МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ РОЗКЛАДОМ ТА ЕЛЕКТРОННИМ ЖУРНАЛОМ В ЗАКЛАДАХ ОСВІТИ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Степан ПАН

«___»_____ 2025 р.

Керівник старший викладач кафедри ІПЗ _____ Марина ФАЛЕНКОВА

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Пан Степан Миколайович
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інформаційна система управління розкладом та електронним журналом в закладах освіти».

Керівник роботи: Фаленкова Марина Володимірівна старший викладач кафедри ІПЗ.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: інформаційна система для керування розкладом та електронним журналом у закладах освіти; вхідні дані включають списки студентів, вчительського складу, навчальних дисциплін, оцінок, а також розклад занять. Передбачені ролі користувачів: адміністратор, вчитель, учень, батьки.

Очікуваний результат - функціональна система з функцією авторизації, чітким розмежуванням прав доступу, можливостями перегляду та редагування розкладу та оцінок, підтримкою пошуку, фільтрації та аналізу успішності студентів.

4. Перелік питань, що підлягають розробці: аналіз поточного стану справ із управлінням розкладом і електронним журналом в освітніх закладах; огляд типових архітектур інформаційних систем, які використовуються в навчальних установах; проєктування структури бази даних системи; розробка функціональних можливостей для різних ролей користувачів, включаючи адміністратора, викладача, студента та батьків; реалізація механізмів авторизації, управління розкладом, а також обліку й перегляду оцінок; тестування системи та аналіз отриманих результатів.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Марина ФАЛЕНКОВА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Степан ПАН
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «__» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інформаційна система управління розкладом та електронним журналом в закладах освіти

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем щодо інформаційної системи управління розкладом та електронним журналом в закладах освіти	31.01.2025	01.03.2025	
4	Огляд архітектур інформаційних систем для управління розкладом та журналом з метою вибору оптимального підходу;	02.03.2025	01.04.2025	
5	Реалізація архітектури інформаційної системи та аналіз її ефективності;	02.04.2025	15.05.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	17.06.2025	

Керівник роботи _____ Марина ФАЛЕНКОВА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Степан ПАН
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«__» _____ 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Пана Степана Миколайовича

на тему: **“ ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ РОЗКЛАДОМ ТА ЕЛЕКТРОННИМ ЖУРНАЛОМ В ЗАКЛАДАХ ОСВІТИ”**

Актуальність цієї кваліфікаційної роботи полягає в необхідності покращення процесів управління навчальним процесом у закладах освіти шляхом автоматизації розкладу занять та електронного обліку успішності. Впровадження таких рішень дозволяє підвищити ефективність організації освітнього процесу, покращити взаємодію між учнями, вчителями та батьками, а також забезпечити зручний і безпечний доступ до навчальної інформації.

Об’єктом роботи є процес автоматизації управління навчальним процесом у закладах освіти.

Предметом роботи є розробка інформаційної системи для управління розкладом занять і ведення електронного журналу з врахуванням ролей користувачів.

Мета роботи: поглиблення знань і навичок у галузі розробки веб-застосунків, зокрема у використанні сучасних технологій Next.js, NestJS, Prisma та систем аутентифікації, а також вдосконалення навичок проектування та реалізації складних інформаційних систем для освітньої сфери.

Робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної області та наявних рішень, сформульовано завдання. Другий розділ присвячений огляду інструментів і методів розробки. У третьому розділі описано розробку інформаційної системи, її структуру та результати моделювання. Четвертий розділ містить деталі програмної реалізації, опис користувацьких інтерфейсів та результати тестування.

Отримані результати підтверджують ефективність розробленої системи для автоматизації управління навчальним процесом, а також її зручність і безпеку для

різних категорій користувачів. Рекомендовано впровадження системи у навчальних закладах для підвищення якості організації навчального процесу.

Кількість сторінок: 66, таблиць: 0, рисунків: 41, додатків: 1, використаних джерел: 27.

Ключові слова: інформаційна система, управління розкладом, електронний журнал, аутентифікація, ролі користувачів, веб-застосунок, Next.js, NestJS, Prisma.

ABSTRACT

for the qualification thesis
by the student of group 401 of Petro Mohyla Black Sea National University

Pan Stepan Mykolaiovych

on the topic: "INFORMATION SYSTEM FOR SCHEDULE MANAGEMENT AND ELECTRONIC GRADEBOOK IN EDUCATIONAL INSTITUTIONS"

The relevance of this qualification thesis lies in the need to improve the management processes of the educational process in institutions through the automation of class scheduling and electronic performance tracking. The implementation of such solutions allows for greater efficiency in educational organization, enhances interaction between students, teachers, and parents, and ensures convenient and secure access to academic information.

The object of the study is the automation process of educational process management in educational institutions.

The subject of the study is the development of an information system for managing class schedules and maintaining an electronic gradebook, considering user roles.

The aim of the thesis is to deepen knowledge and skills in web application development, particularly in the use of modern technologies such as Next.js, NestJS, Prisma, and authentication systems, as well as to enhance competencies in designing and implementing complex information systems for the educational sector.

The work consists of four chapters. The first chapter analyzes the subject area and existing solutions and formulates the task. The second chapter reviews the tools and methods used for development. The third chapter describes the development of the information system, its structure, and the modeling results. The fourth chapter presents the details of the software implementation, user interface description, and testing results.

The obtained results confirm the effectiveness of the developed system for automating the management of the educational process, as well as its convenience and

security for various categories of users. The implementation of the system in educational institutions is recommended to improve the quality of educational process organization.

Number of pages: 66, tables: 0, figures: 41, appendices: 1, sources used: 27.

Keywords: information system, schedule management, electronic gradebook, authentication, user roles, web application, Next.js, NestJS, Prisma.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

VS Code – Visual Studio Code;

ORM – Object-Relational Mapping;

БД – База даних;

СУБД – Система управління базами даних;

MVC – Model-view-controller.

ВСТУП

В умовах сьогодення, позначених цифровою трансформацією освіти, автоматизація навчальних процесів, зокрема - керування розкладом та ведення електронного журналу, стає надзвичайно важливою. Наявні рішення, як-от Google Classroom [1], Moodle [2], "Нові знання" [3] та інші, лише частково задовольняють потреби закладів освіти. Однак, значна кількість з них характеризується обмеженим функціоналом, відсутністю зручного адміністрування або надмірною складністю для впровадження у невеликих школах. Це зумовлює потребу в розробці більш адаптивних, гнучких та зручних для використання систем, що забезпечують централізоване управління навчальною інформацією.

Актуальність обраної теми підкреслюється необхідністю підвищення ефективності організації навчального процесу, забезпечення прозорості оцінювання, а також гарантування постійного доступу до навчальної інформації для всіх зацікавлених сторін: учнів, вчителів, батьків та адміністрації. Розробка сучасного вебдодатку дає змогу вирішити низку організаційних та комунікаційних викликів у межах освітнього закладу.

Представлена дипломна робота базується на результатах переддипломної практики, в процесі якої було спроектовано та реалізовано ключові модулі системи. Вона також є логічним розвитком ідей, сформованих в процесі вивчення технологій розробки вебдодатків та побудови інформаційних систем для потреб освітньої сфери.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1 Опис предметної сфери

Освітній процес у сучасних навчальних закладах — це складна взаємодія, яка включає організацію навчального графіку, оцінювання успішності, зв'язок між учасниками (учні, вчителі, батьки, керівництво) та інші аспекти, що потребують координації та автоматизації. Раніше ці процеси здійснювались вручну або за допомогою локального програмного забезпечення, що призводило до великої кількості паперової роботи, помилок, дублювання інформації та ускладнень із доступом до даних.

З розвитком інформаційних технологій постала потреба у створенні інтегрованих інформаційних систем, які б автоматизували основні процеси навчального закладу. Управління розкладом, ведення електронних журналів, організація зв'язку між всіма учасниками освітнього процесу — все це складові цифрової трансформації освіти.

В умовах прискореного переходу до онлайн та дистанційного навчання важливо розробляти системи, які підтримують віддалену роботу, зберігаючи при цьому зручність традиційного шкільного процесу. Особливо це стосується онлайн-шкіл, які не мають власної IT-інфраструктури і потребують простих, надійних і зручних у використанні рішень.

Відповідно до поставленого завдання було розроблено вебсайт онлайн-школи, що надає широкий спектр функцій для різних користувачів. Основні можливості системи включають:

– перегляд переліку всіх дисциплін - користувачі можуть ознайомитися з повним переліком навчальних предметів, що викладаються в закладі, що сприяє кращому плануванню навчального процесу та вибору курсів;

– перегляд списку батьків - система надає інформацію та контакти батьків учнів, що забезпечує зручність у встановленні зв’язку між викладачами, адміністрацією та сім’ями учнів;

– перегляд списку вчителів - користувачі можуть отримати інформацію про викладачів, їх спеціалізацію та контактні дані, що покращує прозорість і довіру в освітньому середовищі;

– перегляд списку запланованих іспитів - актуальний графік іспитів доступний усім користувачам, що дозволяє краще готуватися до контрольних заходів;

– персональні профілі вчителів та учнів - кожен користувач має особистий кабінет з персональною інформацією, до якої можуть звертатися інші учасники системи. Користувачі можуть редагувати свої дані, що гарантує актуальність інформації;

– перегляд розкладу занять - розклад представлено у зрозумілому вигляді для всіх категорій користувачів, що полегшує навігацію в графіку занять;

– електронний журнал - вчителі можуть відмічати присутність, виставляти оцінки та залишати коментарі щодо успішності учнів, забезпечуючи прозорість та оперативність в обліку навчальних досягнень;

– адміністративне управління - адміністратор системи володіє розширеними правами для керування користувачами (учнями, батьками, вчителями), дисциплінами, розкладом занять, даними електронного журналу та іншими параметрами системи, що дозволяє ефективно підтримувати роботу інформаційної системи.

Важливою особливістю розробленої системи є її орієнтація на сучасні потреби: мобільність, інтуїтивний інтерфейс, масштабованість і безпека. Система розроблена для забезпечення надійного захисту даних, контролю доступу на основі ролей, а також можливості подальшого розширення функціоналу.

На сьогоднішній день інформаційні системи в освіті є не просто інструментами для ведення обліку, а комплексними платформами, що інтегрують

освітній процес, покращують якість навчання та керують даними. Вони сприяють зменшенню навантаження на викладачів та адміністрацію, автоматизуючи рутинні операції, підвищенню прозорості оцінювання та контролю навчальних результатів, забезпеченню швидкого та зручного доступу до навчальних матеріалів і даних, створенню умов для ефективної комунікації та співпраці всіх учасників освітнього процесу.

Кожна категорія користувачів має власні потреби. Учні потребують швидкого доступу до розкладу, оцінок та інформації про завдання. Вчителі потребують інструментів для швидкого внесення оцінок, відміток про відвідуваність, планування уроків. Батьки бажають контролювати успішність та відвідуваність дітей, отримувати повідомлення про важливі події. Адміністрація відповідає за управління всіма процесами, забезпечує контроль та підтримку роботи системи. Розроблена система враховує ці ролі, надаючи кожному відповідний рівень доступу та функціональності.

Вибір веб-технологій для розробки зумовлено їх універсальністю та доступністю. Вебзастосунок не потребує встановлення на локальні пристрої, що робить його зручним для користувачів з різних пристроїв - від комп'ютерів до смартфонів. Крім того, це дозволяє централізовано оновлювати систему, забезпечуючи її актуальність та безпеку.

1.2 Огляд та аналіз наявних аналогів

Для створення інформаційної системи управління розкладом та електронним журналом було проведено аналіз доступних рішень у цій сфері. На сьогодні ринок пропонує безліч сервісів, які частково виконують подібні функції. Деякі з них надають вчителям можливість виставляти оцінки та фіксувати відвідуваність, інші дозволяють організовувати дистанційне навчання, завантажувати навчальні матеріали, проводити тестування та спілкуватися з учнями. Водночас, ці рішення часто виявляються занадто складними, обмеженими у функціональності або

неоптимізованими для потреб середніх та невеликих навчальних закладів, особливо для онлайн-шкіл.

Основні аналоги, що були розглянуті:

– Google Classroom - безкоштовний сервіс від Google, широко використовуваний в освітніх установах. Він дозволяє створювати віртуальні класи, додавати учнів, створювати та оцінювати завдання, залишати коментарі та організовувати навчальний процес. Проте, він не має повноцінного функціоналу електронного журналу, не передбачає можливості управління розкладом занять, а також відсутня гнучка система адміністрування користувачів за ролями (наприклад, відсутній окремий доступ для батьків) [1].

Рисунок 1.1 – Логотип Google Classroom [2]

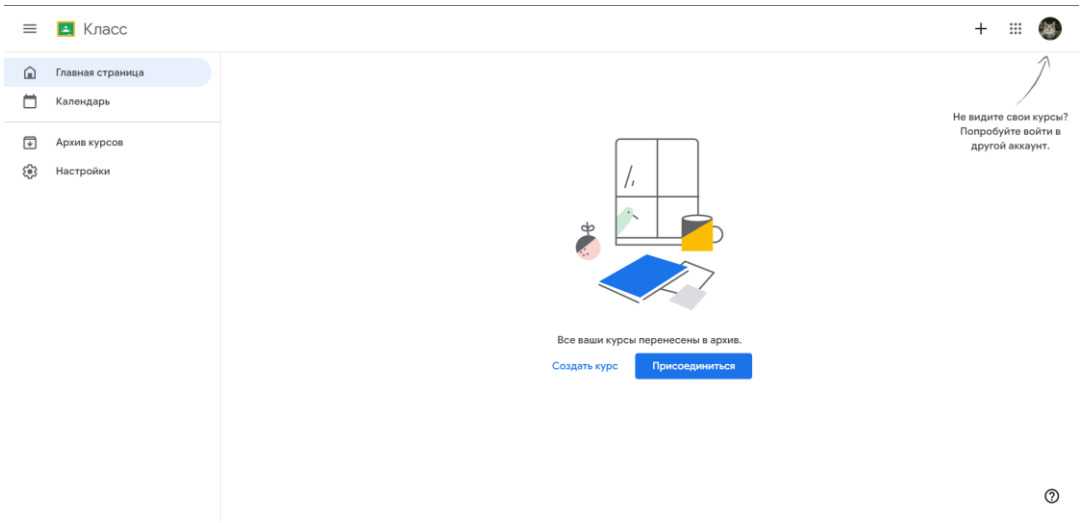

Рисунок 1.2 – Головна сторінка Google Classroom

– Moodle - потужна платформа з відкритим кодом, яка підтримує широкий спектр функцій для організації онлайн-навчання: створення курсів, завдань, тестів, форумів, оцінювання, аналітики та інше. Хоча ця система надзвичайно гнучка та налаштовується, її використання часто вимагає значної технічної підготовки. Крім того, Moodle не має зручного модуля для управління шкільним розкладом та не підходить для швидкого впровадження в невеликих онлайн-школах [3].

Рисунок 1.3 – Логотип Moodle [4]

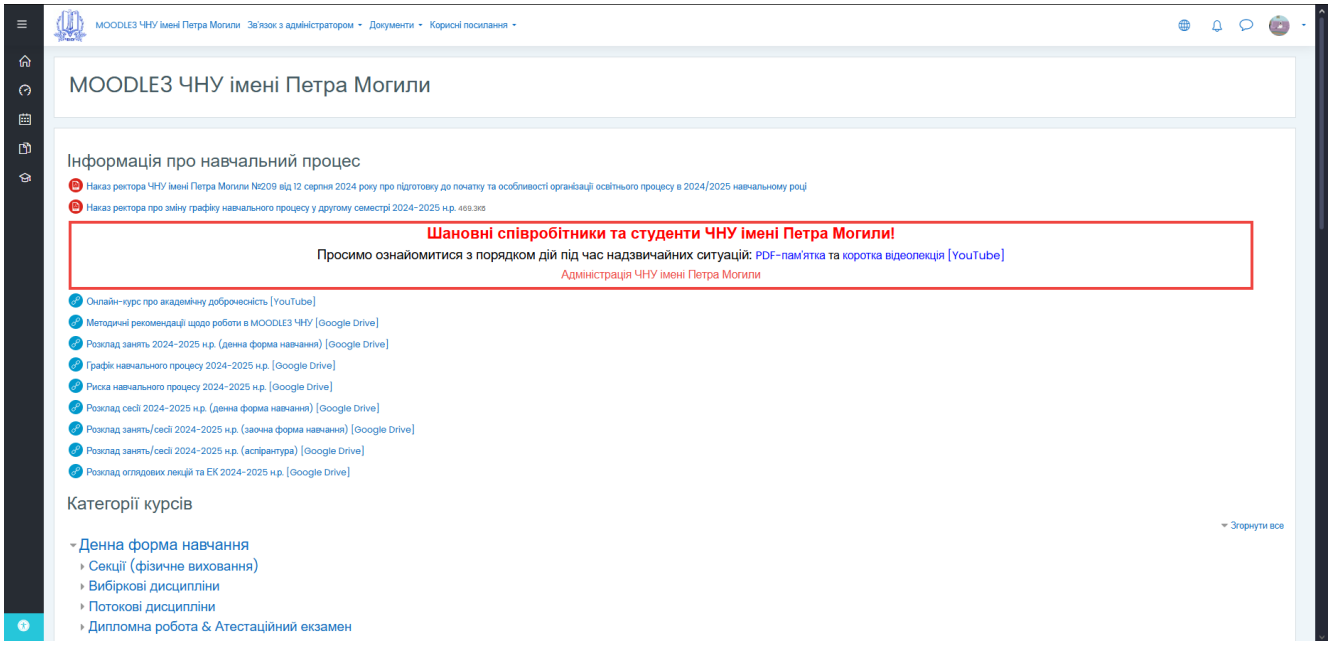

Рисунок 1.4 – Головна сторінка Moodle

– «Нові знання» - українська платформа, орієнтована на середню освіту. Вона дозволяє вести електронні журнали, розклад занять, надсилати повідомлення, формувати звіти для адміністрації та батьків. Платформа інтегрується з

державними вимогами до звітності. Однак її використання платне, що може бути критичним для невеликих закладів. Крім того, залежність від сторонньої компанії створює ризики втрати контролю над збереженими даними [5].

