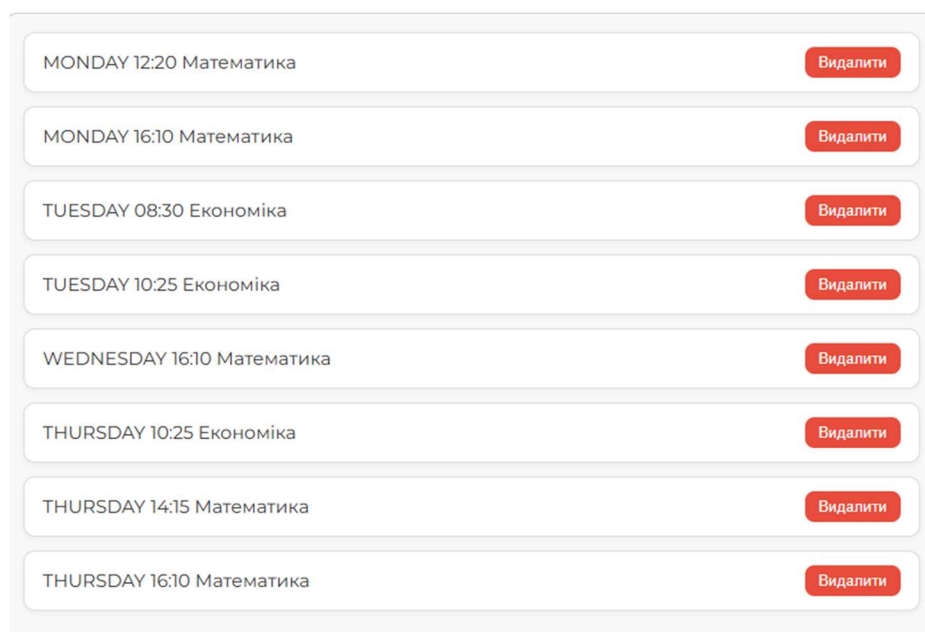


Рисунок 4.2.12 – Друга частина сторінки оновлення розкладу

Таким чином було продемонстровано основні сценарії взаємодії з застосунком для всіх категорій користувачів: студентів, викладачів та адміністрації. Представлені вище інструкції покликані полегшити ознайомлення з функціоналом системи та можуть стати корисним орієнтиром для нових користувачів під час початкового налаштування та повсякденного використання застосунку.

ВИСНОВКИ

У ході виконання завдання дипломної роботи було реалізовано повноцінний веб-застосунок для автоматизованого управління навчальним розкладом, який дозволяє ефективно організовувати інформацію про заняття, студентів, викладачів та курси. Система орієнтована на спрощення процесу складання та перегляду розкладу як для адміністрації навчального закладу, так і для кінцевих користувачів — студентів і викладачів.

Під час реалізації було використано сучасний технологічний стек, включаючи Java, Spring Framework, Hibernate, PostgreSQL, а також відповідні технології для фронтенду (Thymeleaf, HTML, CSS, JavaScript). Було дотримано принципів багаторівневої архітектури, що забезпечує гнучкість, масштабованість та легкість підтримки проєкту.

Створена система має інтуїтивно зрозумілий інтерфейс, який дозволяє зручно переглядати розклад, здійснювати фільтрацію за різними параметрами, а також забезпечує чіткий поділ прав доступу для різних типів користувачів.

Розроблений застосунок може бути розгорнутий у навчальних закладах різного рівня та адаптований до специфічних потреб конкретної установи. Також існують перспективи подальшого розвитку системи, зокрема: інтеграція з мобільними платформами, впровадження push-сповіщень, розширена аналітика, електронні журнали тощо.

