**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ЗАСТОСУНКУ ЕЛЕКТРОННОГО ЩОДЕННИКА ЗАСОБАМИ REACT ТА NODE.JS»**

Виконав: студент 2 курсу, групи 8.1214-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
І.В. Козловський
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент в.о. зав. кафедри комп'ютерних наук, доцент, к.пед.н. Пшенична О.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Козловському Івану Вікторовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка застосунку електронного щоденника засобами React та Node.js

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 04.12.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Основні теоретичні відомості.
2. Проєктування системи.
3. Реалізація програмного застосунку електронного щоденника.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 07.03.2025 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 14.03.2025 | |
| | | | |
| 2. | Збір вихідних даних. | 07.04.2025 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 26.05.2025 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого розділу. | 30.06.2025 | |
| | | | |
| 5. | Розробка другого розділу. | 25.08.2025 | |
| | | | |
| 6. | Розробка третього розділу. | 24.11.2025 | |
| | | | |
| 7. | Оформлення та нормоконтроль | 01.12.2025 | |
| | кваліфікаційної роботи магістра. | | |
| | | | |
| 8. | Захист кваліфікаційної роботи. | 18.12.2025 | |

Студент _____ І.В. Козловський
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

## Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка застосунку електронного щоденника засобами React та Node.js»: 51 с., 17 рис., 17 джерел.

ВЕБЗАСТОСУНОК, ЕЛЕКТРОННИЙ ЩОДЕННИК, НАВЧАННЯ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СЕРВІС, REACT.

Об’єктом дослідження є процес розробки застосунку електронного щоденника.

Мета дослідження – створити функціональний вебзастосунок з функцією перегляду, виставлення оцінок та створенням класів, використовуючи сучасні інструменти веброзробки, з метою цифровізації освітнього процесу.

Методи дослідження – методи проєктування програмного забезпечення та інтерфейсу, тестування CRUD-функціональності.

В даній роботі розроблено програмний застосунок для ведення електронних щоденників учнів та їх упорядкування згідно класів засобами React та Node.js. Основний напрямок розробки спрямований на зручність використання системи, розробки та можливість подальшого розширення.

В роботі пропонується новий підхід до створення електронного щоденника з акцентом на сучасний фронтенд-стек, адаптивний дизайн і централізоване управління станом додатку. Практична значимість роботи полягає у створенні зручного та гнучкого інструменту для організації навчального процесу, який може бути адаптований під потреби конкретного навчального закладу, допрацьований або масштабований відповідно до зростаючих вимог.

Завдяки використанню сучасних технологій, таких як React для створення інтерфейсу, Node.js для серверної логіки та Tailwind CSS для стилізації, було реалізовано ефективне, адаптивне та масштабоване рішення, що відповідає вимогам сучасного програмного забезпечення.

# SUMMARY

Master's qualifying paper "Development of a Digital Diary Application Using React and Node.js": 51 pages, 17 figures, 17 references.

DEVELOPMENT, DIARY, E-LEARNING, REACT, SERVICE, SOFTWARE, WEB APPLICATION.

The object of the study is the process of developing an electronic diary application.

The aim of the study is to create a functional web application that allows teachers to view, grade, and create classes using modern web development tools, with the aim of digitizing the educational process.

The methods of research are software and interface design methods, CRUD functionality testing.

In this work, a software application for maintaining students' electronic diaries and organizing them according to classes using React and Node.js was developed. The main direction of development was aimed at the ease of use of the system, ease of development and further expansion.

The work proposes a new approach to creating an electronic diary with an emphasis on a modern front-end stack, adaptive design and centralized application state management. The practical significance of the work lies in creating a convenient and flexible tool for organizing the educational process, which can be adapted to the needs of a particular educational institution, improved or scaled in accordance with growing requirements.

Thanks to the use of modern technologies, such as React for creating an interface, Node.js for server logic and Tailwind CSS for styling, an effective, adaptive and scalable solution was implemented that meets the requirements of modern software.

**ЗМІСТ**

Завдання на кваліфікаційну роботу……..…..2
Реферат ..... 4
Summary ..... 5
Вступ..... 8
1 Аналіз предметної області та існуючих рішень..... 10
1.1 Призначення, функціональні вимоги до освітніх систем ..... 10
1.2 Аналіз сучасних рішень..... 11
1.3 Огляд архітектури SPA та SSR..... 13
1.3.1 “Single Page Application” ..... 13
1.3.2 Пошукова оптимізація..... 15
1.4 Вибір технологій для реалізації..... 16
1.5 Збереження даних ..... 17
1.6 Адаптивний дизайн та Tailwind CSS..... 18
2 Проєктування застосунку електронного щоденника..... 20
2.1 Проєктування бази даних..... 20
2.2 Структура. MVC..... 22
2.3 Патерн проєктування Dependency Injection..... 24
2.4 Авторизація. JWT..... 25
2.5 Моделювання засобами UML..... 26
2.5.1 Діаграма варіантів використання ..... 27
2.5.2 Діаграма класів..... 29
2.5.3 Діаграма послідовностей..... 33
3 Реалізація програмного забезпечення..... 35
3.1 Налаштування середовища розробки..... 35
3.2 Реалізація бекенду (API) ..... 36
3.2.1 Робота з базою даних..... 36
3.2.2 Створення бази даних у pgAdmin 4 ..... 37

3.2.3 Контролери та маршрути ..... 38
3.2.4 Реалізація автентифікації ..... 40
3.3 Реалізація клієнтської частини ..... 42
3.3.1 Redux-сховище та структура стану ..... 42
3.3.2 Функція-фабрика для ред'юсерів ..... 43
3.3.3 Базовий клас для саг ..... 44
3.4 Головна сторінка ..... 46
3.5 Сторінка щоденника ..... 47
3.6 Сторінка «Контакти» ..... 49
Висновки ..... 50
Перелік посилань ..... 51

# ВСТУП

У сучасному освітньому середовищі цифровізація навчального процесу є невід'ємною складовою його ефективності, прозорості та доступності. Одним із важливих інструментів цифровізації є електронний щоденник – система, яка дозволяє зберігати, аналізувати та надавати учасникам освітнього процесу доступ до інформації про успішність, розклад занять та повідомлення.

Мета цього дослідження – створити функціональний вебзастосунок, що дозволяє викладачам переглядати, виставляти оцінки та створювати класи, використовуючи сучасні інструменти веброзробки, з метою цифровізації освітнього процесу.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

– проаналізувати існуючі рішення;
– дослідити сучасні підходи до розробки вебзастосунків;
– спроєктувати систему для ефективної розробки та взаємодії з користувачем;
– розробити спроєктовану систему використовуючи сучасні інструменти.

Об'єкт дослідження – процес розробки застосунку електронного щоденника.

Предмет дослідження – React, Node.js, Tailwind CSS.

Методи дослідження – методи проєктування програмного забезпечення та інтерфейсу, тестування CRUD-функціональності.

Для досягнення результату було обрано фреймворк Next.js, який поєднує у собі сучасні рішення у розробці вебзастосунків, такі як React – фреймворк для створення швидких односторінкових застосунків, та Node.js – фреймворк для створення серверної частини застосунку. У результаті отримано зручний та сучасний вебзастосунок, який дозволяє створювати об'єднання учнів та вчителів у класи та оперувати оцінками, завданнями та повідомленнями.

Отримана структура дозволяє у майбутньому легко розширювати систему новими можливостями та інтегрувати стороннє програмне забезпечення.

Основні положення кваліфікаційної роботи доповідались на Шістнадцятій Всеукраїнській, Двадцять третій регіональній науковій конференції молодих дослідників «Актуальні проблеми математики та інформатики» (Запоріжжя, 24-25 квітня 2025 р.), за результатами яких опубліковано тези.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, та переліку посилань. Загальний обсяг роботи становить 51 сторінку. У переліку посилань 17 використаних джерел.

Перший розділ присвячений аналізу існуючих рішень та сучасних підходів розробки вебзастосунків. У другому розділі наведено процес проєктування системи засобами UML діаграм. В третьому розділі описано реалізацію програмного застосунку згідно спроєктованої архітектури.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

## 1.1 Призначення, функціональні вимоги до освітніх систем

У сучасному світі цифровізуються все більше і більше сфер життя, зокрема освіта. У освітньому середовищі виникає потреба у сучасному та ефективному рішенні для організації навчального процесу та взаємодії між здобувачами освіти та викладачами .

У якості такого рішення пропонується вебзастосунок «Електронний щоденник», як вебальтернатива водночас традиційним паперовим щоденнику та журналу. Це рішення надає єдиний доступ до інформації про успішність кожного конкретного учня та класу в цілому.

Основне призначення електронного щоденника полягає у спрощенні процесу оцінювання, контролю успішності, ведення розкладу та управління домашніми завданнями. Такий застосунок забезпечує:

- доступ учнів до власних оцінок, розкладу, завдань і повідомлень від викладачів;
- можливість для викладачів виставляти оцінки, задавати домашні завдання та формувати навчальні плани;
- комунікацію між школою, учнями та їхніми батьками;
- автоматизацію процесів та зменшення навантаження на педагогів.

Використання електронних щоденників у закладах освіти підвищить рівень прозорості та оперативності освітнього процесу. Інтеграція з іншими системами, такими як платформи дистанційного навчання та системи керування закладом, дозволить будувати єдину цифрову екосистему.

Таким чином, електронний щоденник є важливим інструментом модернізації освітнього процесу.

## 1.2 Аналіз сучасних рішень

Одним із найвідоміших українських рішень у цій сфері вважають електронний щоденник «МійКлас». У ньому можна вести облік оцінок, додавати чи редагувати домашні завдання, а також стежити за відвідуванням учнів. Платформа підтримує навчальні плани, дає можливість формувати окремі класи або групи й має базову інтеграцію з кількома освітніми сервісами. Загалом система виконує свої функції й працює досить надійно, але інтерфейс вже давно не оновлювався, виглядає застарілим і погано підлаштовується під мобільні пристрої. Через це сервіс сьогодні важко назвати сучасним або зручним для широкого використання.

Ще одним поширеним рішенням є «Нові знання» [1], яке активно застосовують у школах різних регіонів. Платформа пропонує електронний журнал та щоденник, містить механізми сповіщень для батьків, дозволяє переглядати розклад і зміни в ньому, а також підтримує взаємодію з державною освітньою інфраструктурою. Хоча сервіс досить функціональний, у періоди великого навантаження інколи виникають затримки або уповільнення роботи, що користувачі помічають під час пікових годин.

Серед закордонних продуктів варто згадати Google Classroom. Це не зовсім класичний електронний щоденник, проте він дає змогу створювати та приймати завдання, оцінювати роботи, організовувати дистанційні заняття та налагоджувати спілкування між учнями і вчителем. Платформа зручна для педагогів, добре працює разом з іншими сервісами Google – зокрема Drive чи Meet – і побудована на хмарній інфраструктурі. Водночас Classroom не передбачає повноцінного ведення класного журналу в традиційному для українських шкіл форматі та лише частково підходить під національні стандарти освіти.

Іншим міжнародним рішенням є  Moodle – відкрита платформа для створення систем управління навчанням (LMS), яка дозволяє організовувати дистанційне навчання, інтерактивні курси, форуми, тести та ведення журналів.

Вона підтримує гнучке налаштування ролей – викладачів, студентів, адміністраторів, а також має розгалужену систему оцінювання з можливістю формування зведеної інформації по учнях. Moodle орієнтований на університетське або курс-орієнтоване навчання та дистанційний формат. Його інтерфейс, хоч і гнучкий, часто є перевантаженим для шкільного середовища, особливо при роботі з мобільних пристроїв. Крім того, для базових шкільних потреб можливості Moodle можуть виявитися надмірними або надто складними в налаштуванні [2].

До менш відомих систем належить CanopyLAB. Це інтерактивна платформа для створення онлайн-курсів, яка орієнтується на позашкільне навчання та персоналізований досвід користувача. CanopyLAB використовується для самостійного навчання або у невеликих освітніх проєктах, де учні проходять курси у своєму темпі, взаємодіють із ментором, отримують бали та сертифікати. Хоч платформа й не призначена безпосередньо для оцінювання у шкільному сенсі, вона демонструє сучасні підходи до відображення прогресу учня, візуалізації результатів навчання та персоналізованої взаємодії. Її інтерфейс інтуїтивно зрозумілий і адаптований під молодіжну аудиторію, що є цінним прикладом для проєктування електронного щоденника з фокусом на UX [3].