Рисунок 1.5 – Логотип «Нові знання» [6]

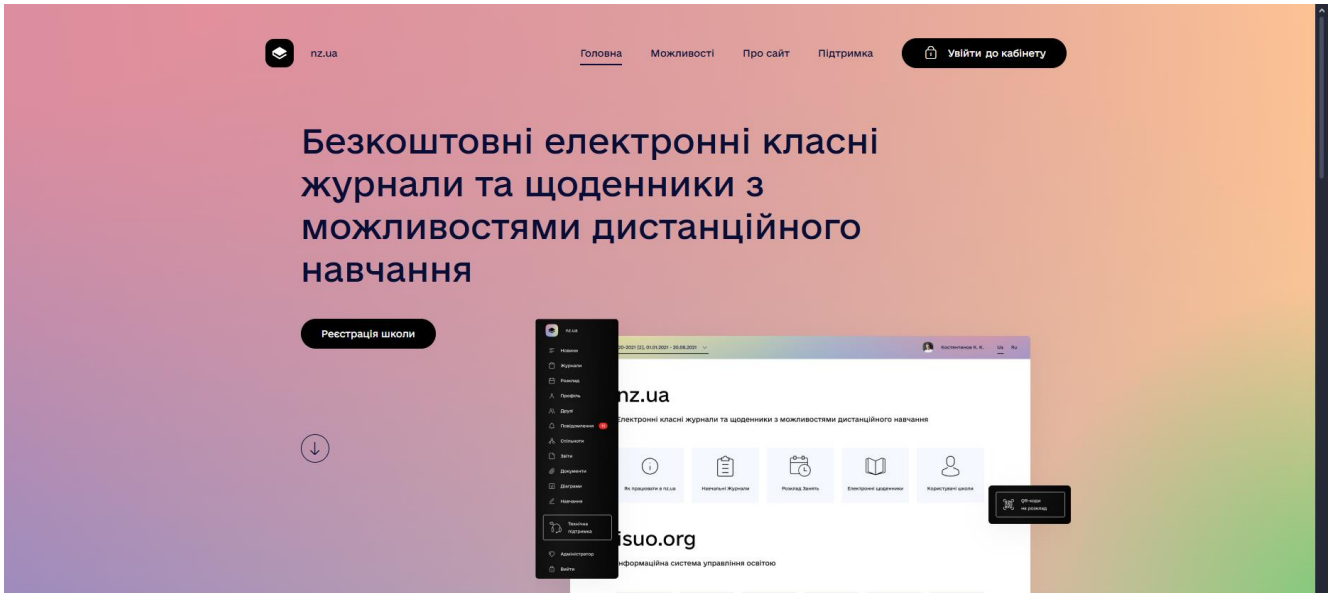

Рисунок 1.6 – Головна сторінка «Нові Знання»

– ДіСО (Державна інформаційна система освіти) - централізована система, розроблена для збору статистичної інформації та обліку даних у сфері освіти. Вона дозволяє реєструвати учнів, формувати звіти та подавати офіційну інформацію до

міністерства. Проте вона не є навчальною платформою в звичному розумінні: відсутні функції, необхідні для організації навчального процесу, такі як оцінювання, ведення журналу чи робота з розкладом занять [7].

Рисунок 1.7 – Логотип ДіСО [8]

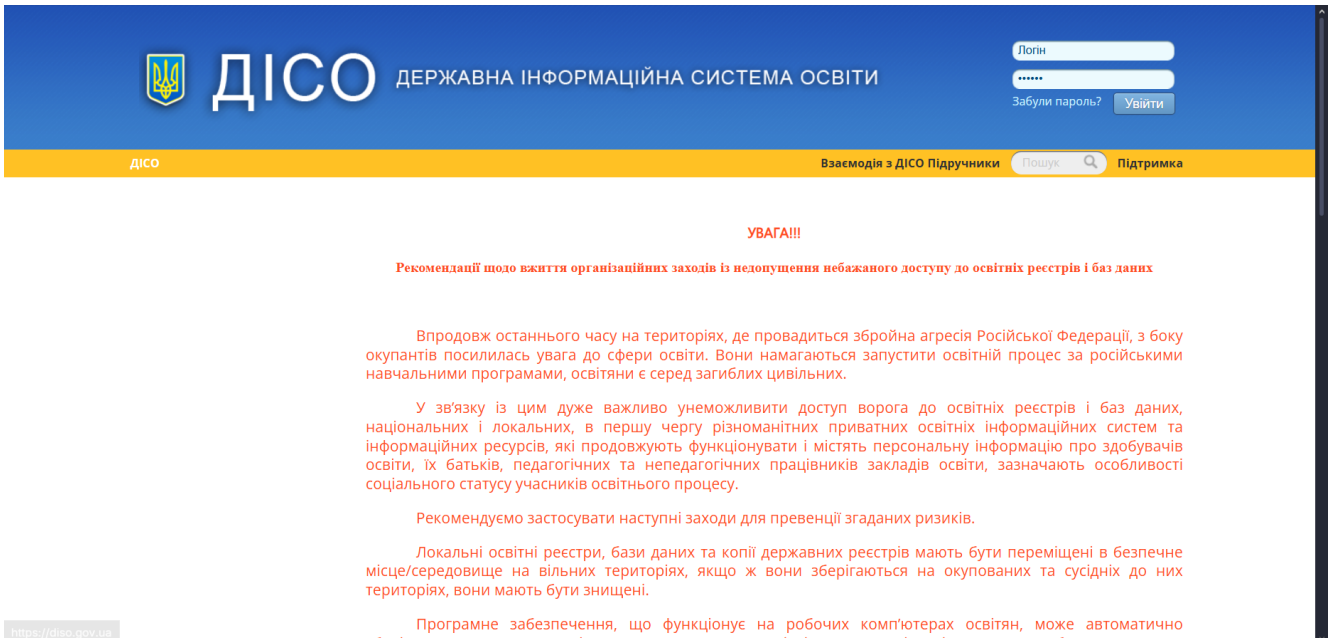

Рисунок 1.8 – Головна сторінка ДіСО

– АІКОМ – комерційна українська система, розроблена спеціально для загальноосвітніх шкіл. Вона забезпечує функціонал для електронного документообігу, зокрема, ведення електронного журналу, відстеження відвідуваності, автоматичне складання розкладу занять, а також передбачає наявність модулів для формування різноманітної звітності та ефективної

комунікації. Система відрізняється високим рівнем персоналізації та гнучким налаштуванням ролей доступу для різних користувачів. Однак, доступ до повної функціональності системи обмежений умовами платної ліцензії, що може стати певним викликом для фінансово обмежених навчальних закладів [9].

Рисунок 1.9 – Логотип AIKOM[10]

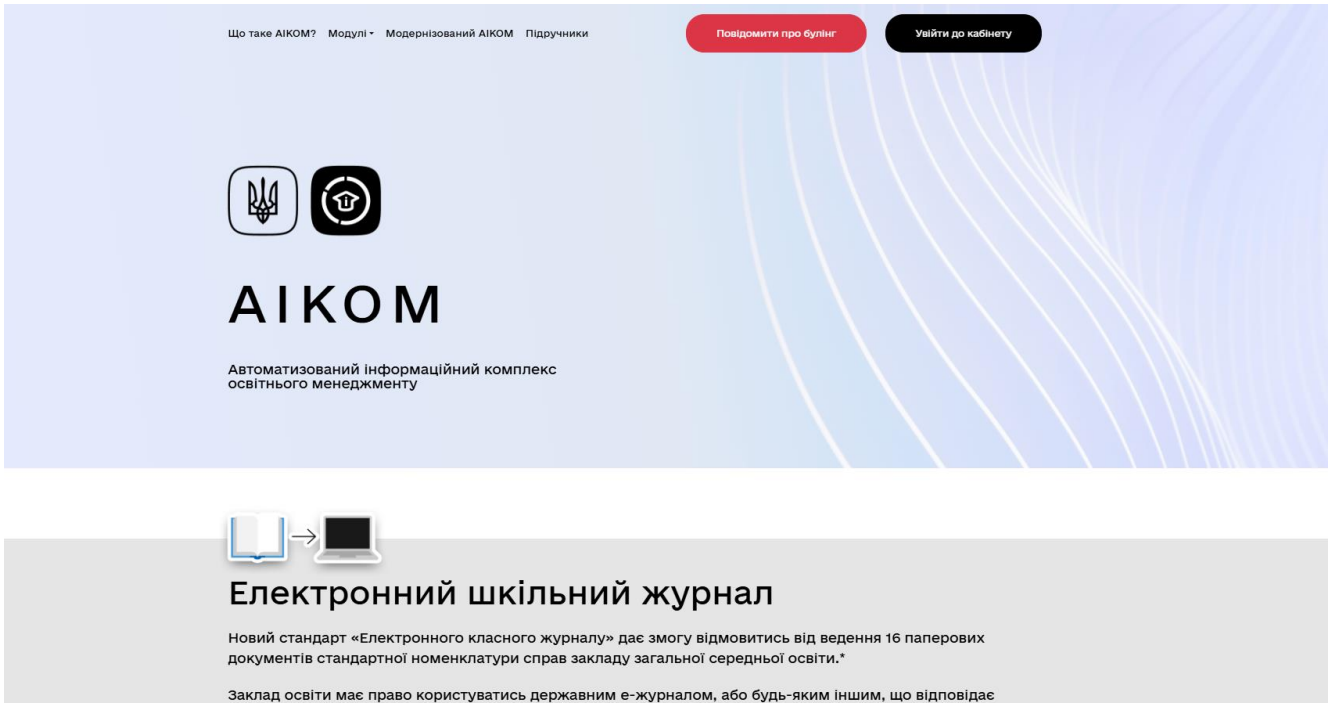

Рисунок 1.10 – Головна сторінка AIKOM

Аналіз аналогів показав, що більшість наявних систем мають обмеження: або не підтримують повного циклу освітнього процесу (розклад, журнал, профілі), або характеризуються складною структурою, що ускладнює їх використання в онлайн-

школах. Часто функціонал розподілений між різними платформами, що змушує користувачів використовувати кілька сервісів одночасно, що негативно впливає на зручність та ефективність.

Враховуючи ці недоліки, було прийнято рішення розробити власну інформаційну систему, яка поєднувала б функціонал розкладу занять, електронного журналу, а також управління профілями користувачів. Вона повинна бути простою у використанні, зручною для адміністрування та адаптованою до потреб сучасної онлайн-школи.

1.3 Постановка задачі

У сучасному контексті цифрової трансформації освіти виникає нагальна необхідність автоматизації управління навчальними процесами. Важливим елементом цього є ефективна організація доступу до розкладу занять та електронного журналу, що забезпечує прозорість оцінювання, сприяє зручній комунікації між усіма учасниками освітнього процесу – учнями, вчителями, батьками та адміністрацією, а також знижує навантаження на адміністративний персонал.

Дослідження наявних інформаційних систем показало, що багато з них мають обмежені можливості, складні в налаштуванні або не адаптовані до особливостей онлайн-освіти, що вимагає розробки власного, адаптованого рішення.

Завданням даної роботи є розробка вебзастосунку для керування розкладом занять та ведення електронного журналу, який забезпечить доступність, зручність та ефективність навчального процесу для всіх користувачів освітнього закладу.

Об'єктом дослідження є процес створення інформаційної системи для освітніх закладів, а предметом - методи, інструменти та технології, що використовуються для розробки вебзастосунку з відповідною архітектурою та функціоналом. Для досягнення поставленої мети передбачається вирішення таких задач:

– створити функціональність для перегляду розкладу занять та ведення електронного журналу, що є ключовим компонентом інформаційної системи, яка забезпечує доступ до освітнього процесу;

– розробити архітектуру вебзастосунку та його ключові структурні елементи, що сприятиме стабільній та масштабованій роботі системи;

– впровадити механізми керування користувачами та адміністрування системи, враховуючи їхні ролі (учень, викладач, батько, адміністратор), для забезпечення гнучкого доступу до функціоналу.

Висновки щодо першого розділу:

У першому розділі увагу було зосереджено на дослідженні предметної області, пов'язаної з автоматизацією навчального процесу, а також було проаналізовано наявні програмні рішення для управління розкладом та електронними журналами. В ході аналізу виявилося, що існуючі системи мають певні обмеження, що проявляються у вигляді недостатнього функціоналу або складності використання, що негативно впливає на їхню ефективність в онлайн-школах. Також розглянуто сильні та слабкі сторони таких платформ, як Google Classroom, Moodle, "Нові знання", ДіСО, AIKOM.

На основі проведеного аналізу було зроблено висновок про необхідність комплексного рішення, яке б забезпечувало одночасне управління розкладом, електронним журналом, профілями користувачів та адміністративними функціями в рамках однієї системи для оптимальної організації навчального процесу. З метою вирішення виявлених проблем запропоновано розробити власний вебзастосунок, який інтегруватиме ці функціональні можливості та матиме інтуїтивно зрозумілий інтерфейс для всіх користувачів. У цьому ж розділі сформульовано ключові вимоги до майбутньої системи, які стануть фундаментом для подальшої розробки вебзастосунку.

2 АНАЛІЗ ІНСТРУМЕНТІВ РОЗРОБКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1 Методи для вирішення поставленої задачі

Для забезпечення успішного управління навчальним процесом та автоматизації важливих операцій освітніх установ, необхідно інтегрувати передові підходи до розробки інформаційних систем. Ключовим є створення легкого доступу до розкладів та електронних журналів для студентів, викладачів та батьків, плюс функціонал адміністрування самої системи. З огляду на ці потреби, обрані підходи до розробки мають гарантувати надійність, здатність до масштабування та інтуїтивно зрозумілий інтерфейс.

В ході роботи були застосовані такі методи:

– об'єктно-орієнтоване програмування (ООП) використовувалося для структурування взаємодії між елементами системи, такими як користувачі, графік, оцінки та інші. Завдяки цьому вдалося створити архітектуру, яка є масштабованою та легкою в обслуговуванні [11];

– метод модульного проєктування застосовано під час розробки інтерфейсу та бекенду. Це забезпечило легкість у тестуванні, внесенні змін та повторному використанні окремих складових;

– методи архітектури клієнт-серверної взаємодії стали основою для побудови системи. Фронтенд (Next.js) взаємодіє з бекендом (NestJS) через REST API, дотримуючись класичної моделі. Це дає змогу розділити задачі та гарантувати масштабованість;

– метод нормалізації бази даних дозволив організувати зберігання інформації про користувачів, розклад, оцінки та інші сутності. Проведено аналіз даних і нормалізацію таблиць, що дозволило уникнути надмірності та забезпечити цілісність інформації;

– методи тестування застосовувалися для перевірки функціональності розроблених модулів. Використовувалися модульні тести, а також ручне тестування, яке проводилося як на клієнтській, так і на серверній стороні.

Завдяки впровадженню згаданих методів вдалося розробити інформаційну систему, котра відзначається надійністю та зручністю у використанні, цілком відповідаючи актуальним потребам освітніх закладів. Використання подібних підходів при розробці сприяє підвищенню ефективності навчання, а також покращує взаємодію між всіма, хто бере участь в освітньому процесі.

2.2 Технології розробки системи

Для створення інформаційної системи управління розкладом та електронним журналом були застосовані передові технології, що дозволили розробити повнофункціональний вебзастосунок з чітко розділеними клієнтською та серверною складовими. Такий підхід гарантує комфорт використання та стабільність функціонування системи.

Створено два ключових блоки. Перший — призначений для взаємодії з користувачами: школярі, вчителі, батьки мають можливість керувати своїми обліковими записами, переглядати розклад занять, отримувати доступ до оцінок та іншої суттєвої інформації. Другий — серверна частина, що відповідає за збереження та обробку даних, аутентифікацію користувачів та налагодження взаємодії між усіма компонентами системи.

Розробка відбувалася у середовищі, яке забезпечує швидке створення та тестування вебзастосунків, а також спрощує роботу з базою даних. Це дозволило спроектувати систему, що легко масштабується (має потенціал для розширення в майбутньому) та зручна для підтримки й оновлення.

2.2.1 Visual Studio Code

Visual Studio Code (VS Code) — безкоштовний, невеликий, проте надзвичайно продуктивний текстовий редактор з відкритим кодом, розроблений Microsoft. Він забезпечує підтримку для багатьох мов програмування, включає в себе інтегровані інструменти налагодження, сумісність з Git та широку бібліотеку розширень [12].

Основні можливості VS Code:

– підтримка великої кількості мов програмування (JavaScript, Python, C++, C#, Java, PHP, HTML, CSS тощо);

– вбудований Git – можливість комітити, пушити, мержити код прямо у редакторі;

– налагодження (Debugger) – проприєтарний дебаггер для багатьох мов;

– intelliSense – розуме автозавершення коду на основі синтаксису;

– теми та іконки – інтерфейс кастомізації;

– термінал – вбудований командний рядок (PowerShell, bash, cmd);

– розширення (Extensions) – великий ринок додатків для розширення функціоналу;

– крос-платформенність – працює на Windows, macOS та Linux [12].

Тепер щодо переваг VS Code:

– швидкий і легкий (на відміну від повноцінних IDE, таких як Visual Studio);

– гнучкий – можна адаптувати під будь-які потреби завдяки розширенням;

– популярний серед розробників – один із найуживаніших редакторів (згідно з опитуванням Stack Overflow);

– інтеграція з хмарними сервісами (Git Hub, Azure, Docker тощо).

Недоліки:

– деякі функції потребують розширень (на відміну від повноцінних IDE);

– менш підходить для великих проектів у порівнянні з JetBrains IDE або Visual Studio.

Visual Studio Code (VS Code) підходить для дуже широкого кола користувачів, наприклад:

– фронтенд-розробники (JavaScript, TypeScript, React, Vue і т.д);
– бекенд-розробники (Python, Node.js, PHP, C# і т.д);
– DevOps та адміністратори (робота з конфігами, Docker, YAML і т.д);
– студенти та початківці (простий у використанні, але потужний) [12].

Рисунок 2.1 – Логотип Visual Studio Code [13]

2.2.2 Next.js

Next.js – це широко вживаний фреймворк React для розробки швидких та інтерактивних вебзастосувань. Він пропонує різноманітні функціональні можливості, включаючи рендеринг на стороні сервера (SSR), генерацію статичних сторінок (SSG), маршрутизацію API та оптимізацію зображень [14]. Основні можливості Next.js:

а) серверний рендеринг (SSR):

1) сторінки генеруються на сервері, що покращує SEO та швидкість завантаження;

2) використовується getServerSideProps [14].

б) статична генерація сторінок (SSG):

1) HTML генерується під час збірки, що ідеально для блогів, документації тощо;

2) використовується getStaticProps та getStaticPaths [14].

в) гібридний підхід (ISR – Incremental Static Regeneration): можливість оновлювати статичні сторінки без повної перезбірки [14].

г) вбудований API-шар: створення API без додаткового серверного коду (файли в /pages/api/) [14].

д) оптимізація зображень: автоматичне форматування, lazy loading через компонент <Image> [14].

е) маршрутизація на основі файлової системи: створення роутів через структуру папок у /pages [14].

ж) підтримка TypeScript та CSS-модулів:

1) вбудована інтеграція TypeScript;

2) підтримка CSS/Sass, Tailwind CSS, styled-jsx [14].

з) покращена продуктивність: автоматичний code splitting, prefetching для швидких переходів [14].

Рисунок 2.2 – Логотип Next.js [15]

2.2.3 Tailwind

Tailwind CSS – це сучасний фреймворк для оформлення вебзастосунків, що дозволяє швидко та легко стилізувати інтерфейс. На відміну від традиційних CSS-фреймворків (де наявні готові компоненти, такі як кнопки чи таблиці), Tailwind надає набір готових класів, які користувач може комбінувати безпосередньо в

HTML або React - компонентах, створюючи унікальний дизайн на власний смак [16].

Tailwind надзвичайно зручний завдяки:

– швидкому редагуванню дизайну прямо в коді компонента;

– створенню чистого, зрозумілого та єдиного стилю на всьому веб-сайті;

– наявності мобільної адаптивності (легко створити сайт, що буде добре відображатися як на смартфоні, так і на комп’ютері);

– генерації лише тих стилів, які фактично використовуються – це забезпечує швидке завантаження сайту [17].

У проєкті було використано Tailwind для розробки дизайну всіх сторінок: панелей користувачів, таблиць з оцінками, списків учнів та вчителів, форм для введення даних тощо. Завдяки Tailwind, інтерфейс виглядає сучасним, акуратним та зручним для користувачів.