Таким чином, поставлену мету було досягнуто — створено стабільну, функціональну, адаптивну систему для вирішення актуальної задачі організації розкладу навчального процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Єльнікова О. В. Фактори складання навчального розкладу. URL: http://umo.edu.ua/images/content/nashi_vydanya/metod_upr_osvit/v_2/10.pdf (дата звернення: 29.05.2025).
2. Переваги мови програмування Java. URL: <https://qagroup.com.ua/publications/benefits-of-java/> (дата звернення: 29.05.2025).
3. 8 найпопулярніших мов програмування у 2024 році. URL: <https://www.devjobsscanner.com/blog/top-8-most-demanded-programming-languages/> (дата звернення: 29.05.2025).
4. Відмінності мов програмування: їх табличне порівняння. URL: <https://foxminded.ua/vidminnosti-mov-prohramuvannia/> (дата звернення: 29.05.2025).
5. Сайти на Java: плюси та мінуси. URL: <https://byshep.com/statti/sajti-na-java-plyusi-ta-minusi/> (дата звернення: 29.05.2025).
6. Фреймворк Spring та його особливості. URL: <https://highload.tech/uk/frejmwork-spring-ta-jogo-osoblivosti/> (дата звернення: 29.05.2025).
7. Розуміння Інверсії Контролю (IoC) у фреймворку Spring. URL: <https://javascript.org.ua/rozuminnya-inversi%D1%97-kontrolyu-ioc-u-frejmworku-spring/> (дата звернення: 29.05.2025).
8. Що таке PostgreSQL і для чого використовується. URL: <https://foxminded.ua/postgresql-shcho-tse/> (дата звернення: 29.05.2025).
9. Переваги та Недоліки PostgreSQL. URL: <https://prezi.com/p/k9j0zsyztgc/postgresql/> (дата звернення: 29.05.2025).
10. Фреймворки ORM та патерни, що в них використовуються. URL: https://elartu.tntu.edu.ua/bitstream/lib/45024/2/VII_MCNTK_2024_Lanevych_T-ORM_frameworks_and_patterns_87-88.pdf (дата звернення: 29.05.2025).
11. Що таке Git, GitHub та як працювати з системою. URL: <https://dzudzylo.com/tehnologii/shho-take-git-github-ta-yak-pratsyuvaty-z-systemoyu.html> (дата звернення: 29.05.2025).

12. У чому полягає Maven Java. URL: <https://www.guru99.com/uk/maven-tutorial.html> (дата звернення: 29.05.2025).
13. Нормалізація структури даних. URL: <https://studfile.net/preview/5391744/> (дата звернення: 29.05.2025).
14. Оптимізація швидкості завантаження сайту за допомогою Lazy loading. URL: <https://webmaestro.com.ua/ua/blog/lazy-loading/> (дата звернення: 29.05.2025).
15. Використання транзакцій в реляційних базах даних. URL: https://journal.gen.tech/post/tranzakciy_v-relyatsiynykh-bazakh-danykh (дата звернення: 29.05.2025).
16. SQL ін'єкції та захист від них. URL: [https://foxminded.ua/sql-iniektsii/](https://foxminded.ua/sql-iniektzii/) (дата звернення: 29.05.2025).
17. Клієнт-серверна архітектура. URL: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/> (дата звернення: 29.05.2025).
18. The pros and cons of a layered architecture pattern. URL: <https://www.techtarget.com/searchapparchitecture/tip/The-pros-and-cons-of-a-layered-architecture-pattern> (дата звернення: 29.05.2025).
19. MVC: шаблон проектування архітектури додатку. URL: <https://brander.ua/technologies/mvc> (дата звернення: 29.05.2025).
20. Data Transfer Object. URL: <https://martinfowler.com/eaCatalog/dataTransferObject.html> (дата звернення: 29.05.2025).
21. Singleton Method Design Pattern. URL: <https://www.geeksforgeeks.org/singleton-design-pattern/> (дата звернення: 29.05.2025).
22. Factory / Builder pattern. URL: <https://refactoring.guru/design-patterns/factory-method> (дата звернення: 29.05.2025).