Загалом, аналіз існуючих рішень дозволяє визначити основні функції, які повинен мати сучасний електронний щоденник. Серед них – ведення оцінок, формування і контроль домашніх завдань, розклад занять, персональні профілі користувачів, адаптивний дизайн для різних типів пристроїв, а також простий та інтуїтивно зрозумілий інтерфейс. У багатьох з проаналізованих систем спостерігаються проблеми з продуктивністю, застарілим дизайном і недостатньою мобільною підтримкою, що було враховано при створення розроблюваної системи.

## 1.3 Огляд архітектури SPA та SSR

Певний час у вебархітектурі панував єдиний, «класичний», підхід “Multi-Page Application”. Клієнтська частина складалась з повністю статичних, завчасно визначених html сторінок. Вміст цих сторінок зазвичай був незмінним, тому для його зміни, або будь-якої дії потребувалось перезавантаження сторінки. Кожне перезавантаження потребувало запиту до серверу поточної або нової сторінки повністю [5].

Розвиток скриптів, зокрема мови JavaScript, додала динаміки, але кожна сторінка знову завантажувалась з серверу, що негативно впливало на досвід користувача. З розвитком технологій домашніх комп’ютерів та браузерів значно зросла швидкість виконання скриптів на сторінках. Це дозволило винайти рішення вищезазначеної проблеми у вигляді архітектури “Single Page Application”.

### 1.3.1 “Single Page Application”

“Single Page Application” (далі SPA) – сучасний архітектурний підхід проєктування вебзастосунків, що уособлює ідею «інсталяції» застосунку безпосередньо у браузер. При першому запиті будь-якої сторінки сервер відправляє до браузеру усі сторінки одразу, що дозволяє не перезавантажувати сторінку зовсім. Головним показником SPA є реактивність компонентів, тобто можливість дуже швидко, або реактивно змінюватись у відповідь на дії користувача, що позитивно впливає на досвід користувача. Цей підхід робить застосунок схожим на мобільний або десктоп-застосунок. Колишні «скрипти» в такому підході перетворюються у повноцінну програмну систему керування сторінками. Під час навігації html-код сторінки «підмінюється» на інший, збережений прямо у пам’яті браузера. А так як не обов’язково змінювати всю сторінку такі застосунки широко використовують компонентний підхід. Будь-

яка частина сторінки є її «компонентом», який відповідає лише за свій вміст та може бути по різному використаний [6].

Даний підхід є досить революційним у вебпрограмуванні, з чого випливають його недоліки. Погано оптимізований застосунок уповільнює швидкість роботи у браузері, що анулює вище зазначену перевагу користувацького досвіду. Найгіршим же недоліком є питання пошукових сервісів, зокрема пошукового робота.

Пошуковий робот, або краулер, – це спеціальна програма пошукових систем, призначена для автоматичного обходу сайтів і збирання інформації для подальшої індексації. Він переходить за посиланнями, зчитує вміст сторінки, аналізує структуру HTML та текст, а отримані дані передає в індекс пошукової системи, що дозволяє користувачам знаходити вебресурс за відповідними запитами. У традиційних вебсайтах, де кожна сторінка існує як окремий HTML-документ, робота такого робота є простою: він заходить за URL та миттєво отримує повний статичний вміст для аналізу.

У випадку використання SPA підхід кардинально змінюється. При першому запиті користувач – так само як і пошуковий робот – отримує майже порожній HTML-файл, оскільки вся логіка та вміст сторінок формуються за допомогою JavaScript уже безпосередньо у браузері. Це означає, що контент «з’являється» тільки після виконання відповідних сценаріїв, а повноцінні HTML-сторінки фактично не надходять із сервера. Проблема полягає в тому, що пошукові роботи не завжди виконують JavaScript або роблять це частково, обмежено й повільно. У результаті вони можуть просто не побачити справжній вміст сторінки й зафіксувати лише той мінімальний HTML, який є до рендерингу.

Через це SPA-застосунки часто погано індексуються: пошуковий робот може не зрозуміти навігації між «сторінками», не зчитати динамічно згенерований контент і вважати сайт майже порожнім. Це суттєво знижує рейтинги ресурсу в пошукових системах або навіть робить його невидимим для пошуку. Саме тому проблема взаємодії SPA з пошуковими роботами

вважається одним із ключових недоліків даного підходу і потребує спеціальних технічних рішень, таких як серверний рендеринг або попереднє формування HTML-змісту спеціально для індексації.

### 1.3.2 Пошукова оптимізація

У кожного продукту є мета розробки і більшість вебсайтів мають на меті поширити свої дані у мережі або для усієї мережі, або у локальній її частині, та заробітку на популярності. Менша частина не має на меті заробіток через кількість відвідувачів. Тому чим більше людей відвідає сайт тим більше може бути заробіток. За статистикою 99 відсотків відвідувачів Google не шукають далі першої сторінки результатів [7]. Саме тому таким сайтам як магазини критично важливо бути на перших місцях пошуку. Саме для цього існує SEO, або Search Engine Optimization (пошукова оптимізація).

Одним з заходів такої оптимізації є семантичне покращення структури сторінки. Кожен елемент має своє призначення та мета-теги. Кожна кнопка, зображення чи текст може бути використаний під час пошуку. За обробку таких даних відповідає пошуковий робот (web crawler, або вебпавук). Це програма, яка сканує усі сторінки сайту з певними обмеженнями, серед яких є повне обмеження на виконання скриптових програм. Саме це обмеження не дозволяє односторінковим застосункам попасти «на першу сторінку». Такий застосунок на запит надсилає пусту сторінку у якій є лише кореневий контейнер. Увесь зміст «інсталюється» під час виконання основного скрипту, який запускається вже після завантаження сторінки. Тобто пошуковий робот «побачить» лише пусту сторінку, що повністю анулює будь яку іншу оптимізацію. Для подолання цієї проблеми впровадили просте рішення – рендеринг на стороні серверу [8].

Server-side rendering (далі SSR) – техніка вебпрограмування за якій, при першому запиті, сторінка генерується на стороні серверу та надсилається вже

готовою. При цьому зі сторінкою надсилаються і весь застосунок, тому далі усі переходи по сторінкам відбуваються без перезавантаження [9]. Натомість пошуковий робот «відвідує» сторінки завантажуючи їх окремо, тому для нього усі сторінки вже згенеровані сервером і оптимізовані для пошуку. Ця техніка викликає певне обмеження на розробку. Сервер, який генерує сторінки, має бути написаний на мові JavaScript.

У перших реалізаціях це працювало за таким алгоритмом:

а) генерація сторінки на сервері та відправка користувачу;

б) початок виконання основної програми застосунку;

в) повторна генерація поточної сторінки на теж місце;

г) заповнення компонентів даними та станами.

Очевидно, що подвійна генерація однієї сторінки не є ефективним рішенням, тому створили інструмент для заповнення завчасно згенерованої сторінки даними та станами – гідратацію.

Завдяки цьому алгоритм першого запиту сторінки стає більш ефективним:

а) генерація сторінки на сервері та відправка користувачу;

б) початок виконання основної програми застосунку;

в) заповнення компонентів даними та гідратація інтерактивних частин.

Обмеженням цього підходу є необхідність у повній відповідності згенерованої сторінки та очікуваної сторінки для гідратації.

### 1.4 Вибір технологій для реалізації

Одним із найпопулярніших реалізацій SPA є бібліотека для веброзробки React. Це сучасний набір інструментів, що охоплює усі потреби для зручної SPA розробки та компонентного підходу. React забезпечує високу продуктивність завдяки віртуальному DOM, що зменшує кількість реальних змін у DOM-дереві браузера [10].

Для розробки серверної частини було обрано фреймворк Next.js. Це фреймворк на основі React, який надає додаткові можливості для розробки, зокрема серверний рендеринг, статичну генерацію сторінок (одноразова генерація під час побудови застосунку), вбудовану систему маршрутизації, оптимізацію продуктивності, автоматичний поділ коду та інші переваги. Next.js дозволяє швидко поєднати SPA-логіку та серверну генерацію контенту для створення повноцінного застосунку.

Одними з найважливіших елементів будь-якого вебпроєкту є автентифікація – виявлення «особистості» користувача, його даних збережених у базі даних та його браузері, та авторизація – надання цьому користувачу прав доступу до ресурсів. Серед існуючих інструментів для впровадження автентифікації було обрано бібліотеку NextAuth.js за її глибоку інтеграцію з Next.js. Бібліотека використовує серверну маршрутизацію Next.js та забезпечує автентифікацію як через зовнішні OAuth-провайдери (Google, Facebook, GitHub тощо), так і через власну систему логіна з використанням електронної пошти та пароля. Впровадження автентифікації дозволить додати авторизацію через надання користувачам певних ролей, а саме «учень» та «викладач». Крім того бібліотека надає зручний контроль над сесіями користувача, зберігаючи їх у браузері клієнта, або прямо у базі даних серверу через спеціальний адаптер [11].

### 1.5 Збереження даних

Як базу для зберігання даних серверу було обрано одну з найпопулярніших систем керування з відкритим вихідним кодом MySQL. Вона забезпечує високу продуктивність, стабільність, масштабованість та добре підходить для організації зв’язків між сутностями, що забезпечить цілісність усієї структури [12].

Для взаємодії з базою було обрано ORM (Object-Relational Mapping) бібліотеку Sequelize. Написана на мові JavaScript, вона дозволяє використати

об’єктно-орієнтований підхід до сутностей бази та звертатись до неї через методи класів, створених на основі цих сутностей. Усі дії з сутностями бібліотека перетворює безпосередньо на sql запити та відправляє до бази. Однією з переваг є можливість «включення» у запитувану сутність також сутності, з якими у неї є зв’язки, що дозволяє зменшити кількість самих запитів та значно спростити бізнес-логіку [13].

Так як однією з особливостей архітектури SPA є повна, або часткова відсутність перезавантажень сторінки, а компоненти з яких складено сторінки отримують дані з серверу окремо, є важливим забезпечити відповідність цих даних між собою. Якщо один з компонентів змінить дані, копія яких є у іншому компоненті, вони зміняться лише у цьому компоненті.

Рішенням цього питання є впровадження єдиного, нормалізованого джерела даних, доступного для усіх компонентів застосунку. Тому для проєкту було обрано бібліотеку Redux. Вона дозволяє створити єдиний, нормалізований «стан» застосунку, структура якого повністю, або частково збігається з базою даних серверу [14]. Така синхронізація структур значно спрощує розробку через узгодженість між фронтендом та бекендом. Redux у поєднанні з додатками (redux-saga, react-redux, next-redux-wrapper) швидко і легко інтегрується у Next.js застосунок.

### 1.6 Адаптивний дизайн та Tailwind CSS

Навіть опинившись на першій позиції пошуку, щоденник має бути зручним та зрозумілим для використання. Саме тому дуже важливо створювати дизайн адаптивним до усіх можливих випадків користування, щоб щоденник залишався функціональним і зручним незалежно від того, з якого пристрою до нього звертається користувач. У сучасному освітньому середовищі доступ до ресурсів здійснюється з різноманітних пристроїв, від смартфонів до телевізорів, і всі вони мають різні розмір і форму екранів.

Найпростішим, але водночас неефективним підходом до адаптації є створення копій користувацького інтерфейсу для певних категорій пристроїв та додавання їм можливості «пристосовуватись» до різних форм. Але такий підхід провокує потенційне збільшення розміру застосунку вдвічі, що у контексті SPA є фатальним.

Натомість кращим рішенням є розробка гнучкої верстки сторінок, при якій структура сторінки залишається незмінною, а її стиль змінюється залежно від ширини екрану. Для цього у мові CSS, яка слугує мовою описання стилів відображення вебконтенту, впровадили різні інструменти, які дозволяють створювати адаптивний дизайн [15]. Але для пришвидшення розробки та підтримки адаптивності в електронному щоденнику було використано фреймворк Tailwind CSS – сучасну утилітарну CSS-бібліотеку, яка дозволяє стилізувати компоненти без написання власних CSS-файлів.