Такий підхід дозволив мені швидко розробляти дизайн, не витрачаючи час на створення окремих CSS-файлів.

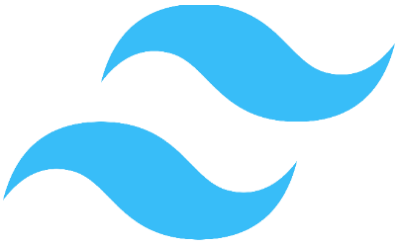

Рисунок 2.3 – Логотип Tailwind [18]

2.2.4 NestJS

NestJS — це фреймворк для створення бекенду, розроблений на основі Node.js, з використанням TypeScript. Він допомагає розробляти ту частину програми, що відповідає на запити, взаємодіє з базами даних та виконує логіку програми [19].

Основна риса NestJS полягає у його структурованому підході до розробки. Код легко впорядковувати завдяки модулям, контролерам і сервісам, що забезпечує чіткість, спрощує процес розробки та полегшує супровід. Приміром, можливо створити окремі модулі для графіку, користувачів, оцінок, а потім об'єднати їх в єдину систему.

Що ще корисного в NestJS:

– підтримка авторизації та аутентифікації — можна легко додати вхід у систему і розмежувати права доступу;

– легка інтеграція з базами даних (через Prisma або інші ORM);

– зручне підключення додаткових інструментів (наприклад, для відправки електронних листів, створення API, логування);

– хороша підтримка для REST API і GraphQL — тобто я можу створити зручний інтерфейс для взаємодії між фронтендом і бекендом.

У даному проєкті NestJS я використав для створення серверної частини, яка відповідає за:

– реєстрацію та авторизацію користувачів (учнів, вчителів, батьків, адміністратора);

– обробку запитів від сайту (наприклад, запити на перегляд розкладу чи оцінок);

– роботу з БД через Prisma;

– управління всією бізнес-логікою системи [19].

Завдяки NestJS бекенд вийшов структурованим, безпечним і готовим до розширення в майбутньому. Це дозволяє легко додавати нові функції або оновлювати існуючі.

Рисунок 2.4 – Логотип NestJS [20]

2.2.5 Prisma

Prisma — це сучасний ORM (Object-Relational Mapping) інструмент, що спрощує розробникам взаємодію з базою даних у проєктах на JavaScript та TypeScript. Замість складання громіздких SQL-запитів, Prisma дозволяє працювати з базою даних через зрозумілі методи, що нагадують звичайний код [21].

Основна мета Prisma — пришвидшити, спростити та мінімізувати помилки при роботі з даними. Для цього використовується Prisma Schema — спеціальний файл, де я описую структуру бази даних: таблиці, поля, типи даних та взаємозв'язки між ними. Потім Prisma автоматично генерує з цього коду зручні функції для роботи з даними [21].

Наприклад, для отримання списку всіх учнів або для додавання нового розкладу, мені не потрібно писати SQL – Prisma надає готові методи, які легко читати та використовувати. Крім цього, Prisma має такі корисні можливості:

– міграції бази даних – автоматично застосовує зміни в структурі бази (додає нові поля, таблиці тощо);

– автоматична генерація типів – якщо я використовую TypeScript, Prisma підказує, які поля є в таблицях, і допомагає уникнути помилок при написанні коду;

– зручне тестування – можна легко створювати тестові дані і перевіряти роботу системи;

– підтримка різних баз даних – Prisma працює з популярними СУБД, наприклад PostgreSQL, MySQL, SQLite [21].

У моєму проєкті Prisma стала в нагоді для ефективної організації даних про студентів, викладачів, батьків, навчальні предмети, оцінки та розклад занять. За допомогою цього інструменту я без зусиль налаштував базу даних, оперативно додавав та редагував інформацію, а також гарантував збереження даних на високому рівні.

Застосування Prisma суттєво прискорило процес розробки та забезпечило читабельність коду, що полегшує його подальшу підтримку.

Рисунок 2.5 – Логотип Prisma [22]

2.2.6 Axios

Axios – це широко вживана JavaScript-бібліотека, що використовується для виконання HTTP-запитів у браузерах або середовищі Node.js. Axios побудований на промісах (Promises) та надає легкий і зрозумілий API для комунікації з RESTful-сервісами [23].

Ключові особливості Axios:

– підтримка усіх основних HTTP-методів: GET, POST, PUT, DELETE та інші;

– асинхронність, забезпечена застосуванням промісів;

– автоматичне перетворення даних у форматі JSON;

– гнучке налаштування заголовків запитів (headers);

– підтримка перехоплювачів (interceptors) для запитів та відповідей;

– зручне оброблення виникнення помилок;

– функціонал роботи з таймаутами, а також можливість скасування запитів (cancel token) [23].

Надання запитів з налаштуванням withCredentials, що використовується для авторизації.

Рисунок 2.6 – Логотип Axios [24]

Висновки щодо другого розділу:

Внаслідок огляду інструментарію та методів розробки було відібрано сукупність стратегій та технік, які найкраще задовольняють поставлені завдання. Застосування об'єктно-орієнтованого підходу, модульного структурування, клієнт-серверної архітектури та методів нормалізації баз даних посприяло створенню стабільної, масштабованої та зручної у використанні системи. Використання принципів UI/UX дизайну та тестування покращило якість інтерфейсу та стабільність функціонування веб-додатку.

Вибір актуальних технологій, таких як Visual Studio Code для розробки, Next.js для фронтенду, Tailwind CSS для стилізації, NestJS для бекенду та Prisma для роботи з базами даних, став ключем до оперативного та ефективного створення функціональної інформаційної системи для керування розкладом та електронним журналом. Комбінація цих інструментів гарантує майбутній розвиток, підтримку та розширення проекту з урахуванням потреб користувачів та освітнього закладу, що постійно змінюються.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних

Для ефективної роботи інформаційної системи, що керує розкладом занять та електронним журналом, було виокремлено ключові типи вхідних даних. Вони відображають весь спектр інформації, потрібної для функціонування системи та задоволення потреб всіх користувачів — учнів, вчителів, батьків та адміністрації.

Дані про учнів є фундаментом системи. Вони включають унікальний ідентифікатор, повне ім’я (ПІБ), фото для візуалізації профілю, а також контактну інформацію (номер телефону, електронна пошта, домашня адреса). Окрім цього, кожен учень прив'язаний до конкретного класу та має логін або унікальний номер для автентифікації в системі. Важливо зберігати дані про відвідування занять, що дає можливість відслідковувати присутність учнів на уроках. Також реєструються результати навчання у вигляді оцінок, що систематизуються за предметами та періодами.

Інформація про вчителів містить унікальний ідентифікатор, ПІБ, фотографію та контактну інформацію, необхідну для внутрішньої комунікації. В системі також зберігаються відомості про предмети, які викладає кожен вчитель, та класи, в яких проводяться заняття. Викладачі мають розширені можливості, як-от виставлення оцінок учням, позначення їх присутності та ведення електронного журналу, тому для них також фіксуються відповідні записи про успішність та відвідування.

Дані про класи включають унікальний номер або назву класу, список учнів, що входять до нього, та викладачів, закріплених за цим класом. Ця інформація служить основою для формування розкладу занять, організації навчального процесу та управління взаємодією між учнями і викладачами.

Розклад занять — це структурований набір даних, що містить інформацію про дні тижня, час проведення уроків, відповідні предмети, вчителів та класи. Цей

розклад служить основою для створення індивідуальних розкладів для учнів і викладачів, забезпечуючи організацію навчального процесу та оптимізацію часу.

Оцінки та відвідування представлені записами, що містять інформацію про успішність кожного учня з конкретного предмета, дату отримання оцінки, а також відмітки про відвідування уроків. Це дозволяє не тільки контролювати навчальні досягнення, а й оперативно реагувати на пропуски та інші відхилення.

Джерелом цих вхідних даних є самі користувачі системи – адміністрація та вчителі, які вводять та оновлюють інформацію через зручний веб-інтерфейс, а також автоматизовані процеси, що забезпечують своєчасне оновлення та підтримання актуальності даних.

Всі дані зберігаються у реляційній базі даних, організованій відповідно до сучасних стандартів та керованій через API, розроблений на сервері з використанням NestJS та ORM Prisma. Для покращення продуктивності та зручності користування передбачено функції пошуку, фільтрації та пагінації, що забезпечують швидкий доступ до необхідної інформації, навіть при значних обсягах даних.