23. Mapping Data Objects Between Layers. URL: <https://medium.com/@wambuinjumbi/mapping-data-objects-between-layers-e16745f67cb2> (дата звернення: 29.05.2025).

24. Model-View-Controller Pattern in Java. URL: <https://java-design-patterns.com/patterns/model-view-controller/> (дата звернення: 29.05.2025).

ДОДАТОК А

Програмні засоби:

- мова програмування – Java (версія 17);
- фреймворк для розробки серверної частини – Spring Boot;
- шаблонізація інтерфейсу користувача – Thymeleaf;
- об’єктно-реляційне відображення – Spring Data JPA;
- база даних – PostgreSQL;
- інструменти безпеки – Spring Security, BCrypt;
- модульне тестування – JUnit, Mockito.

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig {
```

```
    @Bean
```

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
```

```
        http.authorizeHttpRequests(requests -> requests
```

```
            .requestMatchers("/admin-panel/**").hasAuthority("ADMIN")
```

```
            .requestMatchers("/timetable").hasAnyAuthority("STUDENT", "TEACHER",
```

```
            "ADMIN")
```

```
            .requestMatchers("/**").permitAll()
```

```
            .anyRequest().authenticated()
```

```
        )
```

```
        .formLogin(form -> form
```

```

        .loginPage("/login")
        .defaultSuccessUrl("/timetable", true)
        .failureUrl("/login?error=true")
        .permitAll()
    )
    .logout(logout -> logout
        .logoutSuccessUrl("/")
        .permitAll());

    return http.build();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}

public final class ErrorMessages {
    private ErrorMessages() {}

    public static final String ENTITY_WAS_NOT_FOUND_BY_ID = "%s with id %d wasn't found";
    public static final String ENTITY_WAS_NOT_FOUND_BY_NAME = "%s with name %s wasn't found";
    public static final String ENTITY_CAN_NOT_BE_ADDED = "%s can't be added because it already exists";
    public static final String ENTITY_CAN_NOT_BE_UPDATED = "%s can't be updated because it doesn't exists";
    public static final String STUDENT_AND_TEACHER_WITH_SAME_NAME_FOUND = "Student and teacher with the same name were found";
}

```

```

import com.foxminded.dto.CourseDto;
import com.foxminded.service.CourseService;
import com.foxminded.util.ExcelDataUtil;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

```

```

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Comparator;
import java.util.List;

```

```
@Controller
```

```
@RequiredArgsConstructor
```

```
@RequestMapping("/admin-panel/courses")
```

```
public class CourseController {
```

```
    private final CourseService courseService;
```

```
    private final ExcelDataUtil excelDataUtil;
```

```
@GetMapping
```

```
    public String showAll(Model model, @RequestParam(value = "course-name", required = false)
```

```
String courseName) {
```

```
        List<CourseDto> allCourses = courseService.getAllCourses();
```

```
        allCourses = allCourses.stream().sorted(Comparator.comparing(CourseDto::id)).toList();
```

```
        if (courseName != null) {
```

```
            allCourses = allCourses.stream()
```

```
                .filter(course -> course.name().startsWith(courseName))
```

```
                .toList();
```

```
        }
```

```
        model.addAttribute("allCourses", allCourses);
```

```

    return "entityPages/coursePage";
}

```

```

@GetMapping("/course-creation")
public String showCreatePage() {
    return "createPages/createCoursePage";
}

```

```

@GetMapping("/course-update/{courseId}")
public String showUpdatePage(@PathVariable("courseId") Long courseId, Model model) {
    CourseDto course = courseService.getCourseById(courseId);
    model.addAttribute("course", course);
    return "updatePages/updateCoursePage";
}

```

```

@PostMapping("/search")
public String search(@RequestParam("courseName") String courseName) throws
UnsupportedEncodingException {
    String encodedCourseName = URLEncoder.encode(courseName, "UTF-8");
    return "redirect:/admin-panel/courses?course-name=" + encodedCourseName;
}