Tailwind CSS надає зручні та практичні класи, назва яких відповідає певному, часто атомарному стилю який він надає. Це дозволяє швидко розробляти адаптивний дизайн, використовуючи вбудовані фільтри різних екранів – sm, md, lg та xl, що відповідно є малим, середнім, великим та дуже великим екраном. Завдяки цьому інтерфейс електронного щоденника автоматично підлаштовується до ширини екрана: на маленькому екрані меню може згортатися, таблиці – спрощуватись, а компоненти – переміщуватись у вертикальні блоки [16].

Використання адаптивного дизайну з Tailwind CSS дозволило розробити інтерфейс електронного щоденника доступним і зручним як для викладача, який користується ноутбуком на роботі, так і для батьків, які перевіряють оцінки дитини з телефону дорогою додому. Це суттєво покращує користувацький досвід, зменшує бар’єри у взаємодії з системою та сприяє більш активному використанню щоденника всіма учасниками освітнього процесу.

# 2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ЕЛЕКТРОННОГО ЩОДЕННИКА

## 2.1 Проєктування бази даних

Одним із найважливіших етапів розробки електронного щоденника є проєктування бази даних. Саме ця частина визначає логіку зберігання, обробки та подальшої взаємодії з даними, які складають ядро системи. Правильно спроєктована база даних гарантує надійність і масштабованість застосунку, а також значно спрощує реалізацію подальшого функціоналу.

У системі електронного щоденника основними сутностями є користувачі, класи, предмети та оцінки. Для кожної сутності створюються окремі таблиці, пов’язані між собою зовнішніми ключами відповідно до правил нормалізації.

Так як користувачі мають різні ролі, зв’язок між ними та класами є many-to-many (багато до багатьох). Цей зв’язок вказує на те, що як сутність з однієї таблиці пов’язаний з багатьма сутностями іншої, так і навпаки, сутності другої вказують на багато сутностей першої. У контексті проєкту, один клас має декілька користувачів – учнів, а один учень може бути у декількох класів. Також записи класів можуть бути пов’язані з певним користувачем-викладачем, що відображає зв’язок one-to-many, або один до багатьох.

Сутність оцінки має багато зв’язків. Один запис водночас пов’язаний з користувачем-викладачем, який поставив цю оцінку, з користувачем-учнем, який її отримав, з предметом, за який вона отримана та класом, до якого належить учень. Таким чином уможливлюється створення журналів для викладачів, які можна сортувати за класами, предметами та окремими учнями.

Як спосіб візуального проєктування було використано ER-діаграму (діаграму зв’язків). Вона дозволяє наочно відобразити структуру даних, визначити основні сутності (таблиці), атрибути (поля), а також взаємозв’язки між ними. Це особливо актуально у випадку розробки електронного

щоденника, оскільки система повинна зберігати різнорідну інформацію: дані користувачів, навчальні групи, предмети, оцінки, відвідування, типи занять тощо. Створення чіткої та структурованої схеми бази даних дозволяє уникнути плутанини та дублювання інформації, а також забезпечує гнучкість подальшої розробки (див. рис. 2.1).

Саме на базі цієї структури формується централізоване сховище даних, до якого звертаються як серверні API, так і клієнт через стан застосунку. Помилки на цьому етапі проєктування неминуче призводять до складнощів при розробці інтерфейсу, впровадженні бізнес-логіки, а також ускладнюють технічне обслуговування.

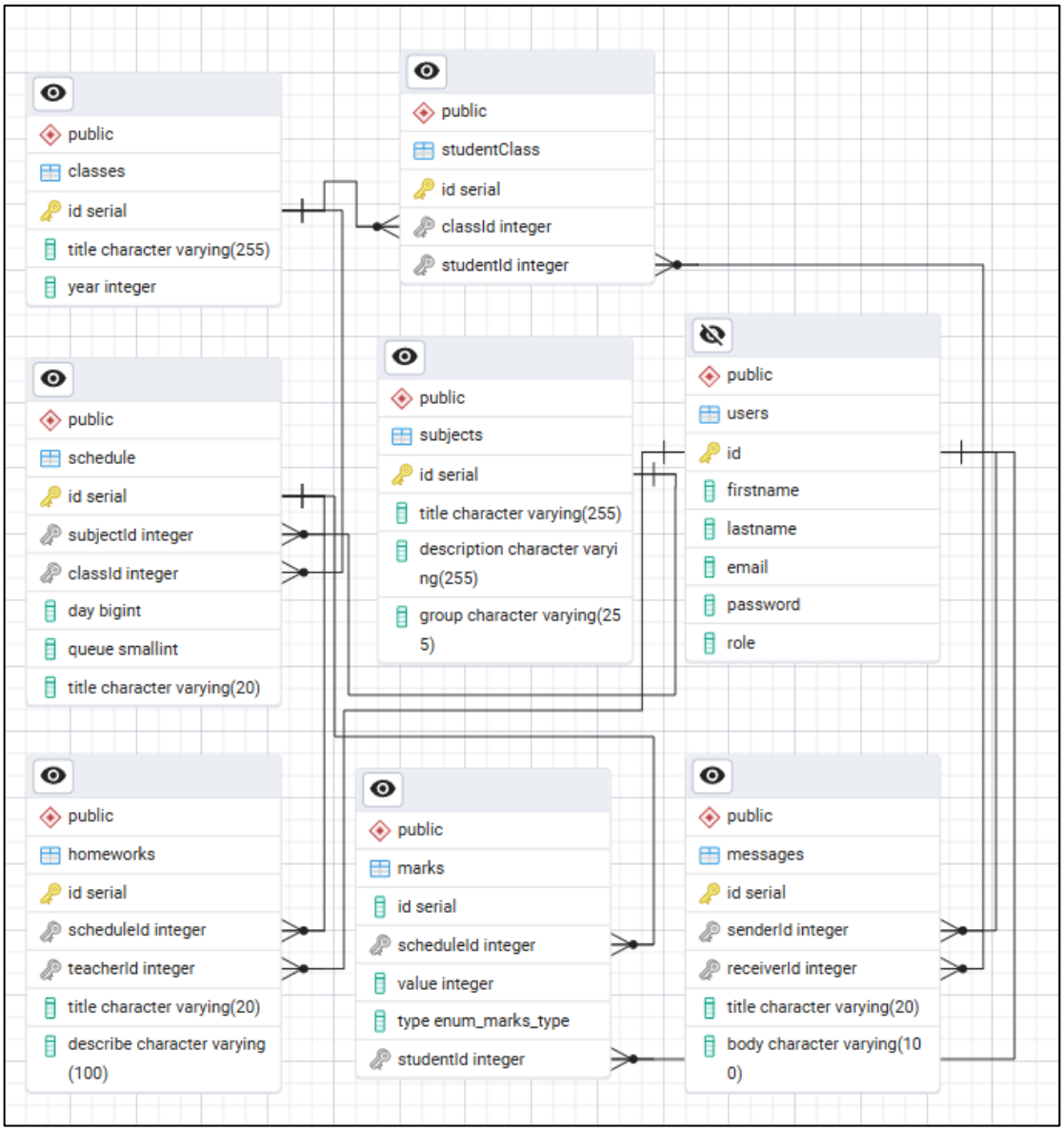



Рисунок 2.1 – ER-діаграма

## 2.2 Структура. MVC

Зі збільшенням масштабу проєкту важливішим організація чітко розмежованої структури коду, де окремі модулі незалежні і мають власне коло відповідальності. Методологія “Model, View, Controller”, або MVC, створена для впровадження такого підходу. Її суть полягає у реалізації трьох основних модулів застосунку: моделі, інтерфейсу та контролера.

Взаємодія цих компонентів діє за правилами:

– користувач може впливати на контролер;

– контролер працює над даними моделі, зазвичай базою даних;

– готові дані надаються інтерфейсу, який показує їх користувачу.

Використання такого підходу полегшить розробку великого проєкту та впровадить чітку структуру, яку потім легко зрозуміти при переході на новий проєкт. Використання стандартизованих підходів значно скорочує час адаптації нового розробника на проєкті, пришвидшує загальний час розробки та додає можливості для полегшення тестування та подальшої модифікації [16].

Окремим шаром між контролерами та моделями є сервіси, який містить бізнес-логіку. Вони дозволяють централізувати операції над даними (наприклад, створення оцінки, перевірку прав доступу, обчислення середнього балу тощо), які використовуються в кількох місцях застосунку. Це знижує навантаження на контролери, зменшуючи їх до перевірки даних запиту та простого виклику сервісу.

Використання SPA архітектури клієнтського застосунку відкриває можливість реалізації структури MVC. У якості відображення очевидно постає React застосунок, моделлю є сховище Redux, а контролерами є класи-нащадки BaseEntity, що наслідують методи для API-запитів та виконуються у «сагах» – асинхронних методах на основі «функцій-генераторів».

Таким чином, реалізація архітектури MVC розташована у папці server окремими папками controllers, models та services. Окремо слід виділити, що модулем View виступає уся MVC модель клієнтської частини.

Клієнтська частина, що розміщена у відповідній папці, у якості контролерів містить папку entities, а каталог store є модулем «Модель». Враховуючи залежність фреймворку Nextjs від файлової системи, модуль view розташований у папці pages, що лежить у корені проєкту. І цій папці також розміщені API шляхи в окремому каталозі. Хоча фреймворк надає інструменти для реалізації API, а саме використання шляху до файлу у каталозі як «шлях» API, було прийнято рішення реалізувати власну систему навігації API, хоча й засновану на правилах Nextjs (див. рис. 2.2).

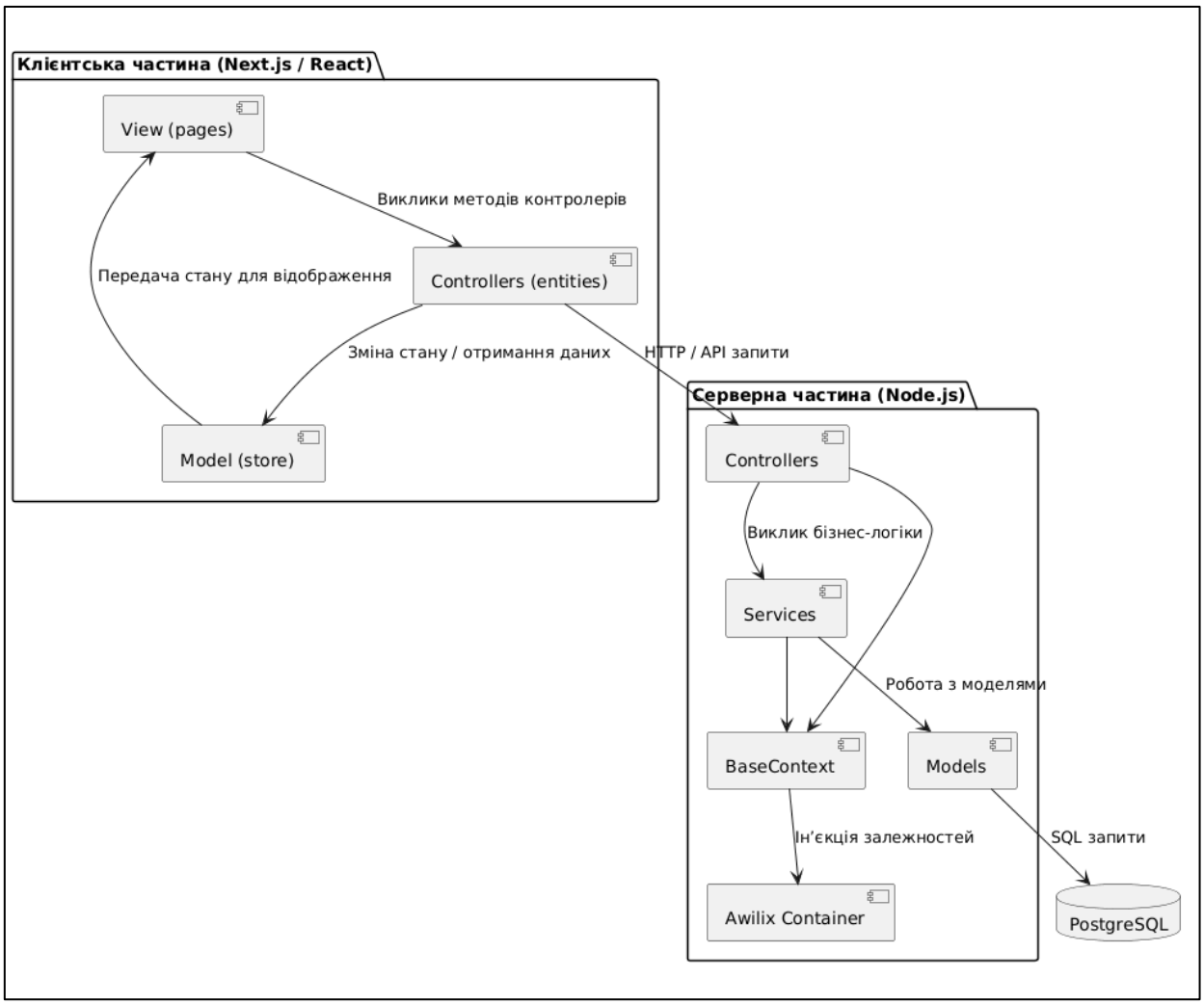



Рисунок 2.2 – Діаграма компонентів системи

Таким чином, розробивши централізоване та уніфіковане «ядро» проєкту, для повторного використання достатньо очистити реалізовані класи з відповідних каталогів та створити нові, які потребує новий проєкт.