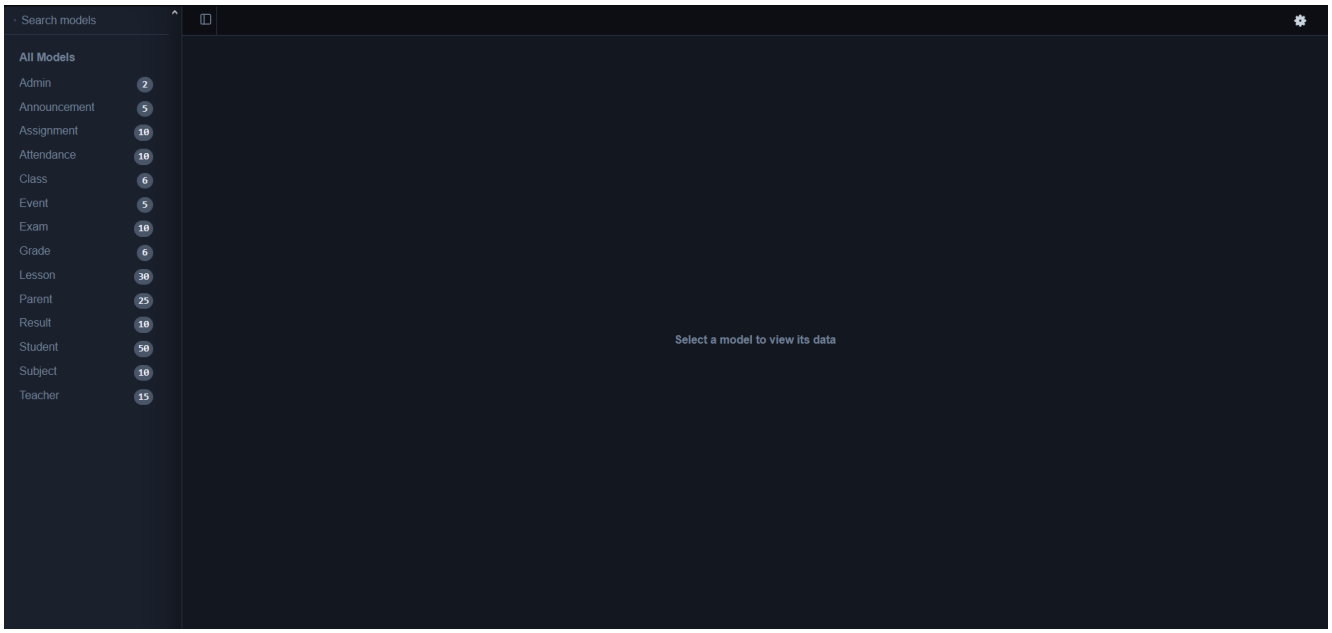

Рисунок 3.1 – База даних

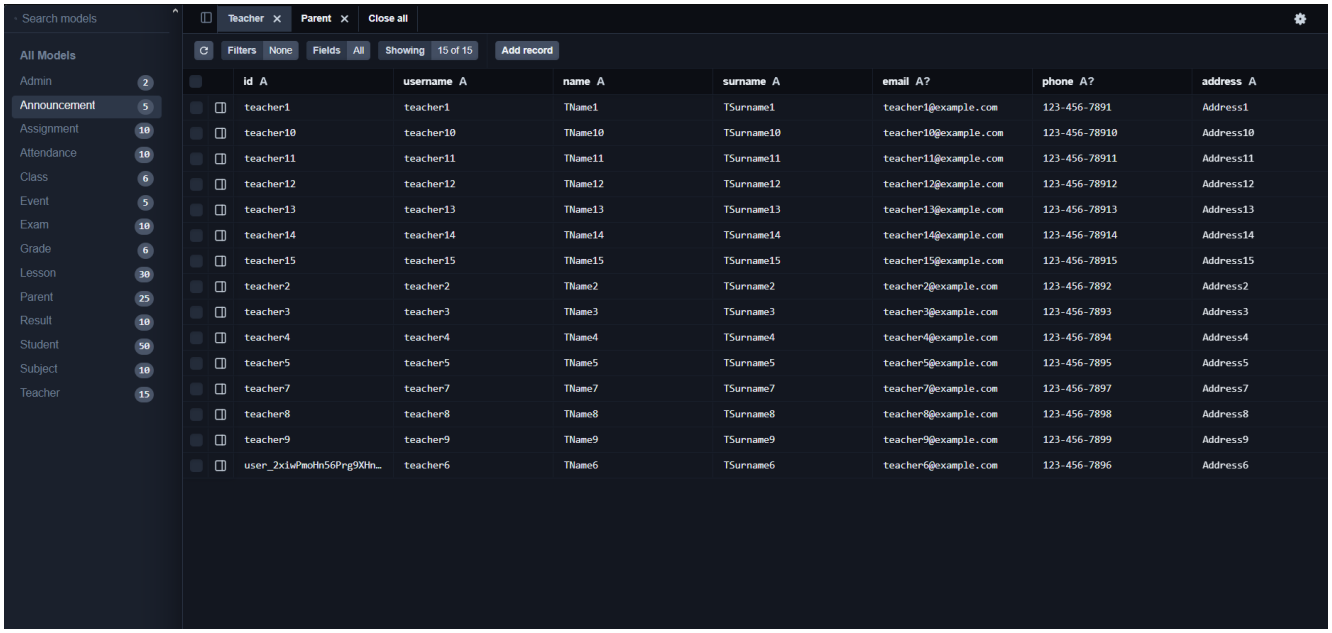

Рисунок 3.2 – База даних вчителів

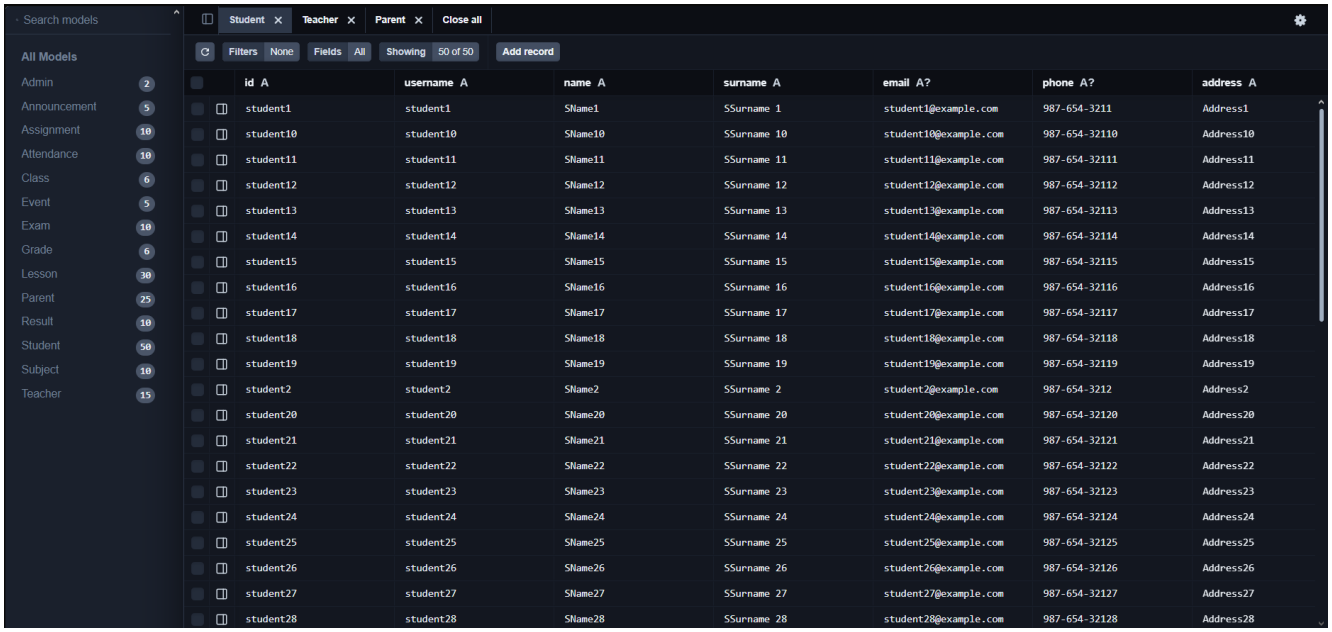

Рисунок 3.3 – База даних учнів

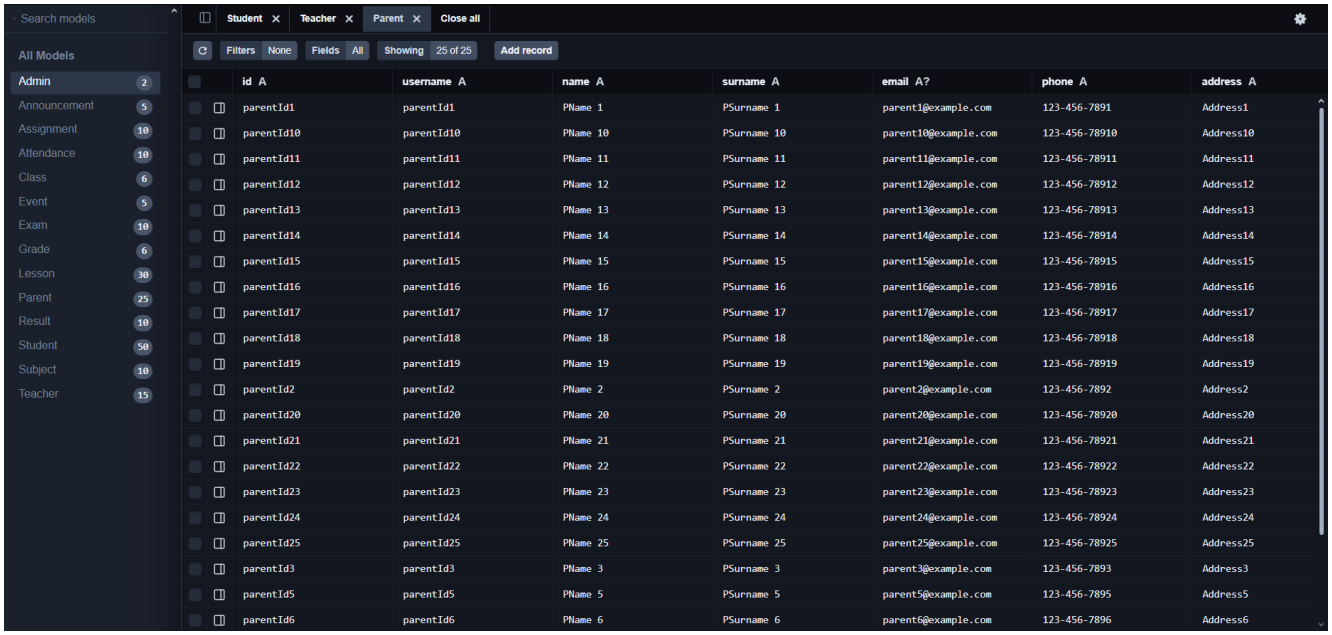

Рисунок 3.4 – База даних батьків

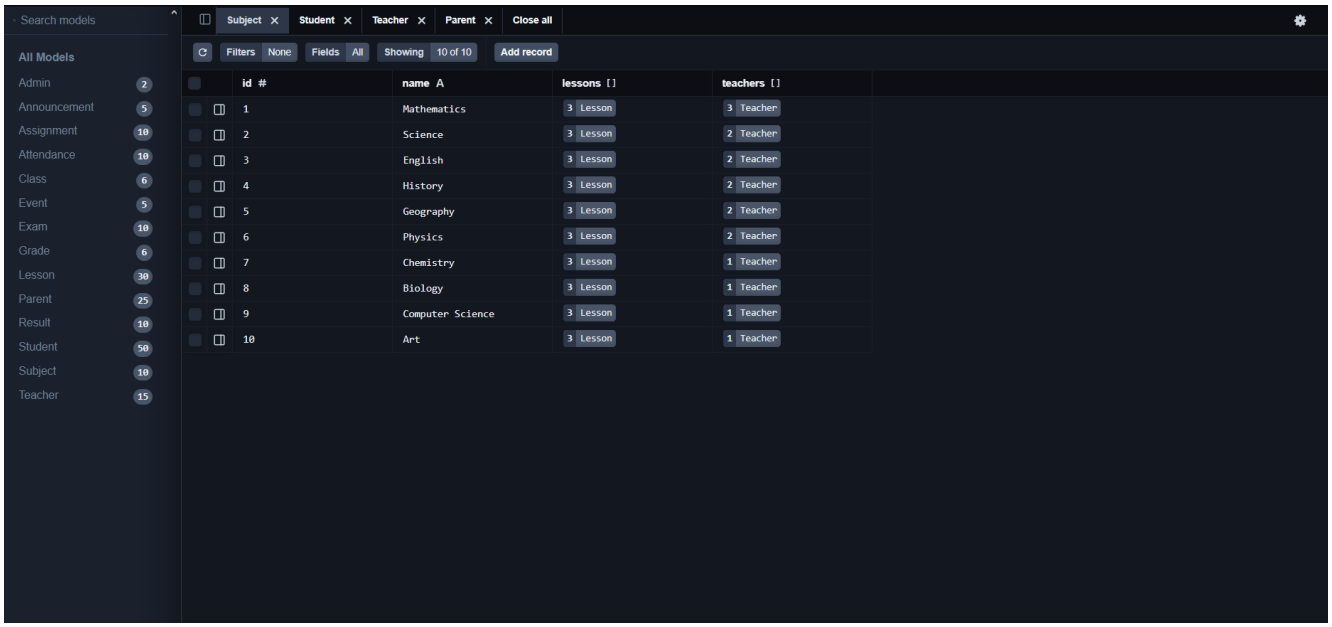

Рисунок 3.5 – База даних дисциплін

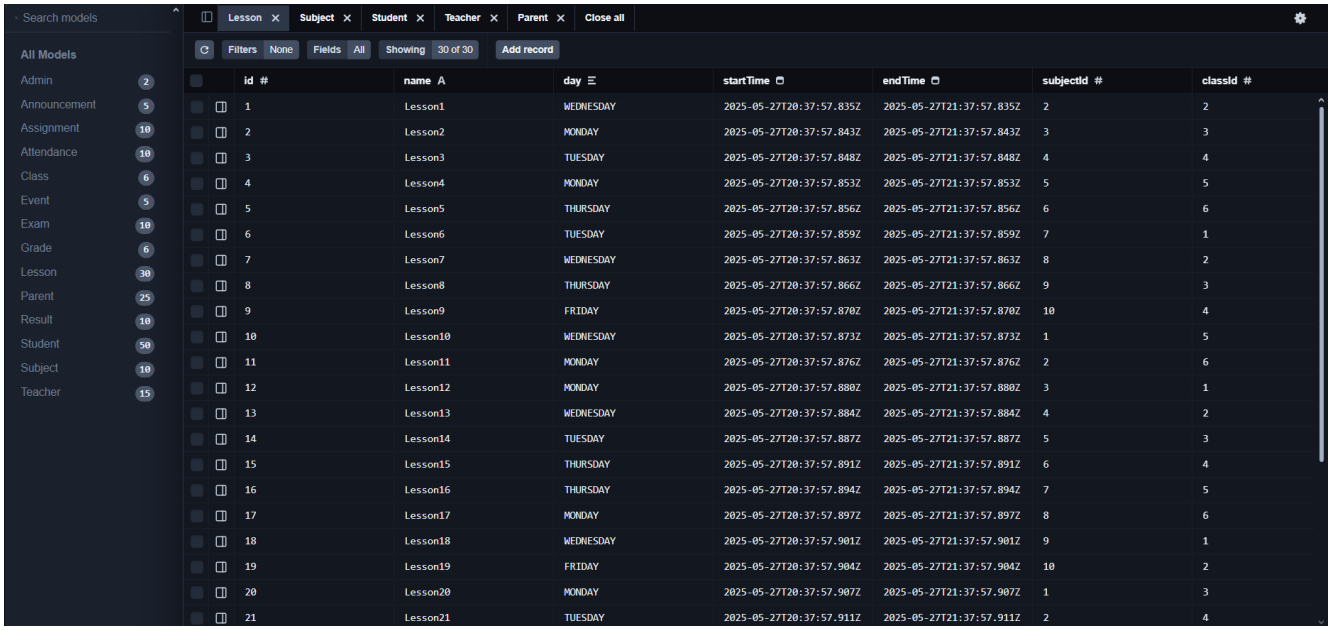

Рисунок 3.6 – База даних уроків

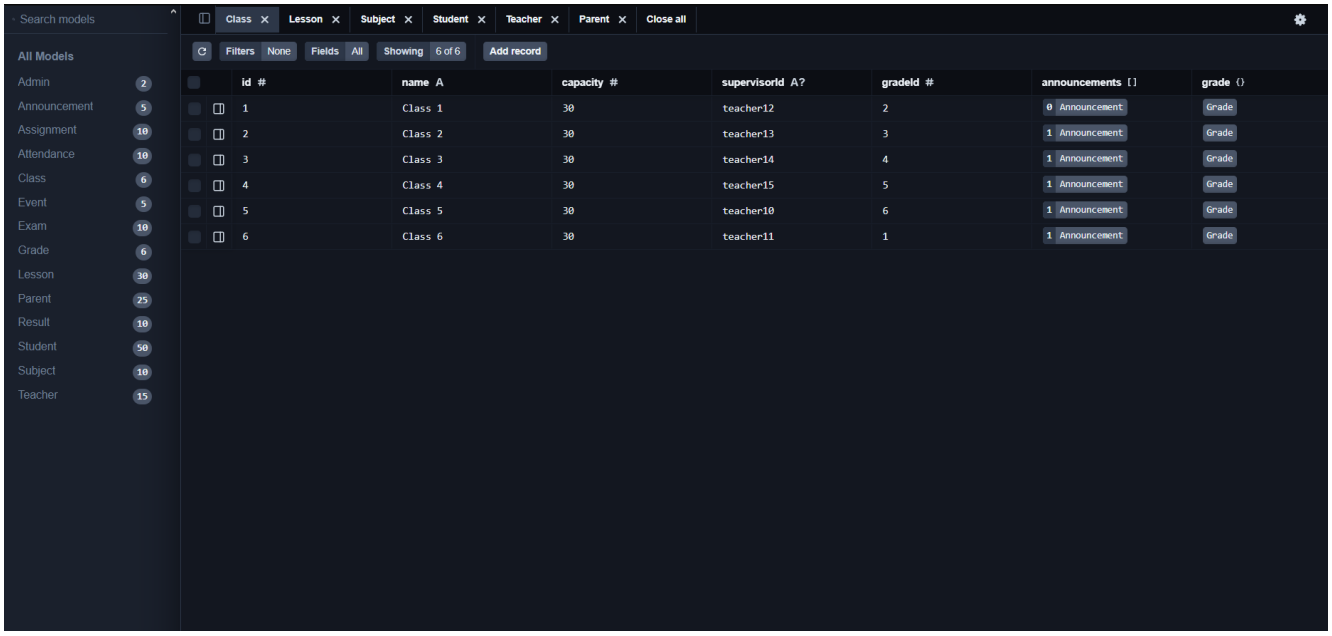

Рисунок 3.7 – База даних класів

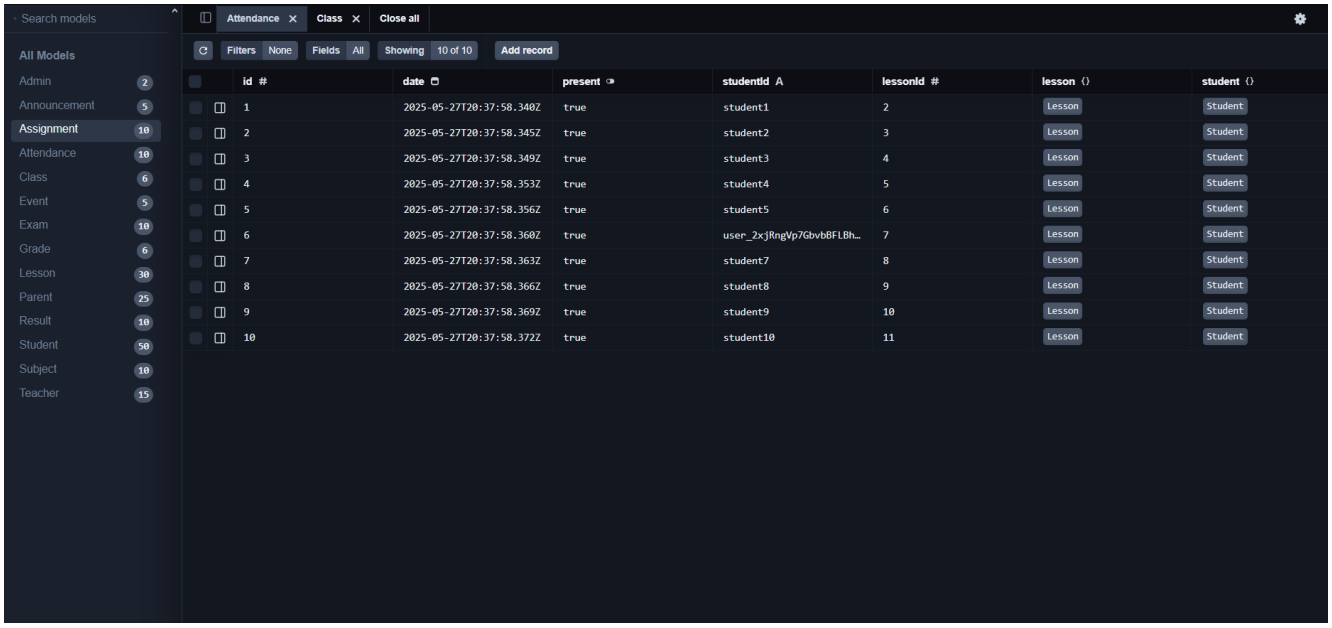

Рисунок 3.8 – База даних відвідувань

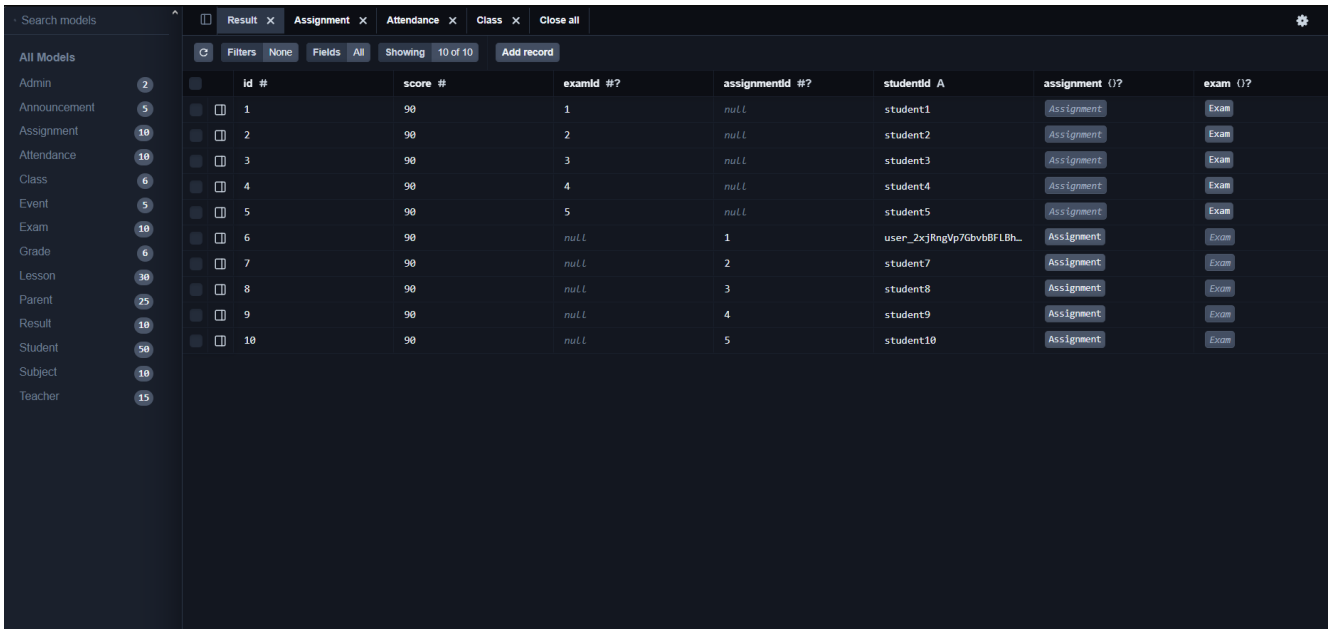

Рисунок 3.9 – База даних результатів

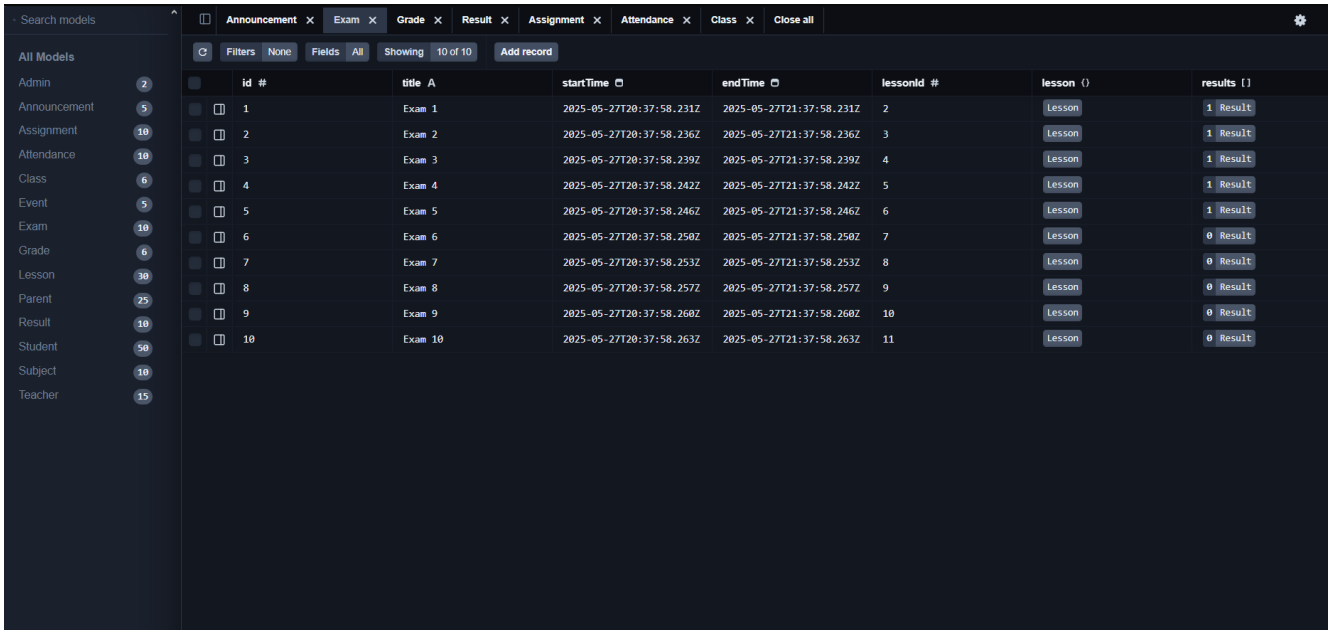

Рисунок 3.10 – База даних екзаменів

Таким чином, вхідні дані системи створюють повну, структуровану та актуальну інформаційну базу, необхідну для автоматизації освітнього процесу, моніторингу відвідуваності та контролю успішності учнів, що значно підвищує ефективність роботи навчального закладу.

3.2 Архітектура застосунку

Розроблена інформаційна система спирається на класичну багаторівневу клієнт-серверну архітектуру, що гарантує продуктивну обробку даних, масштабованість та зручність у підтримці. Архітектура проєкту вміщує три основні шари: клієнтський (frontend), серверний (backend) та рівень бази даних.

Клієнтська частина розроблена з використанням фреймворку Next.js, що забезпечує рендеринг на стороні сервера та підтримку динамічних сторінок. Для стилізації інтерфейсу застосовано бібліотеку Tailwind CSS у поєднанні з компонентами shadcn/ui, що створило адаптивний, сучасний та комфортний інтерфейс для користувача. Основні функції, реалізовані на клієнті, включають перегляд списків користувачів (учнів, вчителів, батьків), фільтрацію, пошук,

пагінацію, перегляд розкладу, оцінок, журналу, а також доступ до персональних кабінетів.

Серверна частина сконструйована з використанням NestJS — масштабованого бекенд-фреймворку, що базується на TypeScript і підтримує модульну архітектуру. NestJS забезпечує обробку HTTP-запитів, управління бізнес-логікою та реалізацію REST API. У системі реалізовані контролери для обробки запитів, сервіси для бізнес-логіки, DTO для валідації вхідних даних, мідлвари та гвардії для аутентифікації та авторизації. Обмін даними між frontend і backend здійснюється за допомогою HTTP-запитів у форматі JSON.

Як базу даних використовується PostgreSQL, що забезпечує надійне зберігання реляційних даних. Для взаємодії з базою даних обрано ORM Prisma, яка дозволяє зручно моделювати структуру таблиць у файлі schema.prisma, генерує відповідні типи TypeScript та забезпечує безпечну роботу з базою даних. У системі реалізовані всі необхідні сутності: Student, Teacher, Parent, Class, Subject, Lesson, Attendance, Result та інші, між якими налаштовані зв’язки один-до-багатьох або багато-до-багатьох згідно з логікою навчального процесу.