```

```

@PostMapping("/course-creation")
public String createCourse(@RequestParam("name") String name,
                           @RequestParam("description") String description) {
    CourseDto newCourse = new CourseDto(0L, name, description);

    if (checkIfCourseExists(newCourse)) {
        return "redirect:/admin-panel/courses/course-creation?error=true";
    }
    courseService.addCourse(newCourse);

    return "redirect:/admin-panel/courses";
}

```

```

@PostMapping("/course-update")
public String updateCourse(@RequestParam("courseId") Long courseId,
                           @RequestParam("name") String name,
                           @RequestParam("description") String description) {
    CourseDto updatedCourse = new CourseDto(courseId, name, description);

    if (checkIfCourseExists(updatedCourse)) {
        return "redirect:/admin-panel/courses/course-update/" + courseId + "?error=true";
    }

    courseService.updateCourse(updatedCourse);
    return "redirect:/admin-panel/courses";
}

@PostMapping("/course-deletion/{courseId}")
public String deleteCourse(@PathVariable("courseId") Long courseId) {
    courseService.deleteCourseById(courseId);
    return "redirect:/admin-panel/courses";
}

@PostMapping("/course-deletion")
public String deleteAllCourses() {
    courseService.deleteAllCourses();
    return "redirect:/admin-panel/courses";
}

@PostMapping("/import-excel")
public String importCourses(@RequestParam("file") MultipartFile file) {
    try {
        excelDataUtil.importCourses(file);
        return "redirect:/admin-panel/courses?importSuccess=true";
    } catch (Exception e) {
        return "redirect:/admin-panel/courses?importError=true";
    }
}

```

```

@GetMapping("export-excel")
@ResponseBody
public void exportCourses(HttpServletResponse response) {
    excelDataUtil.exportCourses(response);
}

private boolean checkIfCourseExists(CourseDto newCourse) {
    List<CourseDto> allCourses = courseService.getAllCourses();
    for (var course : allCourses) {
        if (course.name().equals(newCourse.name()) &&
            course.description().equals(newCourse.description())) {
            return true;
        }
    }

    return false;
}

}

public record CourseDto(
    Long id,
    String name,
    String description
) {}

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.List;

```

```

@Entity
@Table(name = "course")
@Getter
@Setter
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

    @Column(name = "description")
    private String description;

    @OneToMany(mappedBy = "course", cascade = CascadeType.REMOVE)
    private List<Lesson> lessons;

    @ManyToMany(mappedBy = "courses")
    private List<Student> students;

    @ManyToMany(mappedBy = "courses")
    private List<Teacher> teachers;

    @PreRemove
    private void preRemove() {
        for (var student : students) {
            student.getCourses().remove(this);
        }

        for (var teacher : teachers) {
            teacher.getCourses().remove(this);
        }
    }
}

```

```

import com.foxminded.entity.*;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.Collections;
import java.util.UUID;

```

```
@Component
```

```
public class BasicDataGenerator {
```

```

    private final List<String> fullNames;
    private final List<String> courseNames;
    private final Random random;

```

```

    public BasicDataGenerator() {
        random = new Random();
        fullNames = new ArrayList<>();
        initStudentNames();
        courseNames = new ArrayList<>();
        initCourseNames();
        Collections.shuffle(courseNames);
    }

```

```

    public List<Group> generateBasicGroups(int amount) {
        List<Group> groups = new ArrayList<>();
        for (int i = 0; i < amount; i++) {
            Group group = new Group();
            group.setId(i + 1L);
            group.setName(generateGroupName());
            groups.add(group);
        }
    }