## 2.3 Патерн проєктування Dependency Injection

При розширенні будь-якого проєкту, особливо при використанні MVC та подібних підходів, з’являється важлива проблема накопичення залежностей між різними компонентами системи. При запиті на отримання певної сутності створюється певний клас-контролер. Цей клас повинен мати доступ до класу-сервісу, який виконає бізнес логіку та надасть результат від класу-моделі, що відповідає запитуваній сутності. Також сервіси можуть викликати одне одного і мати доступ до різних моделей. Створення екземплярів заради кожного виклику не є правильним і тим-паче ефективним рішенням. Рішення, при якому усі класи створюються одразу також має недоліки. Перша ініціалізація кожного екземпляру класу значно збільшиться через кількість залежностей, які вимагає цей клас. Такий код важко налагоджувати та підтримувати, не кажучи про розширення, яке лише збільшить кількість залежностей. Щоб подолати цю проблему було використано дуже популярний патерн проєктування Dependency Injection.

Суть цього патерну полягає у продовженні ідеї вище-зазначеного рішення. Усі залежності класу не створюються всередині класу а передаються йому ззовні. Проте, щоб не перетворювати створення класу на великий список можливих залежностей, зазвичай, створюється окремий контейнер, усередині якого збережені посилання на усі можливі залежності, кожна з яких має одну єдину залежність, від самого контейнеру. Таким чином можна забути про вибір залежностей для кожного екземпляру. Надання доступу до контейнеру забезпечує впевненість, що усі потрібні залежності точно будуть передані [17].

Для впровадження цього патерну у проєкті було використано бібліотеку Awilix. Це проста та потужна бібліотека для впровадження Dependency Injection. У розроблювальній системі присутнє багато класів і усі залежності між ними поєднані у одному контейнері. Кожен клас має доступ до будь якого іншого класу або ресурсу. Це позбавило від необхідності власноруч додавати усі потрібні залежності, що також збільшує можливість до розширення коду та тестування окремих модулів.

## 2.4 Авторизація. JWT

Немало важливим аспектом програмного продукту є правила доступу до контенту. Кожен користувач може мати певну роль у системі, яка надає йому певні права та обмеження.

У сучасних вебзастосунках авторизація та автентифікація є ключовими елементами безпеки. Без належної системи контролю доступу користувачі можуть отримати доступ до конфіденційної інформації або здійснювати небажані дії в системі. Особливо це критично для застосунків, які працюють із персональними даними, фінансовою інформацією або приватним контентом.

Автентифікація (authentication) – це процес перевірки особи користувача, зазвичай за допомогою логіну та паролю, або через сторонні сервіси (OAuth, SSO тощо). Авторизація (authorization) – це процес перевірки прав доступу: що саме користувач має право робити після автентифікації.

Розділення цих процесів дозволяє створювати гнучкі системи безпеки, в яких можна контролювати доступ до різних частин застосунку залежно від ролі користувача, його статусу чи інших критеріїв.

JSON Web Token – це відкритий стандарт, який дозволяє безпечно передавати інформацію між клієнтом і сервером у вигляді токенів. JWT широко використовується для реалізації безстейтової автентифікації.

Вся інформація про сесію не зберігається безпосередньо на сервері а зберігається у токені. Завдяки цьому немає потреби робити додаткові запити до бази даних для кожної перевірки сесії, що позитивно впливає на швидкість усіх запитів.

JWT зазвичай складається з трьох частин: заголовка (header), тіла (payload) і підпису (signature). Payload може містити ID користувача, роль, час дії токена тощо. Підпис симетричним або асиметричним ключем гарантує цілісність і достовірність даних.

Для реалізації автентифікації на основі JWT було використано популярну бібліотеку next-auth, побудовану на фреймворку Next.js. Вона

забезпечує гнучке та безпечне рішення для інтеграції з різними провайдерами авторизації, включаючи Google, GitHub, Facebook та інші OAuth-провайдери, електронну пошту, кастомні системи логіну. next-auth може зберігати сесію як у базі даних, так і у вигляді JWT. Також бібліотека має повну підтримку Next.js API Routes, що дозволяє швидко налаштувати усі параметри та перейти безпосередньо до використання.

Серед переваг next-auth можна виділити високу швидкість впровадження, підтримку численних провайдерів (Google, GitHub, Discord, Facebook тощо), а також гнучкість через callback-механізми, які дозволяють втручатися у процес на різних етапах – наприклад, змінювати дані токена чи сесії. Крім того, бібліотека підтримує SSR (Server-Side Rendering), що робить сесію доступною під час серверного рендерингу – важливо для SEO-оптимізації та динамічних сторінок у Next.js.

Водночас next-auth має певні обмеження. Її внутрішня логіка не завжди очевидна, документація може бути неповною, а реалізація нестандартних сценаріїв, таких як багатофакторна автентифікація або власна логіка оновлення токенів, може викликати труднощі.

## 2.5 Моделювання засобами UML

Unified Modeling Language (UML) – це уніфікована мова моделювання, яку застосовують для опису та візуалізації різних елементів програмного забезпечення. У вебпроєктах UML допомагає впорядкувати уявлення про архітектуру системи, окреслити її ключові складові та показати, як вони взаємодіють між собою і з бізнес-логікою. Завдяки діаграмам можна наочно подати структуру модулів, користувачів, окремих процесів і ті сценарії, які формують загальну поведінку системи. У цьому проєкті UML-діаграми дали змогу формально відобразити основні елементи – користувачів, контролери, сервіси та моделі даних – і показати їхні взаємозв’язки у зручній візуальній формі.

Діаграма прецедентів демонструє, які можливості надаються різним категоріям користувачів: адміністратору, викладачу й учню. На ній видно, які саме дії дозволені кожній ролі – перегляд розкладу, створення або перевірка домашніх завдань, виставлення оцінок, надсилання повідомлень тощо. Такий підхід допоміг чітко розмежувати відповідальності та структурувати сценарії взаємодії з системою.

Для відображення внутрішньої організації серверної частини застосунку використано діаграму класів. Вона показує, як пов’язані між собою контролери, сервіси та інші ключові модулі. Наприклад, контролери взаємодіють із відповідними сервісами, а ті – із базою даних через спільний контекст BaseContext.

### 2.5.1 Діаграма варіантів використання

Діаграма варіантів використання (use case diagram) – це один із типів діаграм UML, який призначений для відображення взаємодії користувачів із системою через опис її функціональності у вигляді варіантів використання. Основними елементами цієї діаграми є актори (користувачі або зовнішні системи) та варіанти використання, які позначають функції або сервіси, що система надає.

На діаграмі відображено основні ролі користувачів системи електронного щоденника та їх взаємодію із системою (див. рис. 2.3). У системі виділено три основних актори: адміністратор, викладач та учень. Кожен із них має власний набір можливостей, що відповідає його ролі у навчальному процесі.

Адміністратор має найширші права доступу. Він може редагувати користувачів та класи, а також переглядати списки користувачів і класів. Це дає змогу керувати структурою системи, додавати нові облікові записи, формувати навчальні групи та підтримувати актуальність даних.

Викладач взаємодіє з даними навчального процесу. Він може створювати розклад, оцінки та домашні завдання, а також переглядати розклад, оцінки і домашні завдання своїх учнів.

Учень має доступ до інформації, що стосується його навчання. Він може переглядати розклад, домашні завдання та оцінки. Це забезпечує постійний доступ до актуальних даних і дозволяє самостійно контролювати навчальний прогрес.

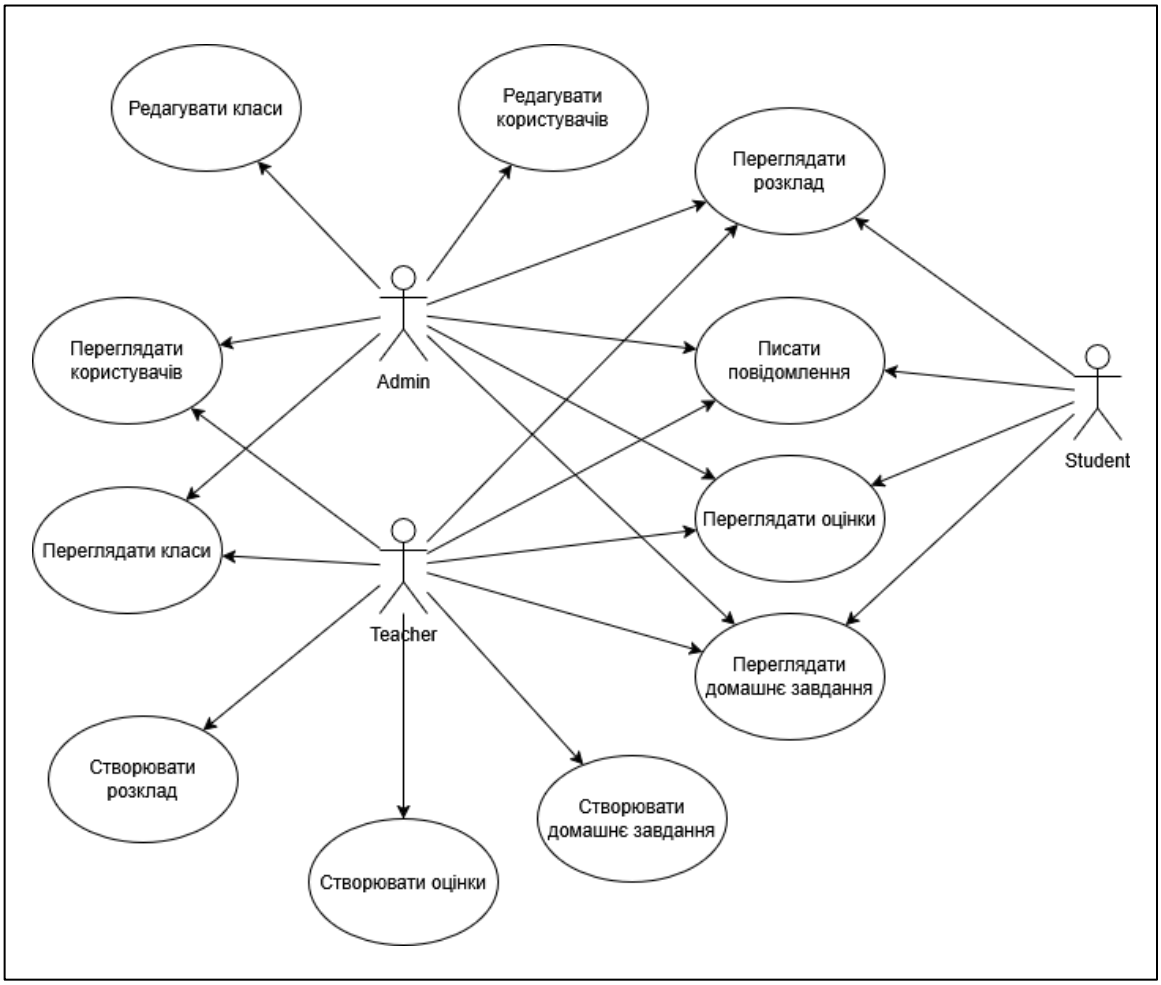



Рисунок 2.3 – Діаграма варіантів використання

Діаграма відображає ієрархію доступу та чітке розмежування прав між ролями. Усі дії згруповано за основними напрямами – управління користувачами та класами, робота з навчальним процесом (оцінки, розклад, завдання) і комунікація через повідомлення. Така структура забезпечує логічну організацію функціональності та зручність у використанні системи.