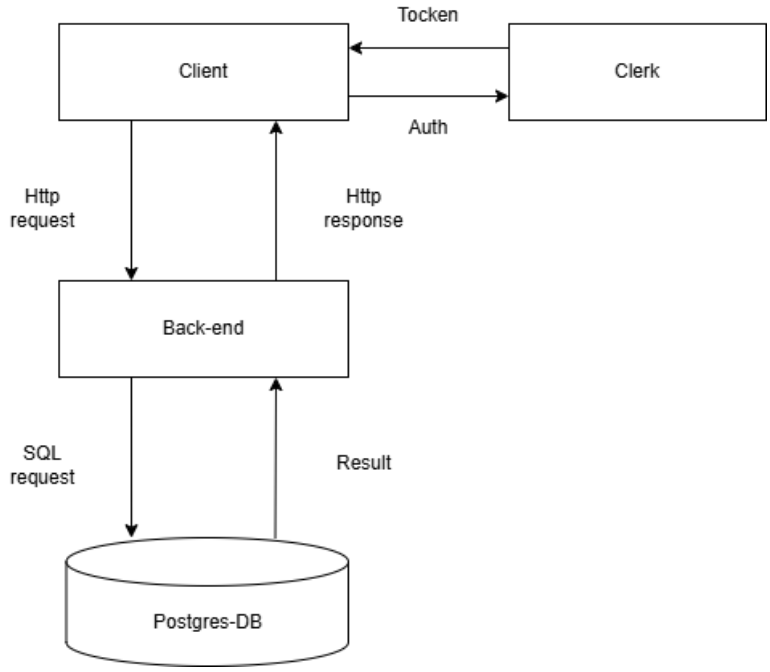

Рисунок 3.11 – Архітектура застосунку

3.3 MVC-архітектура

MVC-архітектура (Model–View–Controller) — це модель організації програмного забезпечення, яка поділяє застосунок на три взаємозалежні, проте окремі складові: модель, вигляд та контролер. Це структурування дозволяє полегшити процес розробки, супроводження та розширення програмних систем шляхом розмежування логіки обробки даних, інтерфейсу користувача та взаємодії з ним. [25]

3.3.1 Model (Модель)

Модель відповідальна за зберігання, обробку та керування даними. Вона інкапсулює бізнес-логіку застосунку, структурує об'єкти системи та контролює взаємодію з базою даних:

– визначає сутності, які використовуються в системі, такі як (наприклад, Teacher, Student, Class, Subject);

– реалізується у вигляді схем бази даних або ORM-моделей. У проєкті для реалізації обрано Prisma;

– виконує запити до бази даних: створення, читання, оновлення та видалення (CRUD);

– забезпечує цілісність та узгодженість даних у системі. [26]

3.3.2 View (Представлення)

Відповідальність представлення полягає у візуалізації інформації, тобто в тому, як користувач бачить дані. Воно створює платформу для взаємодії з системою:

– відображає дані, отримані з моделі, у зрозумілій та зручній формі;

– не містить бізнес-логіки, займається лише обробкою подій інтерфейсу;

– у цьому проєкті реалізація здійснюється за допомогою Next.js та стилізації за допомогою Tailwind CSS і компонентів Shadcn/UI. [26]

3.3.3 Controller (Контролер)

Контролер виступає зв'язковою ланкою між моделлю та інтерфейсом. Він відповідає за обробку отриманих запитів, управління логікою взаємодії та запуск процесів оновлення як моделі, так і представлення.

– отримує інформацію від інтерфейсу (наприклад, з полів форм, з URL-адрес);

– викликає необхідні методи моделі для опрацювання отриманих даних;

– повертає результати, отримані від моделі, у форматі JSON, який потім використовується для оновлення представлення;

– в рамках даного проєкту контролери втілені в NestJS у вигляді окремих класів з набором методів, що відповідають різним типам HTTP-запитів (GET, POST, DELETE та інші). [26]

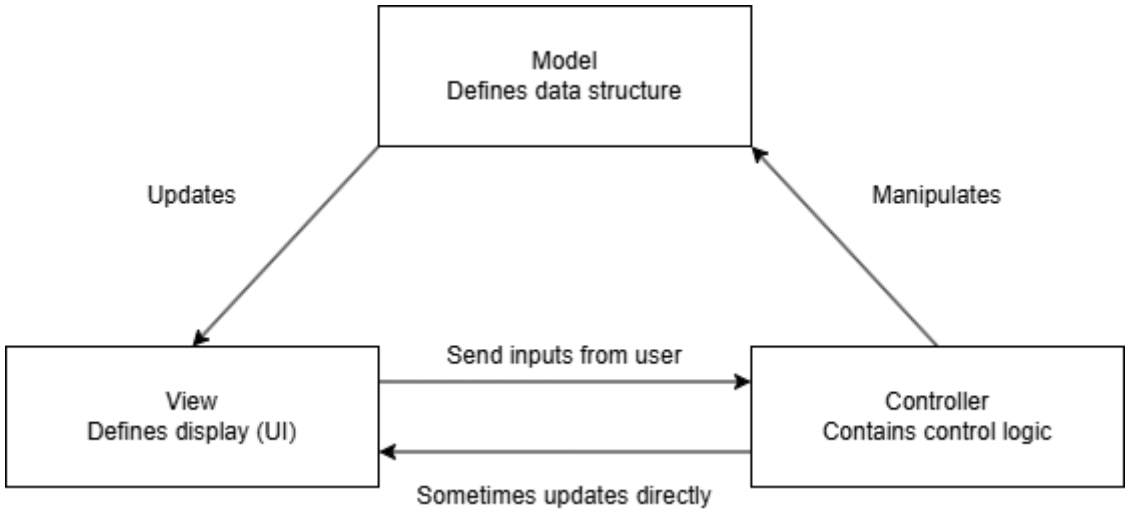

Рисунок 3.12 – MVC-архітектура

3.4 Проєктування/моделювання

У процесі створення інформаційної системи для управління розкладом та електронним журналом особливе значення мав етап ретельного проектування архітектури системи та моделювання основних даних. Це дозволило не тільки окреслити ключові складові програмного продукту, але й визначити принципи взаємодії між ними, логіку обробки даних та узгодити функціональні потреби клієнтської (front-end) та серверної (back-end) частин системи.

На першому етапі була розроблена загальна архітектурна схема, що ґрунтується на класичній клієнт-серверній моделі з чітким розподілом обов'язків. Для побудови front-end був обраний Next.js – сучасний React-фреймворк, який забезпечує швидке відтворення, підтримку server-side rendering та оптимізацію продуктивності інтерфейсу користувача. Серверна частина реалізована на базі NestJS, що дає можливість створювати модульний, масштабований та добре організований back-end з використанням TypeScript. Обмін даними між front-end та back-end здійснюється через REST API, розроблений з урахуванням стандартів безпеки та продуктивності.

Важливим аспектом проектування стала розробка логічної моделі даних, реалізованої у вигляді Prisma-схеми. Ця модель відображає основні сутності інформаційної системи, такі як Teacher (вчитель), Student (учень), Parent (батько), Class (клас), Lesson (урок), Subject (предмет), Attendance (відвідування), Result (оцінки) та інші. Для кожної сутності було визначено атрибути, які забезпечують необхідну деталізацію, а також встановлено зв’язки між ними, які відповідають реальним освітнім процесам. Серед ключових відносин варто відзначити:

– вчитель може викладати кілька предметів, що реалізує зв’язок "один-до-багатьох" між Teacher та Subject;

– учень належить до одного класу, проте бере участь у багатьох уроках, що відображає складну структуру навчальної діяльності;

– один батько може бути пов’язаний з кількома учнями, що відповідає сімейним зв’язкам;

– електронний журнал (Result) фіксує оцінки і результати по предметах і уроках для кожного учня, що забезпечує прозорість і повноту навчальної інформації.

Паралельно з моделюванням даних було проведено детальне проектування інтерфейсів користувача з урахуванням ролей та сценаріїв використання системи. Кожен тип користувача отримав персоналізований набір функцій:

– учні можуть переглядати свій розклад, результати навчання та особисту інформацію;

– вчителі мають змогу вести електронний журнал, виставляти оцінки, відмічати відвідуваність та переглядати інформацію про свої класи та учнів;

– батьки отримують доступ до даних про успішність та відвідуваність своїх дітей;

– адміністратори системи мають повний спектр інструментів для управління даними, включаючи створення, редагування та видалення інформації.

Завдяки такому ретельному підходу до проектування та моделювання вдалося оптимізувати розробку системи, чітко розмежувати зони відповідальності між компонентами, а також забезпечити масштабованість та легкість підтримки проекту в майбутньому. Такий підхід заклав міцний фундамент для подальшої реалізації, тестування та впровадження інформаційної системи, що відповідає вимогам сучасної освіти та сприяє підвищенню ефективності навчального процесу.

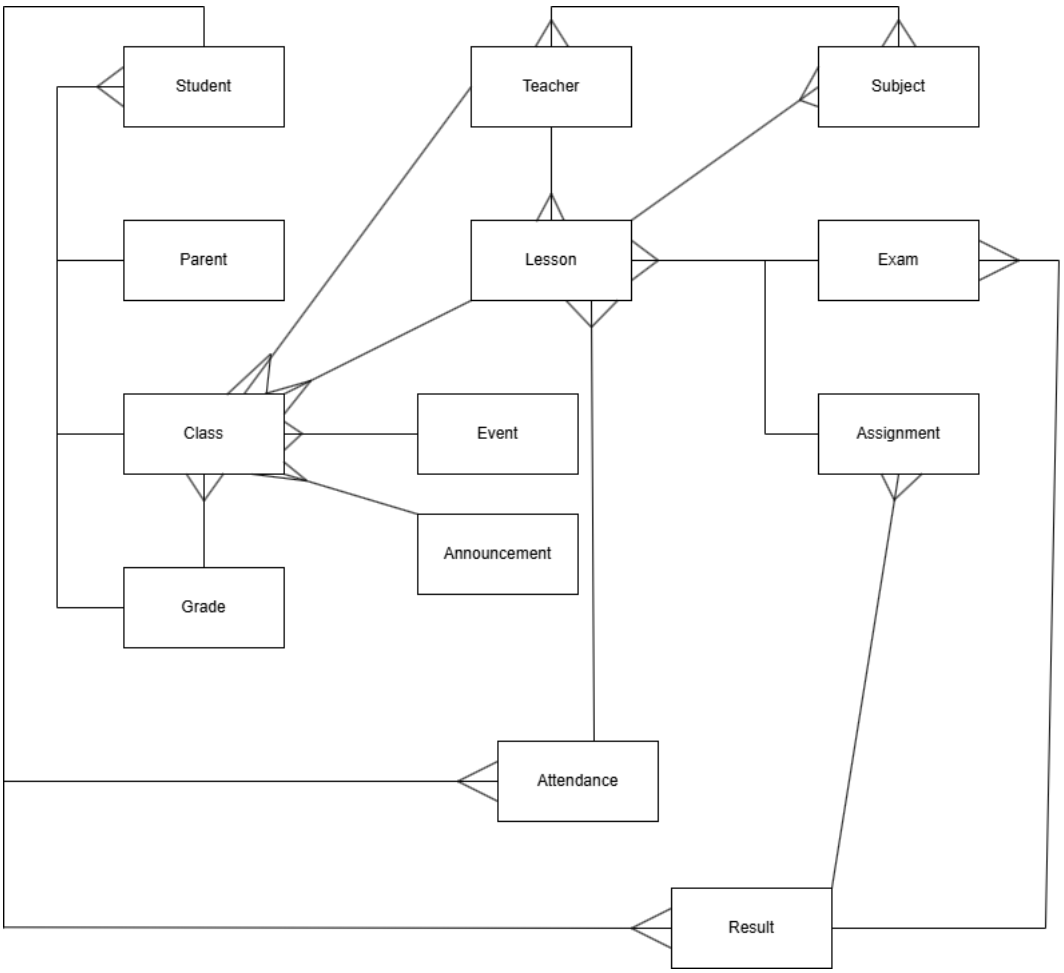

Рисунок 3.13 – ER-diagram суторстей

3.5 Аналіз отриманих результатів

У результаті впровадження проєкту постала сучасна веб-інформаційна система, розроблена для автоматизації ключових аспектів діяльності навчального закладу. Система інтегрує функціонал управління розкладом, ведення електронного журналу, адміністрування облікових записів та забезпечення взаємодії між усіма учасниками освітнього процесу, включаючи учнів, викладачів, батьків та адміністрацію.

Програмне забезпечення базується на передовому технологічному стекі: Next.js застосовується для розробки клієнтської частини, NestJS — для створення логіки на сервері, Prisma — як ORM для роботи з базою даних PostgreSQL, а Clerk — для забезпечення надійної аутентифікації та авторизації користувачів.[27]

Користувацький інтерфейс реалізовано за допомогою Tailwind CSS, що гарантує сучасний вигляд та відмінну адаптивність.

Основні функції, що були впроваджені:

– надійна система реєстрації, входу в систему з урахуванням ролей користувачів (учень, вчитель, батько, адміністратор), а також підтримки сесій;

– інтерактивне відображення списків користувачів (викладачі, учні, батьки) з можливістю фільтрації та пагінації;

– повноцінне представлення розкладу занять, навчальних дисциплін та оцінок;

– можливість адміністрування ключових навчальних процесів та даних для адміністраторів;

– зручний та інтуїтивно зрозумілий інтерфейс, що забезпечує комфортну взаємодію користувачів з будь-якого пристрою.

Підсумовуючи, система є масштабованою, безпечною та легкою у використанні, що відповідає актуальним вимогам до цифрових інструментів в освіті.