```

```

        return groups;
    }

    public List<Course> generateBasicCourses(int amount) {
        List<Course> courses = new ArrayList<>();
        for (int i = 0; i < amount; i++) {
            Course course = new Course();
            course.setId(i + 1L);
            course.setName(getCourseName());
            course.setDescription(UUID.randomUUID().toString());
            courses.add(course);
        }

        return courses;
    }

    public List<Student> generateBasicStudents(int amount) {
        List<Student> students = new ArrayList<>();
        for (int i = 0; i < amount; i++) {
            Student student = new Student();
            student.setId(i + 1L);
            student.setName(getFullName());
            students.add(student);
        }

        return students;
    }

    public List<Teacher> generateBasicTeachers(int amount) {
        List<Teacher> teachers = new ArrayList<>();
        for (int i = 0; i < amount; i++) {
            Teacher teacher = new Teacher();
            teacher.setId(i + 1L);
            teacher.setName(getFullName());
            teachers.add(teacher);
        }
    }

```

```

    }

    return teachers;
}

public List<Lesson> generateBasicLessons(int amount) {
    List<Lesson> lessons = new ArrayList<>();
    for (int i = 0; i < amount; i++) {
        Lesson lesson = new Lesson();
        lesson.setId(i + 1L);
        lessons.add(lesson);
    }

    return lessons;
}

public List<Timetable> generateBasicTimeTables(int amount) {
    List<Timetable> timetables = new ArrayList<>();
    for (int i = 0; i < amount; i++) {
        Timetable timetable = new Timetable();
        timetable.setId(i + 1L);
        timetables.add(timetable);
    }

    return timetables;
}

private String generateGroupName() {
    char[] ukrainianLetters = {'a', 'б', 'в', 'г', 'д', 'е', 'є', 'ж', 'з', 'и',
        'і', 'к', 'л', 'м', 'н', 'о', 'п', 'р', 'с', 'т',
        'у', 'ф', 'х', 'ц', 'ч', 'ш', 'ю', 'я'};

    char firstLetter = ukrainianLetters[random.nextInt(ukrainianLetters.length)];
    char secondLetter = ukrainianLetters[random.nextInt(ukrainianLetters.length)];
}

```

```

int digit1 = random.nextInt(10);
int digit2 = random.nextInt(10);

return "" + firstLetter + secondLetter + "-" + digit1 + digit2;
}

private String getFullName() {
    String fullName = fullNames.get(random.nextInt(fullNames.size()));
    fullNames.remove(fullName);
    return fullName;
}

private String getCourseName() {
    String courseName = courseNames.get(random.nextInt(courseNames.size()));
    courseNames.remove(courseName);
    return courseName;
}

private void initCourseNames() {
    String[] courseNamesArr = new String[] {"Математика", "Біологія", "Фінанси", "Мистецтво",
        "Архітектура", "Інженерія", "Наука", "Менеджмент", "Економіка", "Медицина"};
    courseNames.addAll(Arrays.asList(courseNamesArr));
}

private void initStudentNames() {
    String[] firstNamesArr = new String[] {"Олег", "Максим", "Артем", "Ілля", "Андрій",
        "Владислав", "Роман", "Денис", "Тимофій", "Єгор",
        "Оксана", "Анастасія", "Катерина", "Аліна", "Софія",
        "Марія", "Дарина", "Мирослава", "Єва", "Людмила"};

    String[] lastNamesArr = new String[] {"Клюйко", "Шевченко", "Мельник", "Бондаренко",
"Гнатенко",
        "Савченко", "Данилко", "Литвиненко", "Паламарчук", "Ткаченко",
        "Коваленко", "Сидоренко", "Кравченко", "Мазур", "Іващенко",
        "Олійник", "Семененко", "Хоменко", "Романенко", "Захарко"};

```

```

    for (String firstName : firstNamesArr) {
        for (String lastName : lastNamesArr) {
            fullNames.add(firstName + " " + lastName);
        }
    }
}

```

```

@Mapper(componentModel = "spring")
public interface CourseMapper {

    CourseDto mapToCourseDto(Course course);

    Course mapToCourse(CourseDto courseDto);
}

```

```

@Repository
public interface CourseRepository extends JpaRepository<Course, Long> {

    Optional<Course> findByName(String name);
}

```