### 2.5.2 Діаграма класів

Діаграма класів (class diagram) використовується для моделювання структури системи з точки зору об’єктно-орієнтованого програмування. Вона відображає класи, їхні атрибути, методи (операції), а також зв’язки між класами – такі як асоціації, агрегації, композиції та наслідування.

Діаграма класів дозволяє створити чітке уявлення про логічну структуру системи, що особливо важливо при розробці складних вебпроєктів. Вона дає змогу проаналізувати, як саме об’єкти системи взаємодіють між собою, яка інформація зберігається в кожному класі та які операції можуть виконуватись.

Такий рівень деталізації полегшує написання коду, виявлення потенційних помилок у проєктуванні та забезпечує узгодженість між різними компонентами додатку. Окрім цього, діаграма класів слугує корисною документацією для подальшого супроводу та масштабування проєкту. Завдяки своїй структурованості та точності вона є незамінним інструментом при плануванні архітектури вебсистем.

На серверній частині застосунку реалізовано багаторівневу архітектуру, у якій контролери відповідають за обробку запитів користувачів, виклик відповідних сервісів та повернення результатів клієнту. Основою побудови цієї частини є принципи інверсії залежностей та спадкування, що забезпечують гнучкість, масштабованість і повторне використання коду.

Клас BaseContext використовується для ініціалізації контейнера залежностей (Dependency Injection). Він містить екземпляр IServerContainer, через який контролери отримують доступ до сервісів (див. рис. 2.4) і модулів системи (див. рис. 2.5).

Клас BaseController є спільною базою для всіх контролерів системи (див. рис. 2.6). Він реалізує основну логіку роботи з HTTP-запитами та відповідає за маршрутизацію, обробку параметрів і виконання відповідних дій. Завдяки цьому всі контролери мають уніфікований інтерфейс, що спрощує їх інтеграцію в систему та зменшує кількість дублювання коду.

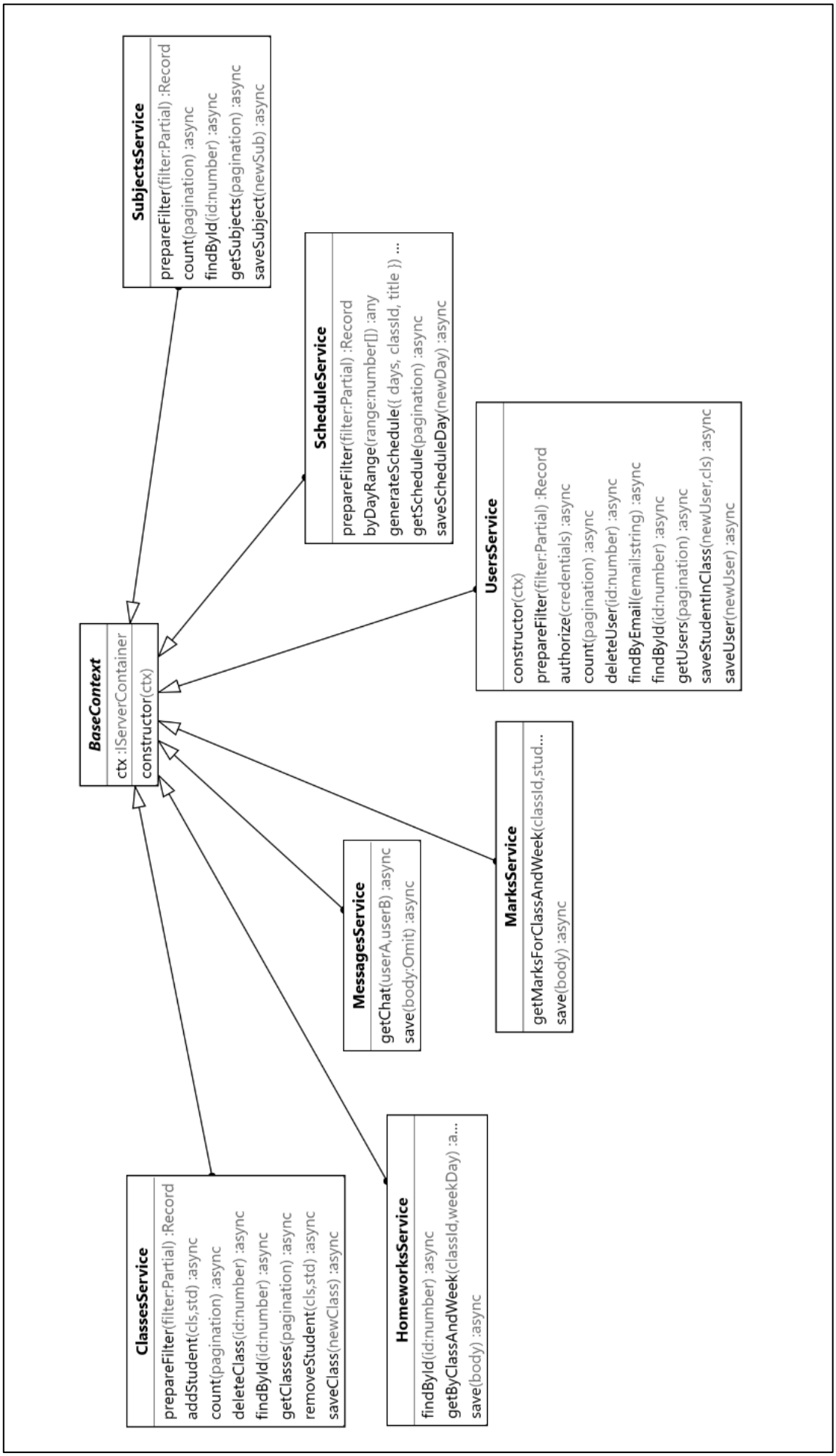



Рисунок 2.4 – Діаграма класів сервісів

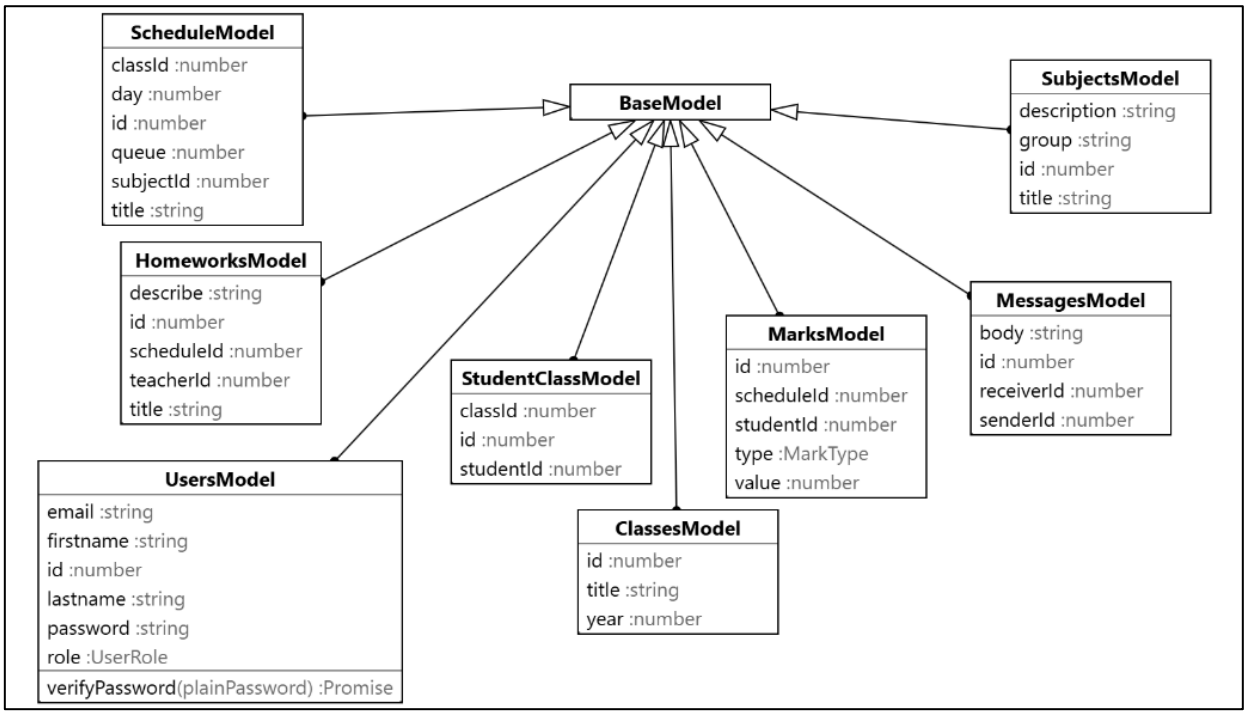



Рисунок 2.5 – Діаграма класів моделей

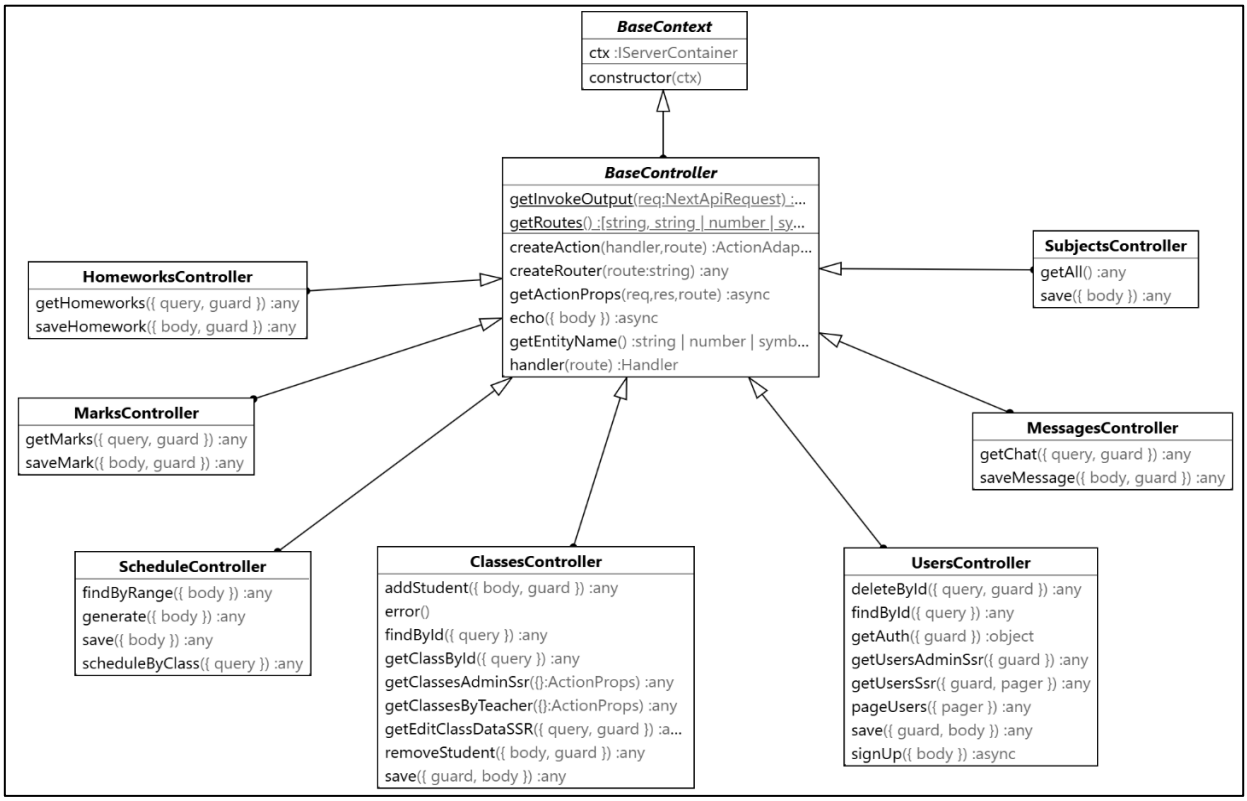



Рисунок 2.6 – Діаграма класів контролерів

Основні методи класу:

– getInvokeOutput(req) – викликає необхідну дію контролера та повертає результат обробки запиту;

– getRoutes() – формує список маршрутів, які підтримує поточний контролер;
– createAction(handler, route) – створює адаптер дій, який зв’язує маршрут із відповідним методом;
– createRouter(route) – створює маршрутизатор для підключення контролера до серверу;
– getActionProps(req, res, route) – отримує параметри запиту;
– getEntityName() – визначає сутність, з якою працює контролер;
– handler(route) – повертає обробник для конкретного маршруту.