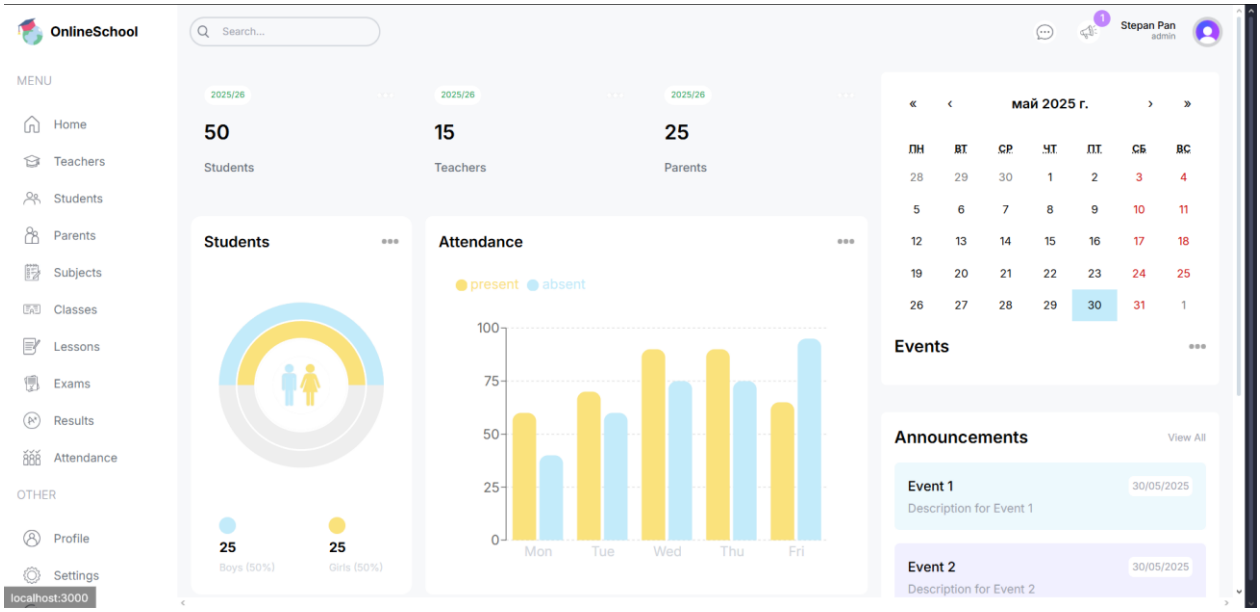

Рисунок 3.14 – Головна сторінка користувача з рівнем доступу admin

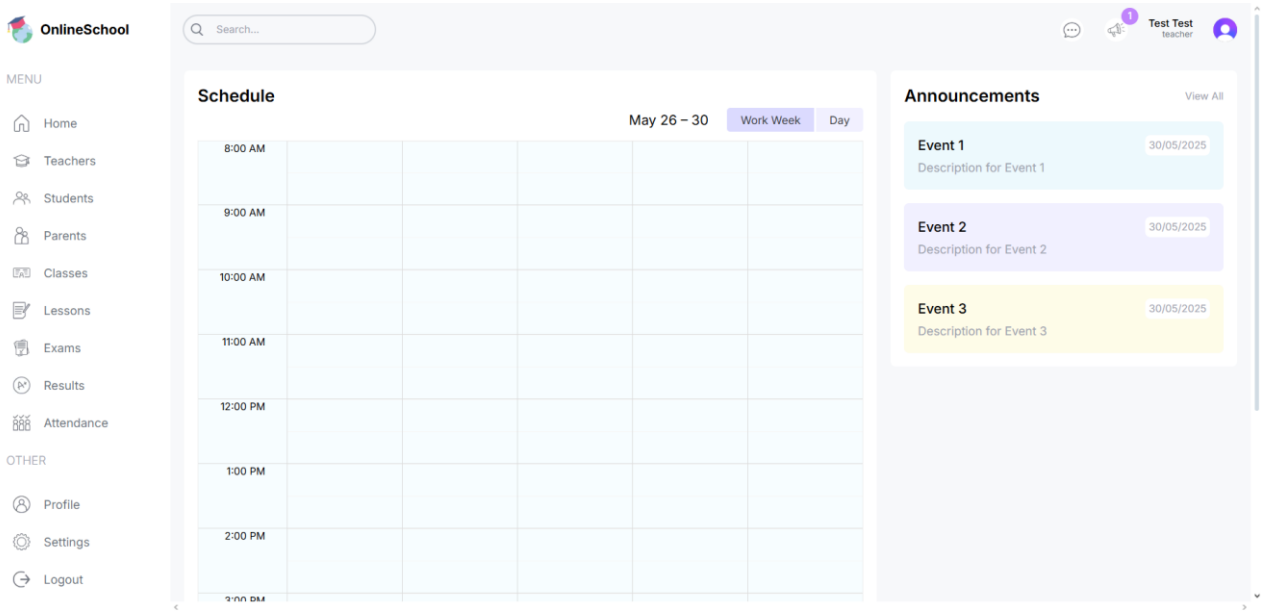

Рисунок 3.15 – Головна сторінка користувача з рівнем доступу teacher

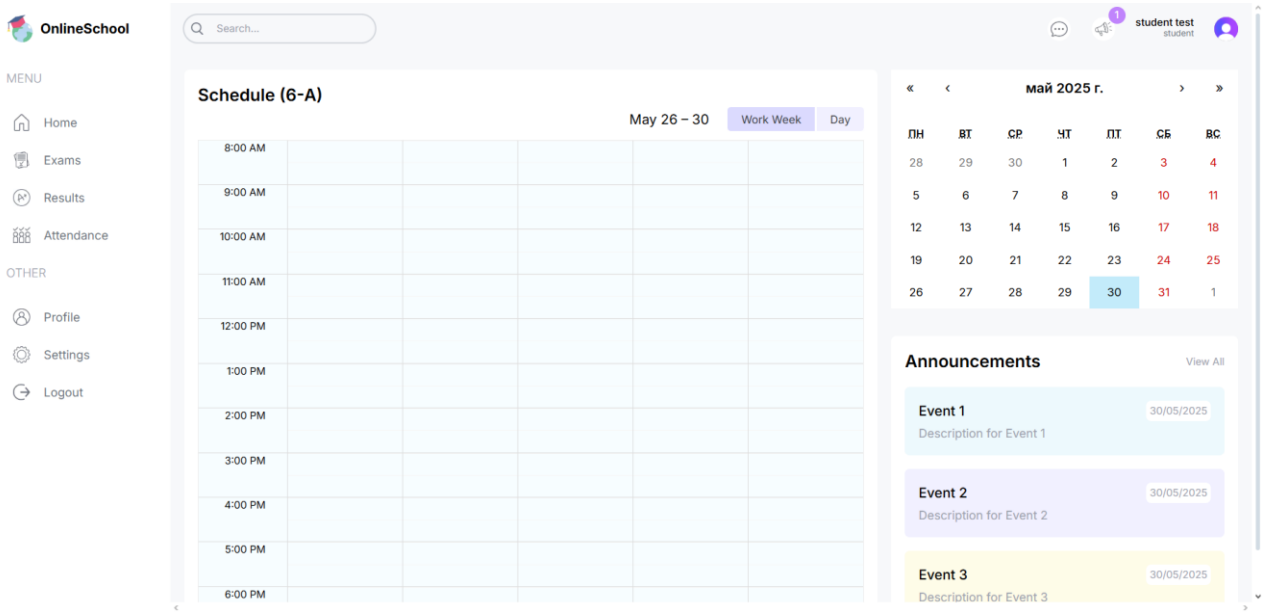

Рисунок 3.16 – Головна сторінка користувача з рівнем доступу teacher

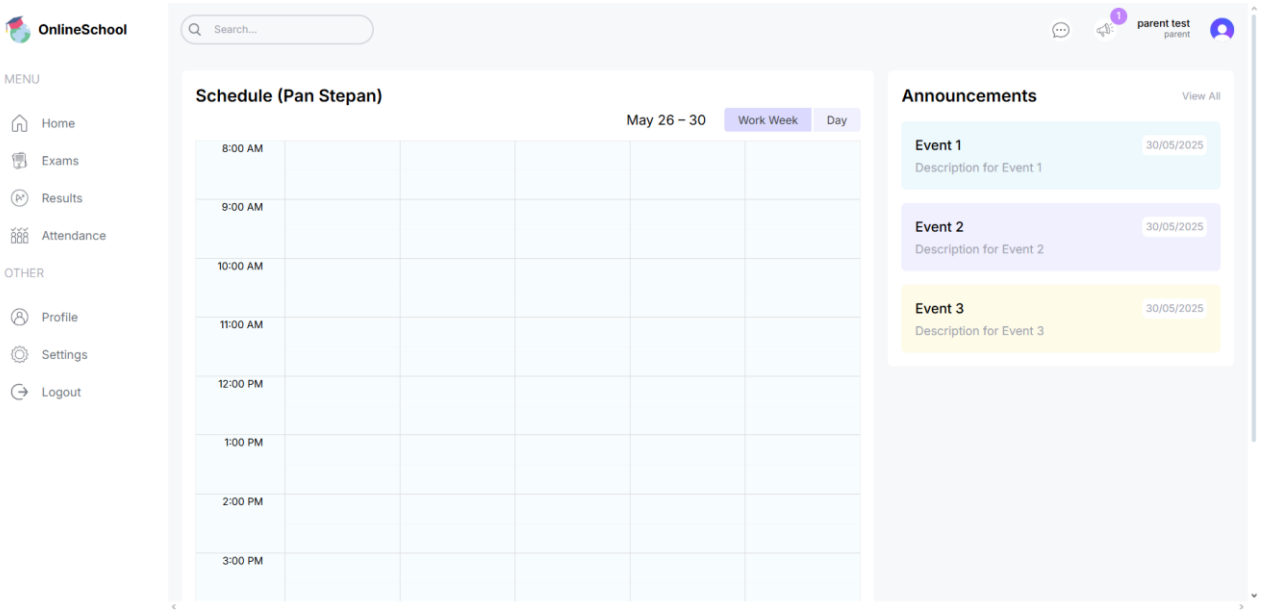

Рисунок 3.17 – Головна сторінка користувача з рівнем доступу parent

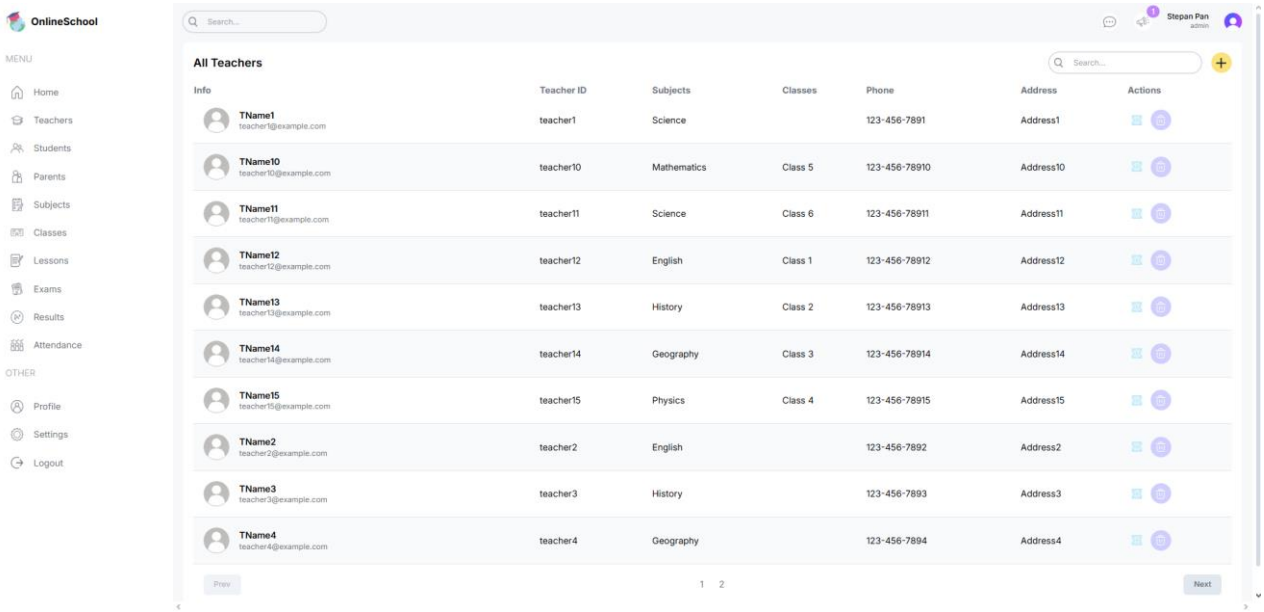

Рисунок 3.18 – Список усіх вчителів

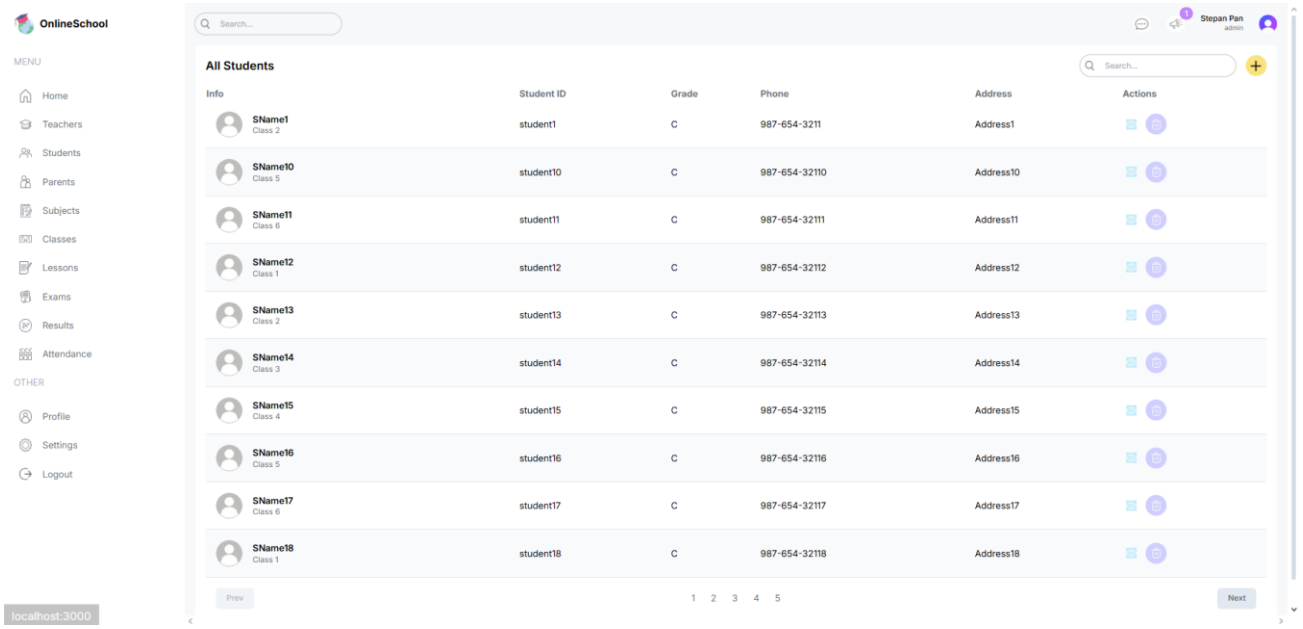

Рисунок 3.19 – Список усіх учнів

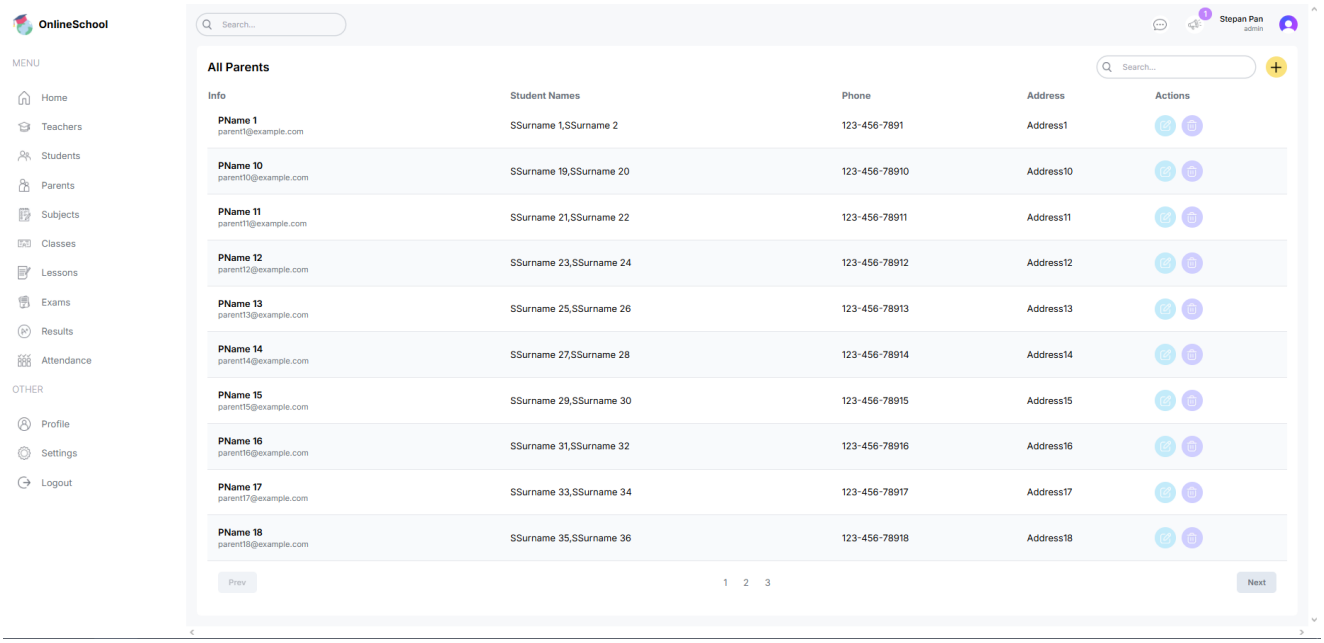

Рисунок 3.20 – Список усіх батьків

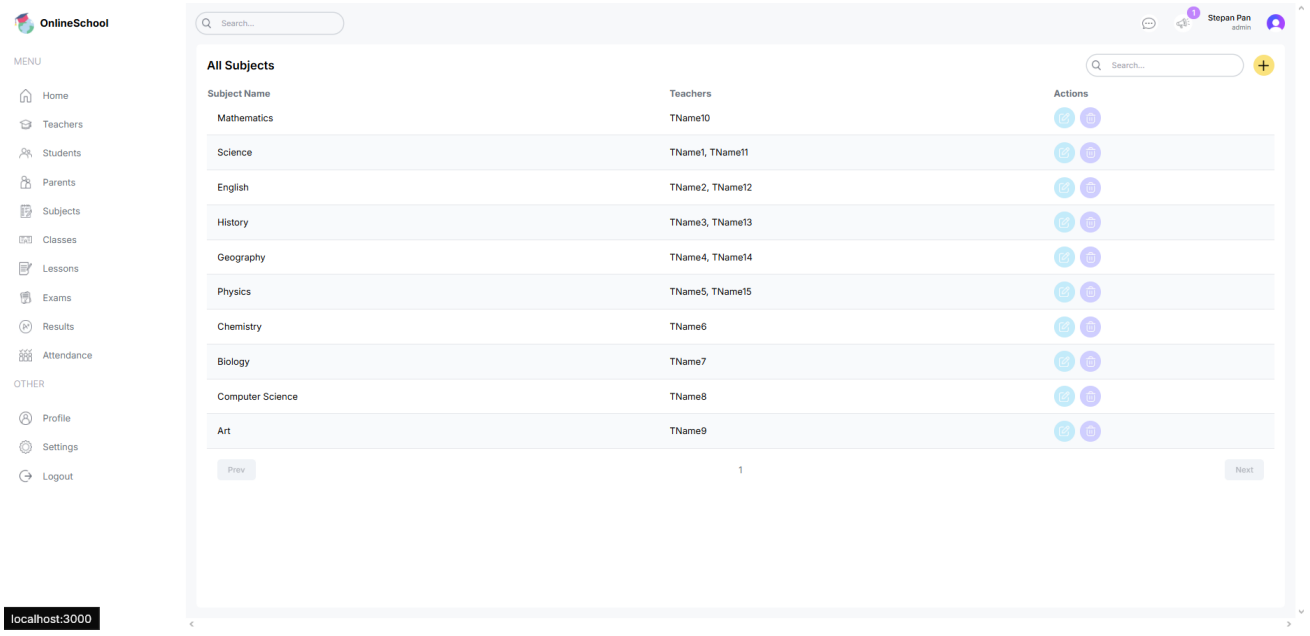

Рисунок 3.21 – Список усіх дисциплін

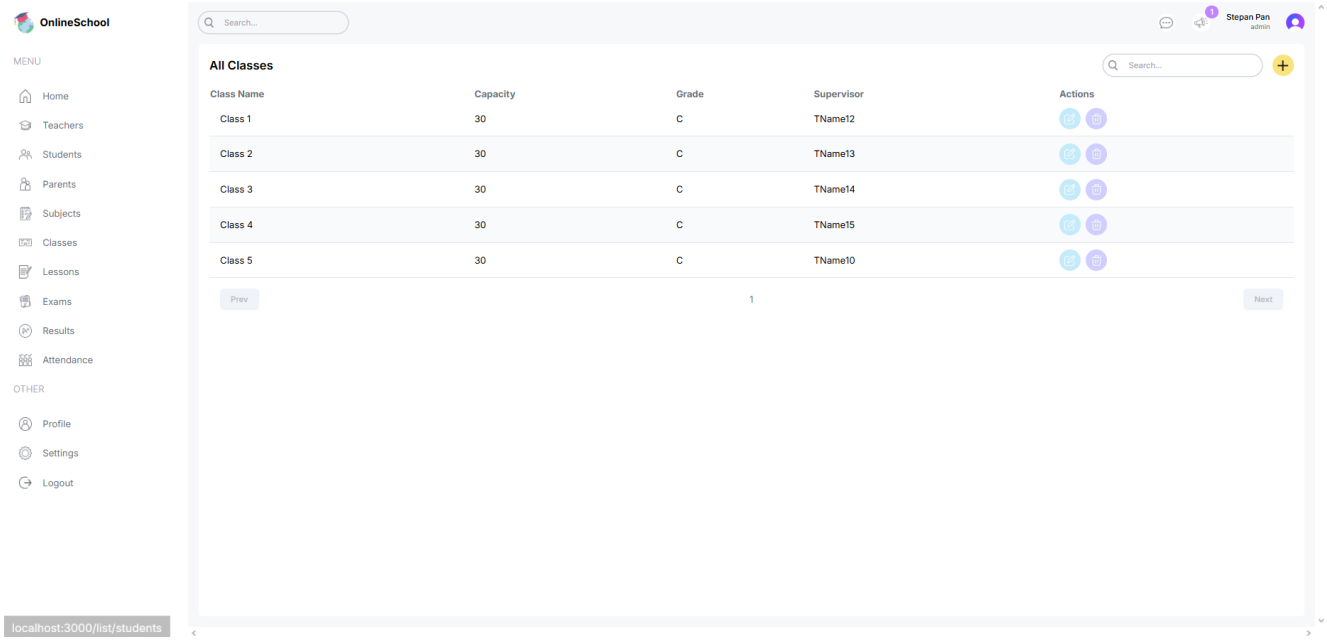

Рисунок 3.22 – Список усіх класів

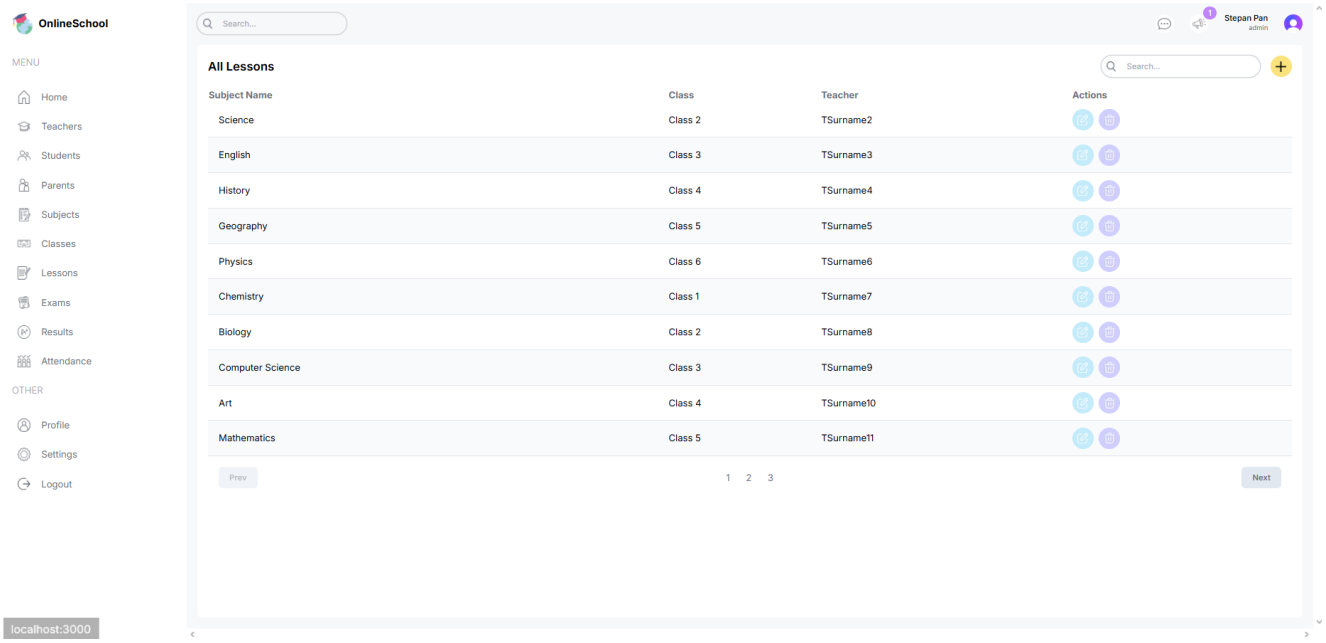

Рисунок 3.23 – Список усіх дисциплін

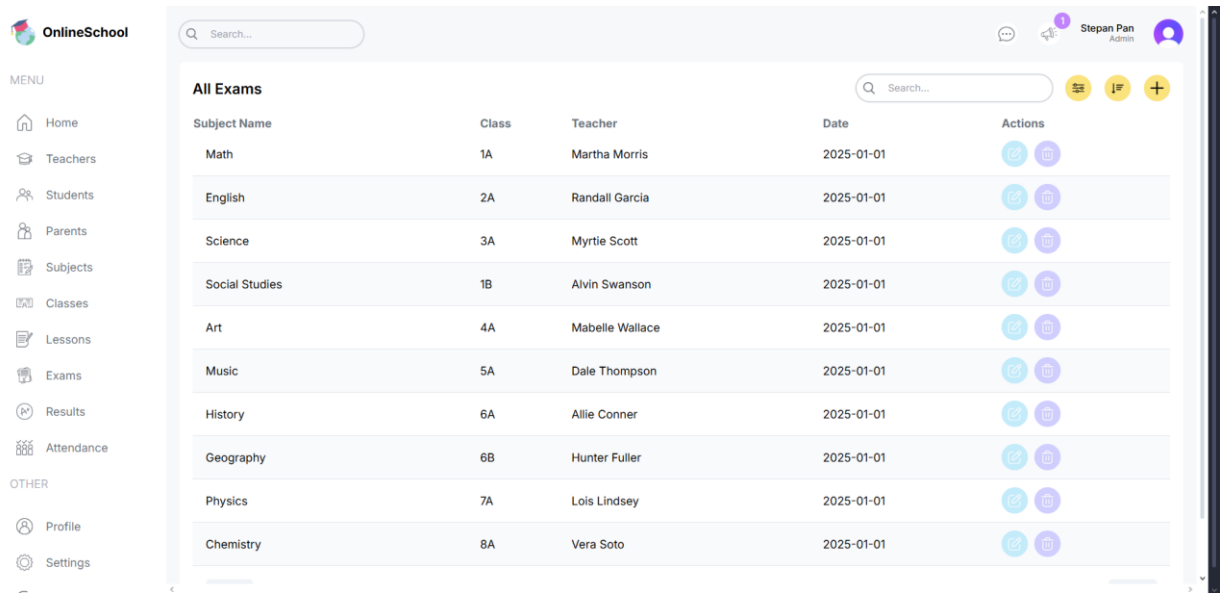

Рисунок 3.24 – Список усіх екзаменів

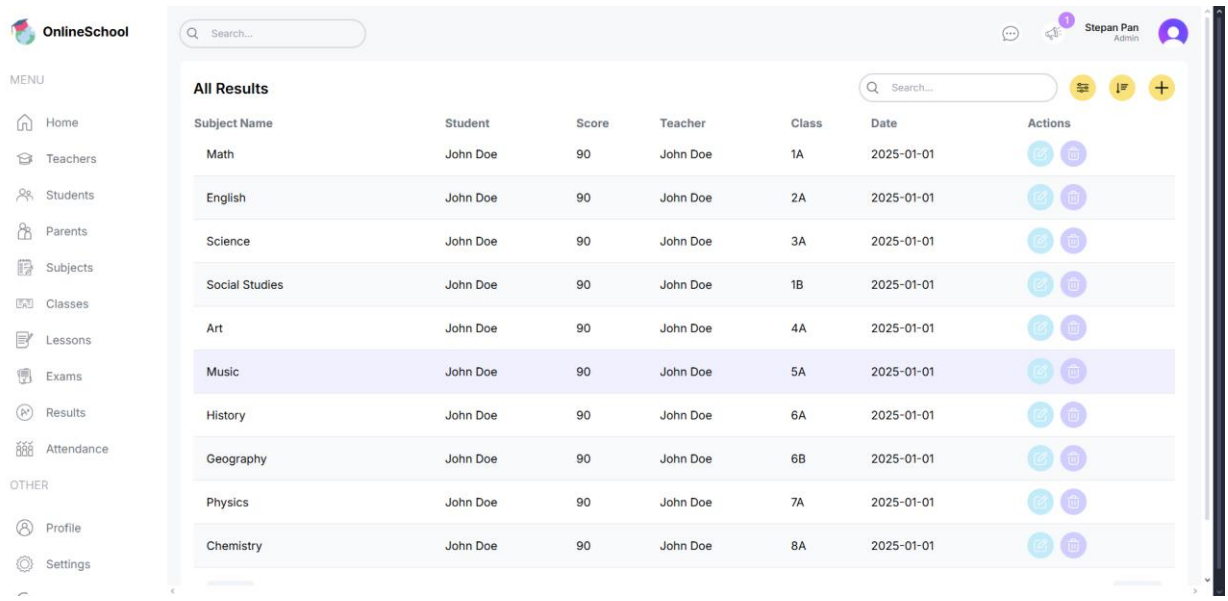

Рисунок 3.25 – Список усіх результатів

Висновки щодо третього розділу:

Під час роботи над інформаційною системою для керування розкладом та електронним журналом було реалізовано низку ключових етапів, включаючи формалізацію вихідних даних, проектування архітектури та моделювання основних складових системи. Детальний розбір структури даних та взаємодії між ключовими об'єктами освітнього процесу дав змогу сформувати логічно цілісну модель, що враховує особливості роботи учнів, вчителів, батьків та адміністрації.

Створена інформаційна система заснована на сучасних технологіях і принципах розробки програмного забезпечення, що забезпечує її надійність, високу продуктивність і здатність до масштабування. Застосування модульного підходу, чітке розподілення обов'язків між компонентами, а також використання REST API полегшують супровід і розвиток системи в майбутньому.

Вбудовані механізми аутентифікації та авторизації гарантують безпечний доступ, а розроблений інтерфейс є зручним та інтуїтивно зрозумілим для користувачів з різними ролями. Функції сортування, пошуку та розбивки на

сторінки оптимізують роботу з великими масивами даних, забезпечуючи швидку та ефективну взаємодію.

Отже, розроблена інформаційна система є потужним інструментом, який сприяє оптимізації роботи навчального закладу, покращує комунікацію між усіма учасниками освітнього середовища та забезпечує гнучке управління навчальним процесом в умовах сучасної цифрової трансформації освіти.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА КЕРІВНИЦТВО КОРИСТУВАЧА

4.1 Опис програмної реалізації

4.1.1 Реалізація авторизації користувачів за їх ролями

Реалізація авторизації користувачів з урахуванням їхніх ролей здійснена через middleware, інтегрований з платформою Clerk у середовищі Next.js. Ключовим елементом авторизації є мапа доступу `routeAccessMap`, що визначає дозволи певних ролей до конкретних маршрутів. При надходженні кожного запиту система аналізує, чи має поточна роль користувача право доступу до запитаного маршруту. У випадку невідповідності ролі, користувач автоматично переадресовується на сторінку, відповідну його ролі. Визначення ролі базується на даних сесії, які зберігаються в метаданих акаунта Clerk. Для ідентифікації шляху використовуються динамічні матчери. Middleware не активується для внутрішніх та статичних ресурсів Next.js, але завжди обробляє API запити. Такий підхід забезпечує централізований та безпечний контроль доступу до компонентів програми. Це знижує ймовірність несанкціонованого доступу та полегшує супровід коду. В результаті, досягнуто зручної та масштабованої системи авторизації, що враховує ролі користувачів.

Реалізація в коді:

```
    import { routeAccessMap } from "@/lib/settings";
    import { clerkMiddleware, createRouteMatcher } from
"@clerk/nextjs/server";
    import { NextResponse } from "next/server";

    const matchers =
Object.keys(routeAccessMap).map((route) => ({
      matcher: createRouteMatcher([route]),
      allowedRoles: routeAccessMap[route],
    }));
```

```
    export default clerkMiddleware(async (auth, req) => {
      // if (isProtectedRoute(req)) auth().protect()

      const { sessionClaims } = await auth();

      const role = (sessionClaims?.metadata as { role?:
string })?.role;

      for (const { matcher, allowedRoles } of matchers) {
        if (matcher(req) && !allowedRoles.includes(role!))
{
          return NextResponse.redirect(new URL(`/${role}`,
req.url));
        }
      }
    });

    export const config = {
      matcher: [
        // Skip Next.js internals and all static files,
unless found in search params

"/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gi
f|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)",
        // Always run for API routes
        "/(api|trpc)(.*)",
      ],
    };
```

4.1.2 Реалізація сторінки входу

Реалізація сторінки входу здійснена на основі React та бібліотеки Clerk, що відповідає за аутентифікацію користувачів. Компонент сторінки функціонує на клієнтській стороні, надаючи інтерактивну форму для введення логіну та паролю з реалізованою валідацією та візуалізацією помилок. Хук useUser застосовується для отримання інформації про статус авторизації користувача, а useEffect слугує для автоматичного перенаправлення користувача на відповідну сторінку залежно від його ролі. Для стилізації використовується Tailwind CSS, забезпечуючи сучасний

та привабливий візуальний стиль. Компонент включає в себе логотип, заголовки та кнопку "Увійти", що запускає процес автентифікації. Цей підхід гарантує безпечний та зрозумілий процес входу для користувачів з різними правами доступу.

Реалізація у коді:

```
    "use client";

    import * as Clerk from "@clerk/elements/common";
    import * as SignIn from "@clerk/elements/sign-in";
    import { useUser } from "@clerk/nextjs";
    import Image from "next/image";
    import { useRouter } from "next/navigation";
    import { useEffect } from "react";

    const LoginPage = () => {
      const { isLoaded, isSignedIn, user } = useUser();

      const router = useRouter();

      useEffect(() => {
        const role = user?.publicMetadata.role;

        if (role) {
          router.push(`/${role}`);
        }
      }, [user, router]);

      return (
        <div className="h-screen flex items-center justify-
center bg-lamaSkyLight">
          <SignIn.Root>
            <SignIn.Step
              name="start"
              className="bg-white p-12 rounded-md shadow-
2xl flex flex-col gap-2"
            >
              <h1 className="text-xl font-bold flex items-
center gap-2">
                <Image src="/logo.png" alt="" width={24}
height={24} />
                OnlineSchool
```

```
              </h1>
              <h2 className="text-gray-400">Sign in to your
account</h2>
              <Clerk.GlobalError  className="text-sm  text-
red-400" />
              <Clerk.Field                name="identifier"
className="flex flex-col gap-2">
                <Clerk.Label  className="text-xs  text-gray-
500">
                  Username
                </Clerk.Label>
                <Clerk.Input
                  type="text"
                  required
                  className="p-2  rounded-md  ring-1  ring-
gray-300"
                />
                <Clerk.FieldError  className="text-xs  text-
red-400" />
              </Clerk.Field>
              <Clerk.Field name="password" className="flex
flex-col gap-2">
                <Clerk.Label  className="text-xs  text-gray-
500">
                  Password
                </Clerk.Label>
                <Clerk.Input
                  type="password"
                  required
                  className="p-2  rounded-md  ring-1  ring-
gray-300"
                />
                <Clerk.FieldError  className="text-xs  text-
red-400" />
              </Clerk.Field>
              <SignIn.Action
                submit
                className="bg-blue-500   text-white   my-1
rounded-md text-sm p-[10px]"
              >
                Sign In
              </SignIn.Action>
            </SignIn.Step>
          </SignIn.Root>
```

```
        </div>
      );
    };

    export default LoginPage;
```

4.1.3 Реалізація сайдбар меню

Реалізація компонента бічного меню для веб-додатку була виконана за допомогою Next.js та служби автентифікації Clerk. Меню генерується динамічно, базуючись на ролі активного користувача, яка вилучається з його відкритих метаданих. Структура меню розділена на два головні розділи — "MENU" та "OTHER", кожен з яких містить набір пунктів з відповідними іконками, назвами та URL-адресами. Для кожного пункту задано список ролей, яким дозволено його бачити, що забезпечує контроль доступу на рівні інтерфейсу. Компонент використовує серверну функцію `currentUser()` для отримання даних користувача, а також враховує адаптивність: на невеликих екранах відображаються тільки іконки, а на великих — повний текст з підписами. Кожен елемент меню — це інтерактивне посилання, що веде до відповідної сторінки додатку. Таким чином, реалізовано зручну навігацію, яка відповідає як потребам користувача, так і його правам доступу.

Реалізація у коді:

```
    import Link from "next/link";
import Image from "next/image";
import { role } from "@/lib/data";
import { currentUser } from "@clerk/nextjs/server";

const menuItems = [
  {
    title: "MENU",
    items: [
      {
        icon: "/home.png",
        label: "Home",
        href: "/",
```

```
        visible: ["admin", "teacher", "student", "parent"],
      },
      {
        icon: "/teacher.png",
        label: "Teachers",
        href: "/list/teachers",
        visible: ["admin", "teacher"],
      },
      {
        icon: "/student.png",
        label: "Students",
        href: "/list/students",
        visible: ["admin", "teacher"],
      },
      {
        icon: "/parent.png",
        label: "Parents",
        href: "/list/parents",
        visible: ["admin", "teacher"],
      },
      {
        icon: "/subject.png",
        label: "Subjects",
        href: "/list/subjects",
        visible: ["admin"],
      },
      {
        icon: "/class.png",
        label: "Classes",
        href: "/list/classes",
        visible: ["admin", "teacher"],
      },
      {
        icon: "/lesson.png",
        label: "Lessons",
        href: "/list/lessons",
        visible: ["admin", "teacher"],
      },
      {
        icon: "/exam.png",
        label: "Exams",
        href: "/list/exams",
        visible: ["admin", "teacher", "student", "parent"],
      },
```

```
        {
          icon: "/result.png",
          label: "Results",
          href: "/list/results",
          visible: ["admin", "teacher", "student", "parent"],
        },
        {
          icon: "/attendance.png",
          label: "Attendance",
          href: "/list/attendance",
          visible: ["admin"],
        },
      ],
    },
    {
      title: "OTHER",
      items: [
        {
          icon: "/profile.png",
          label: "Profile",
          href: "/profile",
          visible: ["admin", "teacher", "student", "parent"],
        },
        {
          icon: "/setting.png",
          label: "Settings",
          href: "/settings",
          visible: ["admin", "teacher", "student", "parent"],
        },
        {
          icon: "/logout.png",
          label: "Logout",
          href: "/logout",
          visible: ["admin", "teacher", "student", "parent"],
        },
      ],
    },
];

const Menu = async () => {
  const user = await currentUser();
  const role = user?.publicMetadata.role as string;
  return (
    <div className="mt-4 text-sm">
```

```
      {menuItems.map((i) => (
        <div className="flex flex-col gap-2" key={i.title}>
          <span className="hidden lg:block text-gray-400
font-light my-4">
            {i.title}
          </span>
          {i.items.map((item) => {
            if (item.visible.includes(role)) {
              return (
                <Link
                  href={item.href}
                  key={item.label}
                  className="flex items-center justify-
center lg:justify-start gap-4 text-gray-500 py-2 md:px-2
rounded-md hover:bg-schoolSkyLight"
                >
                  <Image src={item.icon} alt="" width={20}
height={20} />
                  <span className="hidden
lg:block">{item.label}</span>
                </Link>
              );
            }
          })}
        </div>
      ))}
    </div>
  );
};

export default Menu;
```

4.1.4 Схема бази даних

Схема бази даних розроблена з застосуванням Prisma Schema Language (PSL), що надає змогу ефективно визначати структуру реляційної бази даних. У цій схемі визначено важливі моделі, які описують головні сутності освітньої інформаційної системи: адміністратор, учень, учитель, батько, клас, предмет, урок, екзамен, завдання, оцінка, відвідуваність, подія та оголошення. Для кожної моделі визначені

унікальні ідентифікатори, ключові атрибути, а також створено зв'язки між таблицями за допомогою директив `@relation`. Деякі поля використовують перерахування, зокрема `UserSex` для визначення статі та `Day` для днів тижня. Джерелом даних виступає PostgreSQL, з'єднання з яким встановлюється через змінну середовища `DATABASE_URL`. Для взаємодії з базою даних згенеровано клієнт `prisma-client-js`, який забезпечує взаємодію між програмою та базою даних. Така структура підтримує логіку доменної області освітнього закладу та дозволяє гнучко керувати користувачами, процесом навчання та супровідними даними.