На основі BaseController реалізовано низку спеціалізованих контролерів, кожен з яких відповідає за конкретну частину функціоналу системи:

– UsersController – робота з користувачами;
– ClassesController – управління класами та учнями;
– SubjectsController – робота з навчальними предметами;
– ScheduleController – керування розкладом занять;
– MarksController – обробка оцінок;
– HomeworksController – управління домашніми завданнями;
– MessagesController – робота з повідомленнями.

Усі контролери наслідують базовий клас BaseController, який, у свою чергу, використовує BaseContext для доступу до контейнера залежностей. Така ієрархічна структура дозволяє централізовано керувати логікою обробки запитів та уникати дублювання коду.

Кожен контролер реалізує набір методів, що відповідають за CRUD-операції (Create, Read, Update, Delete) та взаємодію з клієнтською частиною через HTTP-запити.

Завдяки використанню єдиної архітектури рівень контролерів легко розширюється. Створення нового контролеру відбувається успадкуванням базового класу, визначенням власних кінцевих API точок та відповідних методів. Такий підхід забезпечує модульність, зрозумілу структуру та гнучкість у розробці серверної частини застосунку, дозволяє ефективно

розподіляти обов’язки між компонентами системи, спрощує супровід коду та сприяє розширюваності проєкту без порушення існуючої логіки.

За безпосередню взаємодію з базою даних, виконання бізнес-логіки та підготовку даних для контролерів відповідає сервісний рівень застосунку. Кожен сервіс реалізує окрему частину функціональності електронного щоденника, працюючи з певною сутністю – користувачами, класами, предметами, розкладом, оцінками, домашніми завданнями чи повідомленнями, але не обмежуючись лише ними. Як і контролери, усі сервіси побудовані за принципом інверсії залежностей та використовують спільний клас BaseContext, що надає доступ до контейнера залежностей і необхідних ресурсів системи.

### 2.5.3 Діаграма послідовностей

Діаграма послідовностей (sequence diagram) – це тип діаграми UML, який використовується для моделювання динамічної взаємодії між об’єктами або компонентами системи в часі. Вона показує, як об’єкти обмінюються повідомленнями (викликами методів) у межах певного сценарію або функціонального процесу. Основними елементами діаграми є об’єкти (або учасники), які розміщуються горизонтально у верхній частині діаграми, та повідомлення, що передаються між ними, представлені стрілками.

На поточній діаграмі послідовності зображено процес обробки API-запиту у межах архітектури Model-View-Controller + Service (див. рис. 2.7). Процес починається з того, що клієнт (інтерфейс користувача або фронтенд-застосунок) надсилає запит до сервера. Цей запит потрапляє до шару Controller, який відповідає за обробку вхідних запитів.

Далі, контролер виконує валідацію даних – перевіряє коректність вхідних параметрів і наявність необхідних полів. Після успішної перевірки контролер передає запит далі до Service, де реалізовано основну бізнес-логіку

застосунку. Сервісний шар обробляє запит, виконує необхідні операції, формує параметри та звертається до Model, яка відповідає за роботу з базою даних. Модель отримує запит, виконує операцію (наприклад, читання, оновлення чи створення запису) та повертає результат сервісу. Після отримання відповіді від моделі, сервіс узагальнює дані та повертає їх до контролеру, що формує структуровану відповідь через спільний базовий метод та повертає результат обробки клієнту у вигляді відповіді API.

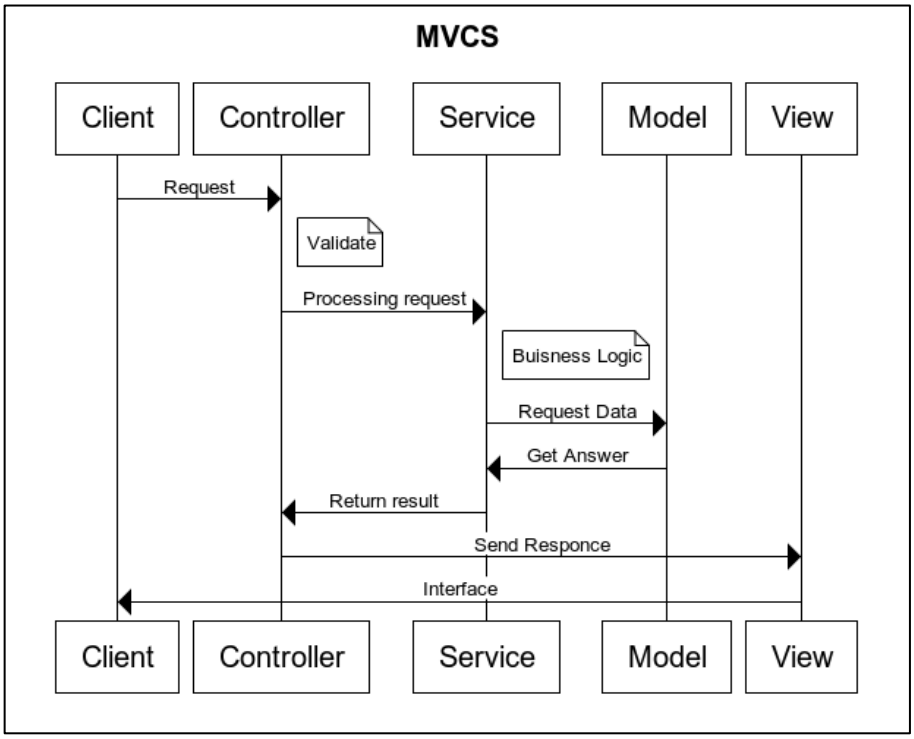



Рисунок 2.7 – Діаграма послідовності обробки API запиту

Таким чином, діаграма демонструє чітку послідовність дій та взаємодію між компонентами системи під час виконання одного API-запиту. Такий підхід забезпечує розділення відповідальностей, що робить код легшим у супроводі, тестуванні й розширенні.

# 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Налаштування середовища розробки

Для початку було створено новий проєкт за допомогою команди npx create-next-app, після чого сформувалася стандартна структура директорій Next.js. Надалі були видалені непотрібні приклади компонентів та сторінок, а також налаштована базова конфігурація типів у tsconfig.json.

Після ініціалізації проєкту було додано контейнер залежностей Awilix. У кореневій частині застосунку, у папці server створено папку container, де розміщено файл ініціалізації контейнера та реєстрація сервісів. Контейнер створюється один раз під час запуску сервера, після чого він передається у компоненти та модулі, яким необхідний доступ до сервісів. Це дозволяє централізовано управляти створенням екземплярів класів.

Для зручності взаємодії з контейнером було створено базовий клас BaseContext. Він містить посилання на контейнер та надає доступ до зареєстрованих сервісів через властивості класу.

В подальшому всі класи, які потребують доступу до сервісів (наприклад, модулі роботи з базою даних або бізнес-логіка), успадковують BaseContext. Завдяки цьому, замість створення екземплярів вручну, сервіси завжди отримуються через контейнер.

Таким чином, на цьому етапі було сформовано базову структуру застосунку: створено проєкт Next.js, налаштовано контейнер Awilix та введено єдиний базовий клас для доступу до сервісів. Це забезпечило однаковий підхід до роботи з логікою системи у всіх модулях, що було важливо перед початком реалізації основної функціональності електронного щоденника.

## 3.2 Реалізація бекенду (API)

Бекенд у застосунку реалізовано у вигляді API-ендпоінтів, які обробляються на стороні сервера Next.js. Усі запити проходять через контролери, що відповідають за роботу з даними та взаємодію з сервісами.

### 3.2.1 Робота з базою даних

Для підключення до бази даних був створений окремий файл, який відповідає за формування та зберігання єдиного екземпляру з'єднання (див. рис. 3.1).

```
function dbConnectionFactory({ config }: IServerContainer) {
  const sequelize = new Sequelize(config.dbUrl);
  sequelize.sync();
  return sequelize;
}
```

Рисунок 3.1 – Підключення бази даних

У цьому файлі описано параметри доступу до бази, а також логіку ініціалізації клієнта. Такий підхід дозволив уникнути повторного створення з'єднання при кожному запиті та забезпечив взаємодію із БД у всіх модулях застосунку.

Після цього було додано моделі доступу до таблиць, які використовують це з'єднання для виконання операцій вибірки, запису, оновлення та видалення даних.

Кожен клас моделі працює з конкретною сутністю, наприклад користувачами, класами або оцінками (див. рис. 3.2).

```
export class ClassesModel extends BaseModel {
  declare id: number;
  declare title: string;
  declare year: number;
}
const ClassesModelFactory = (ctx: IServerContainer) => {
  ClassesModel.init(
    {
      id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
        allowNull: false,
      },
      title: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      year: {
        type: DataTypes.INTEGER,
        allowNull: false,
      },
    },
    {
      sequelize: ctx.db,
      modelName: "ClassesModel",
      tableName: "classes",
      timestamps: false,
      underscored: true,
    },
  );
  return ClassesModel;
};
```

Рисунок 3.2 – Ініціалізація моделі таблиці класів

### 3.2.2 Створення бази даних у pgAdmin 4

Для зберігання даних застосунку було використано систему PostgreSQL. Створення бази даних виконувалося через інструмент pgAdmin 4. Після запуску pgAdmin було виконано підключення до локального сервера PostgreSQL та відкрито консоль керування об’єктами.

У дереві серверів було створено нову базу даних для застосунку. Після цього в ній сформовано необхідні таблиці відповідно до логічної структури

електронного щоденника. Таблиці створювалися вручну через SQL-запити, що дозволило чітко визначити поля, типи даних та зв’язки між сутностями.

Було створено такі основні таблиці: користувачі, класи, предмети, розклад, домашні завдання, оцінки та повідомлення. Кожна таблиця отримала первинний ключ у вигляді цілочисельного ідентифікатора з автоінкрементом. Для встановлення зв’язків між таблицями використовувалися зовнішні ключі, які забезпечують відповідність записів, наприклад: учні прив’язані до класу, оцінки – до учня та предмета, розклад – до класу й предмета.

Додатково були налаштовані індекси для полів, які використовуються при пошуку та сортуванні. Це дозволило підвищити швидкість обробки запитів, особливо при роботі зі списками учнів, оцінок або розкладу.

Після створення структури база даних була наповнена тестовими записами для перевірки коректності операцій вставки, оновлення та вибірки даних. Це дало змогу переконатися, що всі таблиці пов’язані правильно, а контролери й API працюють із ними коректно.

Таким чином, на цьому етапі була сформована база даних із визначеною структурою сутностей і зв’язків, яка слугує основою для роботи застосунку та обміну даними між його компонентами.

### 3.2.3 Контролери та маршрути

Для обробки API-запитів використано підхід з динамічним маршрутом. У директорії pages/api був створений файл […slug].ts. Він приймає всі запити до API і виконує розбір шляху, визначаючи, до якого контролера необхідно звернутися. Це дозволило зосередити логіку маршрутизації в одному місці та уникнути дублювання коду.

Для зручності роботи було створено базовий контролер. Він містить загальний метод обробки запитів: приймає параметри, перевіряє тип HTTP-запиту та викликає відповідний метод бізнес-логіки. Такий підхід зробив структуру контролерів однаковою та спростив їх розширення.

Окремо була додана функція-фабрика для getServerSideProps, яка забезпечує доступ контролерів до контейнера залежностей на стороні сервера. Це дозволило отримувати необхідні сервіси без прямого імпорту та зберегти модульність коду (див. рис. 3.3).

```
export default function getServerSidePropsContainer(
  ctx: IContextContainer,
): GSSPFactory {
  return (controllersNames, isPublic = false, route?) =>
    redux.getServerSideProps((store) => async (context) => {
      // init
      …
      if (!isPublic && !req.session) {
        return {
          redirect: { destination: "/403", permanent: true },
        };
      }
      const identity = req.session?.identity ?? GUEST_IDENTITY;
      const auth: AuthState | null = { identity };
      store.dispatch(setAuth(auth));
      // collect promises from controllers
      const promises = controllersNames.map((name) => {
        return ctx[name]
          .handler(route)(req, context.res as NextApiResponse)
          .then((r) => {
            // saving action result to store
            }
          });
      });
      // run promises
      try {
        await Promise.all(promises);
      } catch (error) {
        console.error("GSSP ERROR: ", (error as Error).message);

        if (error instanceof AccessDeniedError) {
          console.info("\n\nRedirect because access denied");
          return { redirect: { destination: "/403", permanent: true } };
        }
        // handle other errors
      // end
      return {
        props,
      };
    });
}
```

Рисунок 3.3 – Фабрика для getServerSideProps

Після створення базової інфраструктури було додано контролери для основних сутностей системи: користувачів, класів, предметів, оцінок, розкладу та домашніх завдань. Кожен контролер реалізує набір дій над відповідною сутністю – створення, оновлення, видалення та отримання даних. Всі контролери працюють за єдиними правилами та наслідують одну базову логіку.

### 3.2.4 Реалізація автентифікації

Для реалізації автентифікації було створено окремий конфігураційний файл, у якому визначено параметри авторизації, зокрема секретний ключ та термін дії сесії. У цьому файлі описано базові налаштування для створення та перевірки токенів, що використовуються під час доступу користувачів до захищених розділів застосунку.

Після налаштування конфігурації був створений окремий API-ендпоінт, який відповідає за авторизацію користувача. Він приймає дані форми входу, перевіряє правильність облікових даних у базі даних і, у випадку успішної перевірки, формує токен доступу. Токен передається клієнту та зберігається у cookie, що дозволяє підтримувати авторизований стан між запитами.

У разі, якщо дані введені некоректно, API повертає відповідне повідомлення про помилку без створення токена. Це забезпечує контроль доступу до особистих даних та дозволяє працювати з профілем тільки після успішного входу.

Таким чином, на цьому етапі було налаштовано обробку авторизації користувачів та встановлено механізм перевірки доступу до захищених ресурсів у системі електронного щоденника.

Система підтримує три основні ролі користувачів:

– адміністратор – має повний доступ до всіх даних, може створювати класи, призначати учителів і додавати користувачів;

– учитель – може виставляти оцінки, додавати домашні завдання, формувати розклад та надсилати повідомлення;
– учень – має доступ до власного профілю, оцінок, розкладу, завдань і повідомлень.

Кожен користувач під час входу до системи отримує доступ до функцій відповідно до своєї ролі. Ролі зберігаються у базі даних у таблиці Users у полі role. Для автентифікації було використано Email/Password Provider з кастомним обробником перевірки даних користувача у базі (див. рис. 3.4).

```
export function authOptionsFactory(ctx: IServerContainer): AuthOptions {
  return {
    providers: [
      Credentials({
        name: "Credentials",
        credentials: {
          email: { type: "email", placeholder: "example@email.com" },
          password: { label: "Password", type: "password" },
        },
        async authorize(credentials) {
          try {
            const user: IUser = await ctx.UsersService.authorize(credentials);
            return user;
          } catch (e) {
            console.error(e);
            return null;
          }
        },
      }),
    ],
  };
}
```

Рисунок 3.4 – Налаштування провайдеру авторизації

Після успішної автентифікації Next-Auth створює сесію користувача, яка зберігається у JWT-токені. Завдяки цьому користувач може виконувати запити до API без повторного входу до системи. Для безпеки всі токени підписуються секретним ключем, який зберігається у змінних середовища (.env файл).

На стороні клієнта інтеграція з Next-Auth здійснюється через хук useSession(). Це дозволяє динамічно змінювати інтерфейс залежно від ролі користувача. Наприклад, учитель бачить кнопку “Додати оцінку”, а учень – лише список власних оцінок.

Також реалізовано можливість виходу із системи через метод signOut() бібліотеки Next-Auth, після чого сесія видаляється, а користувача перенаправляють на сторінку входу.

Це забезпечує надійну основу для подальшої роботи із захищеними даними, такими як оцінки, завдання та повідомлення, гарантуючи конфіденційність і безпеку користувачів.

## 3.3 Реалізація клієнтської частини

На клієнтській частині було повторно застосовано підхід із контейнером залежностей Awilix. Для цього було створено окремий контейнер, який ініціалізується під час рендерингу застосунку на клієнті та під час серверного рендерингу. Як і у випадку з бекендом, для доступу до контейнера був створений базовий клас BaseContext. Він надає можливість отримувати сервіси та модулі без прямого імпорту, що забезпечує однакову структуру роботи у всіх частинах застосунку.

### 3.3.1 Redux-сховище та структура стану

Стан застосунку було реалізовано на основі Redux (див. рис. 3.5). Такий підхід дозволив точніше контролювати створення ред’юсерів, middleware та підключення саг. Було створено окремі директорії для дій, ред’юсерів, саг та селекторів. Сховище підтримує роботу як на клієнті, так і на сервері, що забезпечує коректну ініціалізацію стану під час серверного рендерингу сторінок.

```
state: {
  entities: {
    users: Record<number, IUser>,
    classes: Record<number, IClass>,
    subjects: Record<number, ISubject>,
    schedule: Record<number, ISchedule>,
    homeworks: Record<number, IHomework>,
    marks: Record<number, IMark>,
    messages: Record<number, IMessage>
  };
  auth: AuthState | null;
  pagination: {
    [key: string]: {
      entityName: string;
      pageName: string;
      currentPage: number;
      count: number;
      perPage: number;
      filter?: { [key: string]: any };
      sort?: ISortParams;
      pages?: { [key: number]: [number] };
      touched?: number[];
      fetching?: boolean;
    };
  };
}
```

Рисунок 3.5 – Структура загального стану проєкту

Блок entities використовується як єдине джерело даних у форматі словників. Це дозволило уникнути дублювання інформації та спростило оновлення стану при отриманні нових даних із сервера. Розділ pagination відповідає за поточні параметри сторінкової навігації, що використовується при завантаженні великих списків даних.

### 3.3.2 Функція-фабрика для ред’юсерів

Щоб уникнути дублювання коду для кожної сутності, була створена фабрика ред’юсерів (див. рис. 3.6). Вона отримує ім’я сутності й повертає

ред’юсер, який уміє обробляти стандартні дії: додавання, оновлення та видалення записів у entities. Це значно скоротило кількість коду при додаванні нових сутностей у систему.

```
function entityReducer<K extends keyof Entities>(collectionName: K) {
  return (collection: Entities[K] = {}, action: EntitiesAction): Entities[K] => {
    switch (action.type) {
      case ADD_ENTITIES: {
        const newEntities = action.payload[collectionName];
        return { ...collection, ...newEntities };
      }
      case "CLEAR": {
        return initialState;
      }
    }
    return collection;
  };
}
export default entityReducer;
```

Рисунок 3.6 – Фабрика для ред’юсеру колекції

Сховище Redux аналізує усі ред’юсери та перевіряє посилання на об’єкт, який повертає функція, таким чином перевіряється зміна стану. Важливим є повернення або незміненого об’єкту стану, або нового об’єкту, адже зміна та повернення оригінального об’єкту викликає ефект «мутації». Сховище просто не помітить змін усередині об’єкту та не викличе подію оновлення.

### 3.3.3 Базовий клас для саг

Для організації бізнес-логіки на рівні Redux-Saga був створений базовий клас BaseEntity (див. рис. 3.7). Він містить узагальнені методи для взаємодії з API:

– xFetch – приватний метод для виконання HTTP-запитів;

– xRead та xWrite – методи для читання та запису даних;

– actionRequest – обробка запиту через Redux-сагу;

– pageEntity – отримання даних із підтримкою пагінації.

Також клас визначає метод для отримання списку саг поточної сутності та метод для формування ред’юсера. Завдяки цьому достатньо створити новий клас, що наслідує BaseEntity, вказати ім’я сутності та реалізувати специфічні методи. Уся інфраструктура запитів, стану та оновлення даних підключається автоматично.

```
abstract class BaseEntity extends BaseContext {
  protected abstract schema: Schema;

  public normalize (data: object): void;

  public static sagas(di: IClientContainer): Function[];

  public sagas(): Function[];

  private static async xFetch(
    url?: string,
    method?: "GET" | "POST",
    body?: any,
    force?: boolean,
  ): Promise;

  private *actionRequest(
    method: "GET" | "POST",
    url?: string,
    body?: any,
    force?: boolean,
  ): Generator<Effect>;

  public xRead(
    url?: string,
    method: "GET" | "POST" = "GET",
    body?: object,
    force?: boolean,
  ): Generator;

  public xSave(url: string, body: any = {}): Generator;

  public xDelete(url?: string, method: "GET" | "POST" = "GET"): Generator;

  public *pageEntity(uri: string, params: IPagerParams): Generator<Effect>;
```

Рисунок 3.7 – Оголошення класу BaseEntity

### 3.4 Головна сторінка

Головна сторінка застосунку слугує основною точкою навігації по даним та об’єднує відразу кілька ключових елементів: список класів, список учнів вибраного класу, таблицю розкладу на поточний тиждень та їх оцінки. Вся інформація на сторінці завантажується через Redux-сховище із використанням саг, що забезпечує синхронне оновлення даних без перезавантаження браузера (див. рис. 3.8).

У верхній частині сторінки розміщується список доступних класів. Після вибору класу викликається відповідна сага, яка завантажує список учнів цього класу, а також пов’язані з ним дані розкладу та оцінок. Завдяки цьому зміна одного параметра не потребує ручного оновлення всієї сторінки – дані оновлюються автоматично через стан Redux.

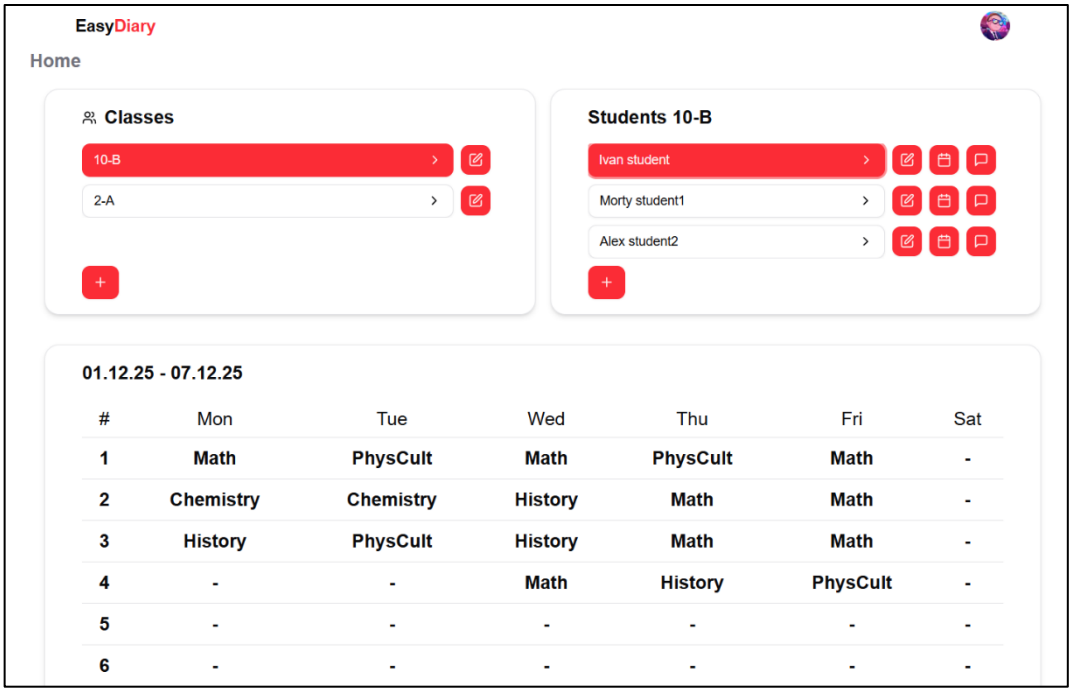



Рисунок 3.8 – Головна сторінка

Праворуч або нижче (залежно від адаптивності інтерфейсу) відображається список учнів вибраного класу. Кожен учень виділяється окремим рядком і може бути вибраний для перегляду детальнішої інформації

або внесення оцінок. Структура списку спирається на дані з entities.users, що дозволяє швидко отримувати доступ до інформації без додаткових запитів.

Нижче розташована таблиця розкладу на поточний тиждень. Вона формується на основі списку предметів та розкладу з entities.schedule. Для кожного дня тижня відображається перелік уроків із зазначенням предмета та викладача. Побудова таблиці виконується динамічно, з урахуванням прив’язки конкретних предметів до вибраного класу.

У разі, якщо користувач авторизований у ролі викладач, головна сторінка надає йому можливість вибору класу та конкретного учня зі списку. Кнопка редагування класу перенаправляє на окрему сторінку, де можна змінити характеристики класу, додати або видалити учнів, а також автоматично згенерувати розклад навчального тижня. Редагування або створення нового учня здійснюється через модальне вікно, що забезпечує швидку взаємодію без переходу на інші сторінки. Додатково доступні кнопки переходу до чату та до сторінки перегляду щоденника. Якщо ж користувач є учнем, система автоматично підставляє його клас і ім’я у відповідні списки, а всі елементи інтерфейсу, що відповідають за редагування, стають недоступними. Це гарантує коректність даних і забезпечує різні сценарії використання залежно від ролі користувача.

### 3.5 Сторінка щоденника

Сторінка електронного щоденника є одним із ключових елементів застосунку, оскільки забезпечує учням, батькам та викладачам зручний доступ до інформації про заняття, домашні завдання та отримані оцінки. На сторінці відображається назва активного класу та вибраного учня, що одразу вказує, з даними якого учня працює користувач (див. рис. 3.9).

У центральній частині інтерфейсу розташовано блок навігації тижнями. Користувач може перемикатися між поточним, попередніми та наступними

навчальними тижнями за допомогою стрілок «вперед» і «назад». Поточний тиждень відображається у форматі початкова дата – кінцева дата, що дозволяє легко орієнтуватися у часовому контексті щоденника.

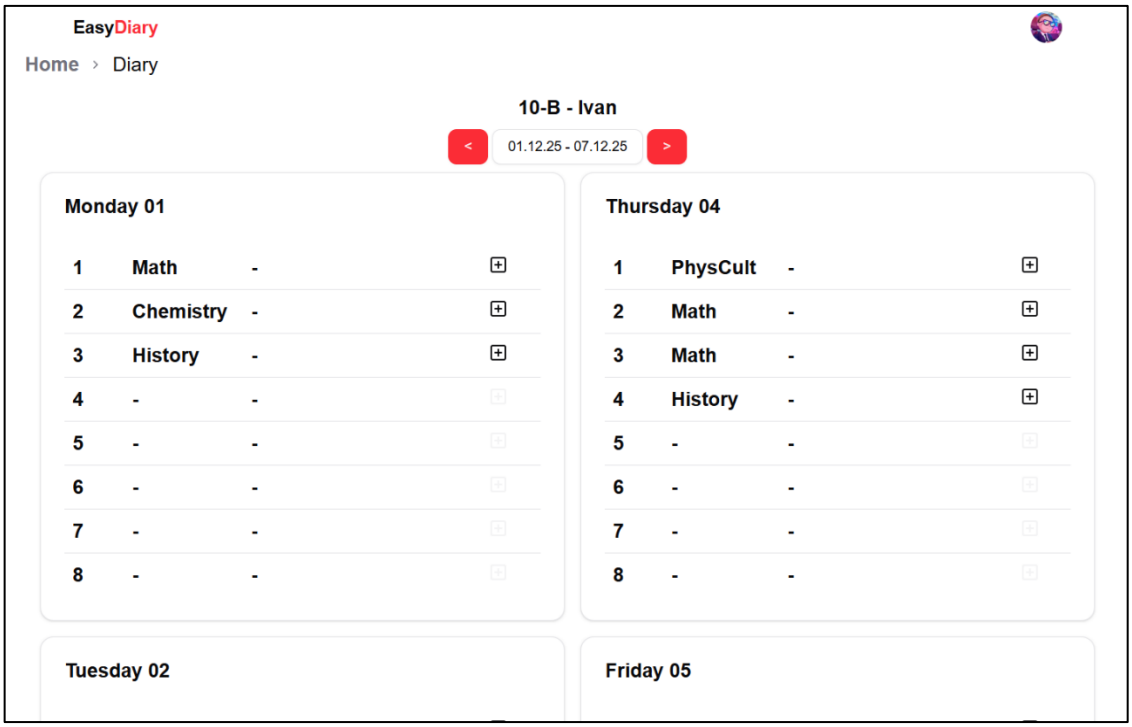



Рисунок 3.9 – Сторінка щоденника

Основний вміст сторінки представлений у вигляді кількох окремих таблиць, по одній для кожного дня тижня. Кожна таблиця має заголовок з назвою і містить список навчальних пар. Для кожного уроку в таблиці відображаються три основні елементи:

– назва предмета – вказує, який урок проводиться на даній парі;
– домашнє завдання – поле, де відображається або короткий опис завдання;
– оцінка – колонка для виставлення чи перегляду оцінки за урок.

У кінці кожного рядка розташована кнопка, що відкриває модальне вікно, у якому викладач може додати або редагувати інформацію про урок: вибрати предмет, вказати домашнє завдання, додати коментар та виставити оцінку. Це дозволяє здійснювати весь процес заповнення щоденника в одному місці, без переходів на інші сторінки.

### 3.6 Сторінка «Контакти»

Сторінка «Контакти» реалізує функціонал внутрішнього чату, який забезпечує комунікацію між учнями та викладачами в межах одного класу (див. рис. 3.10). Інтерфейс сторінки складається і списку контактів, що відображає учасників класу, з якими дозволено листування (учні та викладачі), та області листування у вигляді стрічки повідомлень. Користувач може обрати контакт, після чого завантажується історія чату.

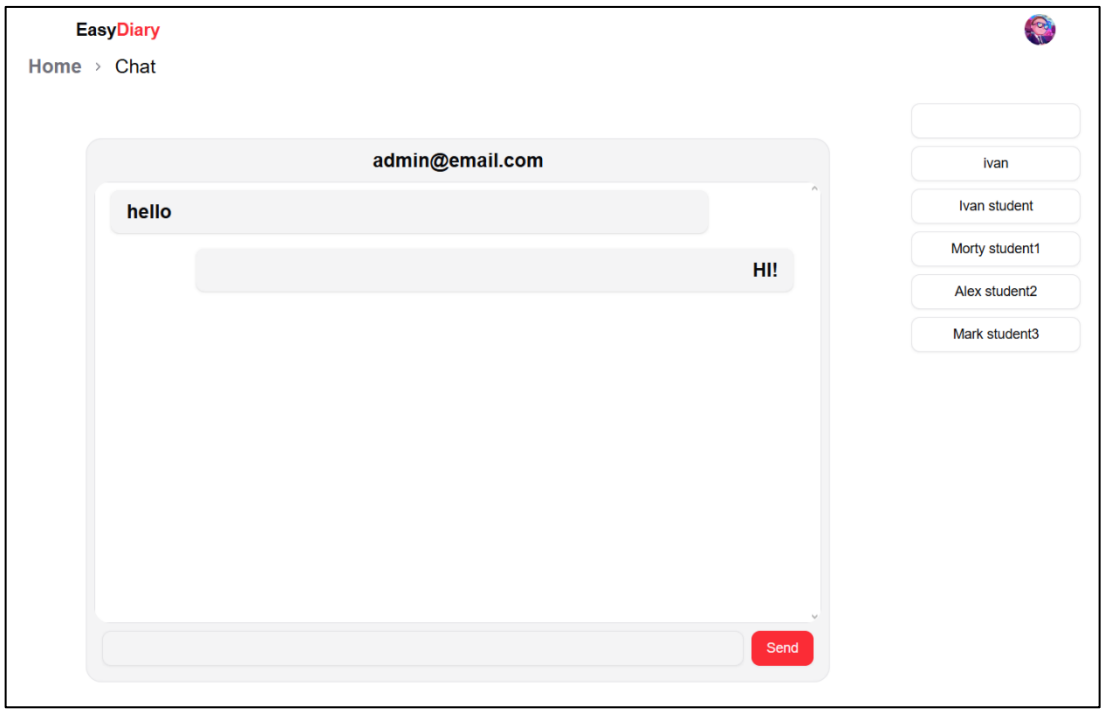



Рисунок 3.10 – Сторінка чату

Повідомлення зберігаються у базі даних і автоматично відображаються в інтерфейсі. Стилізація чату мінімалістична: повідомлення користувача та адресата візуально відрізняються й вирівнюються по різні сторони, що покращує читабельність.

# ВИСНОВКИ

У кваліфікаційній роботі було розроблено вебзастосунок електронного щоденника з використанням технологій React та Next.js. Отриманий застосунок забезпечує комплексну підтримку навчального процесу для різних категорій користувачів: адміністратора, учителя та учня. У процесі проєктування та реалізації було застосовано сучасні підходи до архітектури, зокрема MVC та Dependency Injection, що дозволило забезпечити чіткий поділ відповідальностей, масштабованість та потенціал для повторного використання ядра системи в майбутніх проєктах.

Структура серверної та клієнтської частин включає кастомну модель API поверх можливостей Next.js, а також використання ORM Sequelize для роботи з реляційною базою даних PostgreSQL. Створена модель даних охоплює всі ключові сутності навчального процесу: користувачів, класи, предмети, оцінки, розклади, повідомлення та домашні завдання, що дозволило забезпечити повноцінну функціональність системи.

Особлива увага приділялася зручності та інтуїтивності інтерфейсу. Клієнтська частина реалізує набір окремих сторінок відповідно до ролей користувачів: сторінку розкладу, щоденник з оцінками та завданнями, чат для комунікації між учасниками класу, сторінку контактів і профілю. Це забезпечує логічну структурованість та комфорт взаємодії з системою.

Розроблений електронний щоденник є технологічно сучасним, структурно витриманим і практично орієнтованим рішенням, яке може слугувати основою для подальших освітніх сервісів та систем управління навчальним процесом. Завдяки використаним підходам, система має потенціал для впровадження нових функцій та інтеграцій зі сторонніми сервісами.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Нові знання. URL: https://www.nz.ua (дата звернення: 07.05.2025).
2. Moodle. URL: https://moodle.org/?lang=uk (дата звернення: 07.05.2025).
3. CanopyLab. URL: https://canopylab.com/ (дата звернення: 07.05.2025).
4. Multi-Page Application definition. URL: https://surl.li/crughw (дата звернення: 08.05.2025).
5. Single-Page Application. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 08.05.2025).
6. How People Use Google Search. URL: https://backlinko.com/google-user-behavior (дата звернення: 09.05.2025).
7. Що таке Googlebot і як він працює? URL: https://wedex.com.ua/blog/shho-take-googlebot-i-yak-vin-pratsyuye/ (дата звернення: 09.05.2025).
8. Definition of Server-Side Rendering. URL: https://surl.lt/fgwuwn (дата звернення: 12.05.2025).
9. React. URL: https://react.dev (дата звернення: 13.05.2025).
10. Next-Auth. URL: https://next-auth.js.org (дата звернення: 13.05.2025).
11. MySQL. URL: https://www.mysql.com (дата звернення: 13.05.2025).
12. Sequelize. URL: https://sequelize.org (дата звернення: 13.05.2025).
13. Redux. URL: https://redux.js.org (дата звернення: 13.05.2025).
14. Responsive design. URL: https://surl.li/uxwltr (дата звернення: 14.05.2025).
15. TailwindCSS. URL: https://tailwindcss.com (дата звернення: 14.05.2025).
16. Benefit of using MVC. URL: https://www.geeksforgeeks.org/benefit-of-using-mvc/ (дата звернення: 20.05.2025).
17. Що таке DI-контейнер? URL: https://doc.nette.org/uk/dependency-injection/container (дата звернення: 21.05.2025).