Реалізація у коді:

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Admin {
  id       String @id
  username String @unique
}

model Student {
  id          String       @id
  username    String       @unique
  name        String
  surname     String
  email       String?      @unique
  phone       String?      @unique
  address     String
  img         String?
  bloodType   String
  sex         UserSex
  createdAt   DateTime     @default(now())
  parentId    String
  classId     Int
  gradeId     Int
  birthday    DateTime
```

```
      attendances Attendance[]
      results     Result[]
      class       Class        @relation(fields: [classId],
references: [id])
      grade       Grade        @relation(fields: [gradeId],
references: [id])
      parent      Parent       @relation(fields: [parentId],
references: [id])
    }

    model Teacher {
      id         String    @id
      username   String    @unique
      name       String
      surname    String
      email      String?   @unique
      phone      String?   @unique
      address    String
      img        String?
      bloodType  String
      sex        UserSex
      createdAt  DateTime  @default(now())
      birthday   DateTime
      classes    Class[]
      lessons    Lesson[]
      subjects   Subject[] @relation("SubjectToTeacher")
    }

    model Parent {
      id         String    @id
      username   String    @unique
      name       String
      surname    String
      email      String?   @unique
      phone      String    @unique
      address    String
      createdAt  DateTime  @default(now())
      students   Student[]
    }

    model Grade {
      id       Int       @id @default(autoincrement())
      level    Int       @unique
      classess Class[]
```

```
      students Student[]
    }

    model Class {
      id          Int          @id @default(autoincrement())
      name          String         @unique
      capacity      Int
      supervisorId  String?
      gradeId       Int
      announcements Announcement[]
      grade            Grade              @relation(fields:
[gradeId], references: [id])
      supervisor      Teacher?           @relation(fields:
[supervisorId], references: [id])
      events        Event[]
      lessons       Lesson[]
      students      Student[]
    }

    model Subject {
      id       Int       @id @default(autoincrement())
      name     String    @unique
      lessons  Lesson[]
      teachers Teacher[] @relation("SubjectToTeacher")
    }

    model Lesson {
      id          Int          @id @default(autoincrement())
      name         String
      day          Day
      startTime    DateTime
      endTime      DateTime
      subjectId    Int
      classId      Int
      teacherId    String
      assignments Assignment[]
      attendances Attendance[]
      exams        Exam[]
      class        Class        @relation(fields: [classId],
references: [id])
      subject    Subject     @relation(fields: [subjectId],
references: [id])
      teacher    Teacher     @relation(fields: [teacherId],
references: [id])
```

```
    }

    model Exam {
      id        Int      @id @default(autoincrement())
      title     String
      startTime DateTime
      endTime   DateTime
      lessonId  Int
      lesson      Lesson     @relation(fields:  [lessonId],
references: [id])
      results   Result[]
    }

    model Assignment {
      id        Int      @id @default(autoincrement())
      title     String
      startDate DateTime
      dueDate   DateTime
      lessonId  Int
      lesson      Lesson     @relation(fields:  [lessonId],
references: [id])
      results   Result[]
    }

    model Result {
      id           Int        @id @default(autoincrement())
      score        Int
      examId       Int?
      assignmentId Int?
      studentId    String
      assignment           Assignment?    @relation(fields:
[assignmentId], references: [id])
      exam         Exam?       @relation(fields: [examId],
references: [id])
      student     Student    @relation(fields: [studentId],
references: [id])
    }

    model Attendance {
      id        Int      @id @default(autoincrement())
      date      DateTime
      present   Boolean
      studentId String
      lessonId  Int
```

```
      lesson       Lesson     @relation(fields: [lessonId],
references: [id])
      student     Student    @relation(fields: [studentId],
references: [id])
    }

    model Event {
      id          Int      @id @default(autoincrement())
      title       String
      description String
      startTime   DateTime
      endTime     DateTime
      classId     Int?
      class         Class?    @relation(fields: [classId],
references: [id])
    }

    model Announcement {
      id          Int      @id @default(autoincrement())
      title       String
      description String
      date        DateTime
      classId     Int?
      class         Class?    @relation(fields: [classId],
references: [id])
    }

    enum UserSex {
      MALE
      FEMALE
    }

    enum Day {
      MONDAY
      TUESDAY
      WEDNESDAY
      THURSDAY
      FRIDAY
    }
```

4.2 Керівництво користувача

Розроблена інформаційна система – це вебзастосунок, який можна використовувати через браузер на будь-якому пристрої, підключеному до Інтернету. Інтерфейс програми спроектований з урахуванням зручності, доступності та мінімалізму, тому робота з системою не потребує особливих навичок.

Загальні принципи взаємодії з системою. Після переходу за адресою вебзастосунку, користувач бачить головну сторінку з можливістю входу до системи. Для кожної категорії користувачів (адміністратор, вчитель, учень, батьки) реалізовано відповідний функціонал, враховуючи ролі та рівні доступу.

Авторизація та реєстрація:

– вхід в систему відбувається за допомогою логіна (email) та пароля;

– після авторизації користувач потрапляє до особистого кабінету відповідно до своєї ролі;

– адміністратор має можливість створювати нові облікові записи для учнів, батьків та вчителів.

4.2.1 Керівництво користувача: учень

Після успішного входу учень опиняється в своєму персональному кабінеті, що містить такі секції:

– мій профіль - тут відображаються персональні дані (ПІБ, клас, фото, адреса електронної пошти). Передбачено можливість коригування власних контактних даних;

– мій розклад - учень має доступ до списку уроків на поточний тиждень, де вказано назву предмету, час проведення заняття, ім'я викладача та номер аудиторії;

– оцінки та журнал - учень може переглядати свої оцінки з предметів, коментарі від викладачів та інформацію про відвідування уроків;

– мої вчителі - містить список викладачів, які викладають у цього учня, з можливістю перегляду їхніх профілів;

– мої екзамени - тут зібрано перелік майбутніх іспитів або контрольних робіт із зазначенням дат проведення.

Користувач не має права змінювати інформацію інших користувачів або структуру самої системи.

4.2.2 Керівництво користувача: вчитель

Після авторизації вчитель переноситься в особистий простір, де доступні такі опції:

– мій профіль - інформація про вчителя, включно з можливістю оновлення контактних даних;

– розклад вчителя - показ всіх запланованих занять вчителя, з інформацією про предмет, клас, час та аудиторію;

– мої класи/учні - доступ до переліків класів, у яких викладає вчитель. Також є можливість перегляду списку учнів кожного класу;

– електронний журнал - функціонал, що дозволяє виставляти оцінки, фіксувати присутність/відсутність учнів, додавати коментарі до уроків або про учнів, вносити зміни до записів протягом обмеженого часу;

– профілі учнів - доступ до короткої інформації про кожного учня з предметів, які викладає користувач;

– екзамени - створення та коригування даних про екзаменаційні роботи, що стосуються його дисциплін.

Викладач не має права керувати адміністративними налаштуваннями користувачів або розкладом інших викладачів.

4.2.3 Керівництво користувача: батьки

Батьки отримують доступ до системи, щоб слідкувати за успішністю своїх дітей. Після входу в систему відкриваються наступні можливості:

– оцінки дитини - відображення оцінок з усіх дисциплін кожної дитини;

– відвідування дитини - відомості про пропуски, запізнення чи відвідуваність;

– розклад дитини - актуальний графік занять;

– вчителі дитини - список вчителів, з можливістю перегляду їхніх профілів та контактних даних.

Батьки не мають права вносити зміни до навчальних даних, але можуть їх переглядати.

4.2.4 Керівництво користувача: адміністратор

Адміністратор наділений широкими правами для управління всіма складовими системи. Йому доступні наступні операції:

– управління користувачами - створення/редагування/видалення акаунтів учнів, вчителів, батьків, а також призначення ролей, зміна паролів;

– управління розкладом - додавання/редагування розкладу занять для класів та викладачів. Також уникнення конфліктів у навантаженні;

– управління дисциплінами - створення нових предметів та закріплення викладачів за предметами та класами;

– управління журналом - перегляд загальної статистики успішності, доступ до журналів всіх викладачів;

– управління екзаменами - створення, редагування або видалення даних про контрольні роботи та екзамени.

Адміністратор є єдиним користувачем, що має доступ до всіх розділів і відповідає за актуальність даних у системі.

Висновки щодо четвертого розділу:

У четвертому розділі було подано детальний опис програмної частини системи. Зокрема, розглянуто способи авторизації користувачів, прив'язані до їхніх ролей. Це гарантує захист даних та правильний розподіл прав доступу. Окрім цього, представлено покрокові інструкції для різних категорій користувачів: учнів, вчителів, батьків та адміністраторів. Це значно полегшує роботу з системою. Чітка структура та продуманий дизайн інтерфейсів дають змогу користувачам легко знаходити потрібні функції, відповідно до їхніх ролей. Створена система ефективно керує розкладом та електронним журналом у навчальних закладах, що значно спрощує навчальний процес та адміністративну роботу. Отже, застосовані рішення підтверджують ефективність обраних технологій та підходів для створення інтерактивної та захищеної освітньої платформи.

ВИСНОВКИ

Під час виконання завдання за власною темою кваліфікаційної роботи було створено інформаційну систему, що призначена для управління шкільним розкладом та електронним журналом. Під час розробки системи було укріплено навички веб-розробки, роботи з базами даних та загального структурування інформаційних систем. Реалізовано функціонал для школярів, вчителів та батьків, що надає зручний доступ до розкладу, оцінок та іншої важливої інформації щодо навчального процесу.

Для втілення системи в життя застосовано сучасні технології, які гарантують високу продуктивність, стабільність та захист. Next.js, NestJS, Prisma та зручне середовище Visual Studio Code дозволили створити систему з потенціалом до легкого масштабування та можливістю подальшого вдосконалення.

Здобутий досвід показав важливість чіткої організації проєкту, розподілу функціональності між фронтендом та бекендом, а також значення тестування та перевірки даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Google Classroom Help. Google Support: довідковий центр. URL: https://support.google.com/edu/classroom

2) Логотип Google classroom. URL: https://ctl.uaf.edu/wp-content/uploads/sites/2/2019/05/google-classroom-logo-1024x1024.png

3) Moodle Developer Docs. Moodle: вебсайт. URL: https://moodledev.io/docs

4) Логотип Moodle. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Moodle-logo.svg/2560px-Moodle-logo.svg.png

5) Нові знання. Нові знання: освітня платформа. URL: https://nz.ua

6) Логотип «Нові знання». URL: https://play-lh.googleusercontent.com/UC-6HpOzy5A5lLt4PsMAW3Lfpy8yYmec5EMgRajGGEgmOEYyQmXwzQlRED1z8wfJ4mcz

7) ДІСО - Державна інформаційна система освіти. DISO.gov.ua: урядовий сайт. URL: https://diso.gov.ua

8) Логотип ДіСО. URL: https://osvita-omr.gov.ua/wp-content/uploads/2023/06/dyso.png

9) AIKOM. Що таке AIKOM? URL: https://aikom.iea.gov.ua/site/about

10) Логотип AIKOM. URL: https://aikom.iea.gov.ua/images/redizine/main-logo.png

11) Основи ООП. EPAM Campus: освітній портал. URL: https://campus.epam.ua/ua/blog/275

12) Visual Studio Code. Wikipedia: енциклопедичний вебсайт. URL: https://uk.wikipedia.org/wiki/Visual_Studio_Code

13) Логотип Visual Studio Code. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/9/9a/Visual_Studio_Code_1.35_icon.svg/1200px-Visual_Studio_Code_1.35_icon.svg.png

14) Next.js Documentation. Next.js: вебсайт. URL: https://nextjs.org/docs

15) Логотип Next.js. URL: https://miro.medium.com/v2/resize:fit:720/1*_bJ2z2NRfTncHAv5UjUxwA.jpeg

16) Tailwind CSS Guide. DreamHost: блог. URL: https://www.dreamhost.com/blog/uk/tailwind-css/

17) Tailwind CSS Docs. TailwindCSS: вебсайт. URL: https://tailwindcss.com/docs/installation/using-vite

18) Логотип Tailwind. URL: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQNhoXisDruJMDAq3Ltd-wuaMW2lGxck9wAKw&s

19) NestJS Documentation. NestJS: вебсайт. URL: https://docs.nestjs.com

20) Логотип NestJS. URL: https://upload.wikimedia.org/wikipedia/commons/a/a8/NestJS.svg

21) Prisma Documentation. Prisma: офіційний сайт. URL: https://www.prisma.io/docs

22) Логотип Prisma. URL: https://cdn.worldvectorlogo.com/logos/prisma-3.svg

23) Axios Documentation. Axios: вебсайт. URL: https://axios-http.com/uk/docs/

24) Логотип Axios. URL: https://cdn.prod.website-files.com/5e19ea5aa7d3a217492e372b/624de949df5a11680ab170b9_Axios%20logo%20-%20RGB%20-%20minimum%20space.png

25) Model–view–controller. Wikipedia: вебсайт. URL: https://en.wikipedia.org/wiki/Model–view–controller

26) MVC Framework Introduction. GeeksforGeeks: вебсайт. URL: https://www.geeksforgeeks.org/mvc-framework-introduction/

27) Clerk Docs (аутентифікація). Clerk: офіційний сайт. URL: https://clerk.com/docs

ДОДАТОК А Лістинг деякої частини коду

teacher.module.ts:

```
import { Module } from '@nestjs/common';
import { TeacherService } from './teacher.service';
import { TeacherController } from
'./teacher.controller';
import { PrismaService } from
'src/prisma/prisma.service';
import { FileService } from 'src/file/file.service';

@Module({
  controllers: [TeacherController],
  providers: [TeacherService, PrismaService,
FileService],
})
export class TeacherModule {}
```

teacher.controller.ts:

```
import { Body, Controller, Delete, Get, Param, Post,
Query, UploadedFiles, UseInterceptors } from
'@nestjs/common';
import { TeacherService } from './teacher.service';
import { FileFieldsInterceptor } from
"@nestjs/platform-express";
import { CreateTeacherDto } from './dto/teacher-
create.dto';
@Controller('teachers')
export class TeacherController {
  constructor(private readonly teacherService:
TeacherService) {}

  @Get()
  async getAllTeachers(
    @Query('page') page = '1',
    @Query('limit') limit = '10',
    @Query('search') search: string,
  ) {
```

```
        return this.teacherService.findAll(+page, +limit,
search);
      }

      @Get('/all')
      async getAll() {
        return this.teacherService.getAll();
      }

      @Get(':id')
      async getOne(@Param('id') id: string) {
        return this.teacherService.getOne(id);
      }

      @Get('/select')
      async getSelectTeacher() {
        return this.teacherService.getSelectTeacher();
      }

      @Post()
      @UseInterceptors(FileFieldsInterceptor([
        { name: 'img', maxCount: 1 },
      ]))
      public async create(
        @UploadedFiles() files: { img?:
Express.Multer.File[] },
        @Body() dto: CreateTeacherDto
      ) {
          const image = files.img?.[0];
          return this.teacherService.create(dto, image);
      }

      @Delete()
      async delete(@Body('id') id: string) {
        return this.teacherService.delete(id);
      }
    }
```

teacher.service.ts:

```
    import { Injectable } from '@nestjs/common';
    import { FileService, FileType } from
'src/file/file.service';
    import { PrismaService } from
'src/prisma/prisma.service';
    import { CreateTeacherDto } from './dto/teacher-
create.dto';
    import { UserSex } from '@prisma/client';

    @Injectable()
    export class TeacherService {
      constructor(private readonly prisma: PrismaService,
private readonly fileService: FileService) {}

      async findAll(page: number, limit: number, search?:
string) {
        const skip = (page - 1) * limit;

        const where: any = {};

        if (search) {
          where.OR = [
            {
              name: {
                contains: search,
                mode: 'insensitive',
              },
            },
          ];
        }

        const [teachers, count] = await
this.prisma.$transaction([
          this.prisma.teacher.findMany({
            where,
            include: {
              subjects: true,
              classes: true,
            },
            skip,
            take: limit,
          }),
          this.prisma.teacher.count({ where }),
```

```
        ]);

        return { teachers, count };
      }

      async getAll() {
        const [teachers] = await
this.prisma.$transaction([
          this.prisma.teacher.findMany(),
        ]);

        return teachers;
      }

      async getSelectTeacher() {
        const [teachers] = await
this.prisma.$transaction([
          this.prisma.teacher.findMany({
            select: { id: true, name: true, surname: true
},
          }),
        ]);


        return teachers;
      }

      public async create(dto: CreateTeacherDto, image?:
Express.Multer.File) {
        let img = {url: ''};
        if(image) img = await
this.fileService.uploadToCloudinary(image, FileType.IMAGE);

        const numberSubjects = dto.subjects.map((s) =>
Number(s));

        const subjects = await
this.prisma.subject.findMany({
          where: {
            id: {
              in: numberSubjects,
            },
          },
          select: { id: true },
```

```
        });

        const teacher = await this.prisma.teacher.create({
          data: {
            id: "teacher" + Number(Math.random() * 1000),
            name: dto.name,
            username: dto.username,
            surname: dto.surname,
            email: dto.email || null,
            phone: dto.phone || null,
            address: dto.address,
            img: img.url || null,
            bloodType: dto.bloodType,
            sex: dto.sex === 'MALE' ? UserSex.MALE :
UserSex.FEMALE,
            birthday: dto.birthday,
            subjects: {
              connect: subjects.map((s) => ({ id: s.id
})),
            },
          },
        });

        return teacher;
      }

      public async delete(id: string) {
        return this.prisma.teacher.delete({
          where: { id },
        });
      }

      public async getOne(id: string) {
        const [teachers] = await
this.prisma.$transaction([
          this.prisma.teacher.findMany({
            where: {id},
            include: {
              subjects: true,
              classes: true,
              lessons: true
            },
          }),
        ]);
```

```
        return teachers;
      }
    }
```

student.module.ts:

```
    import { Module } from '@nestjs/common';
    import { StudentService } from './student.service';
    import { StudentController } from
'./student.controller';
    import { FileService } from 'src/file/file.service';

    @Module({
      controllers: [StudentController],
      providers: [StudentService, FileService],
    })
    export class StudentModule {}
```

student.controller.ts:

```
    import { Body, Controller, Delete, Get, Param, Post,
Query, UploadedFiles, UseInterceptors } from
'@nestjs/common';
    import { StudentService } from './student.service';
    import { FileFieldsInterceptor } from
'@nestjs/platform-express';
    import { CreateStudentDto } from './dto/create-
student.dto';

    @Controller('students')
    export class StudentController {
      constructor(private readonly studentService:
StudentService) {}

      @Get()
      async getAllStudents(
        @Query('page') page = '1',
        @Query('limit') limit = '10',
        @Query('search') search: string,
      ) {
```

```
        return this.studentService.findAll(+page, +limit,
search);
      }

      @Get('/all')
      async getAll() {
        return this.studentService.getAll();
      }

      @Get(':id')
      async getOne(@Param('id') id: string) {
        return this.studentService.getOne(id);
      }


      @Post()
      @UseInterceptors(FileFieldsInterceptor([
        { name: 'img', maxCount: 1 },
      ]))
      public async create(
        @UploadedFiles() files: { img?:
Express.Multer.File[] },
        @Body() dto: CreateStudentDto
      ) {
          const image = files.img?.[0];
          return this.studentService.create(dto, image);
      }

      @Delete()
      async delete(@Body('id') id: string) {
        return this.studentService.delete(id);
      }
    }
```

student.service.ts:

```
    import { Injectable } from '@nestjs/common';
    import { PrismaService } from
'src/prisma/prisma.service';
    import { Prisma, UserSex } from '@prisma/client';
    import { CreateStudentDto } from './dto/create-
student.dto';
```

```
    import { FileService, FileType } from
'src/file/file.service';

    @Injectable()
    export class StudentService {
      constructor(private readonly prisma: PrismaService,
private readonly fileService: FileService) {}

      async findAll(page: number, limit: number, search?:
string) {
        const skip = (page - 1) * limit;

        const where: any = {};

        if (search) {
          where.OR = [
            {
              name: {
                contains: search,
                mode: 'insensitive',
              },
            },
          ];
        }

        const [students, count] = await
this.prisma.$transaction([
          this.prisma.student.findMany({
            where,
            include: {
              class: true,
              attendances: true,
              results: true,
            },
            skip,
            take: limit,
          }),
          this.prisma.student.count({ where }),
        ]);

        return { students, count };
      }

      async getAll() {
```

```
        const [students] = await
this.prisma.$transaction([
          this.prisma.student.findMany(),
        ]);

        return students;
      }

      public async create(dto: CreateStudentDto, image?:
Express.Multer.File) {
          let img = {url: ''};
          if(image) img = await
this.fileService.uploadToCloudinary(image, FileType.IMAGE);

          const student = await
this.prisma.student.create({
            data: {
              id: "student" + Number(Math.random() *
1000),
              name: dto.name,
              username: dto.username,
              surname: dto.surname,
              email: dto.email || null,
              phone: dto.phone || null,
              address: dto.address,
              img: img.url || null,
              bloodType: dto.bloodType,
              sex: dto.sex === 'MALE' ? UserSex.MALE :
UserSex.FEMALE,
              birthday: dto.birthday,
              gradeId: Number(dto.gradeId),
              classId: Number(dto.classId),
              parentId: dto.parentId,
            },
          });

          return student;
        }

        public async delete(id: string) {
          return this.prisma.student.delete({
            where: { id },
          });
        }
```

```
        public async getOne(id: string) {
          const [teachers] = await
this.prisma.$transaction([
            this.prisma.student.findMany({
              where: {id},
              include: {
                class: true
              }
            }),
          ]);

          return teachers;
        }
    }
```

class.module.ts:

```
    import { Module } from '@nestjs/common';
    import { ClassService } from './class.service';
    import { ClassController } from './class.controller';


    @Module({
      controllers: [ClassController],
      providers: [ClassService],
    })
    export class ClassModule {}
```

class.controller.ts:

```
    import { Body, Controller, Delete, Get, Post, Put,
Query } from '@nestjs/common';
    import { ClassService } from './class.service';
    import { CreateClassDto } from './dto/class-
create.dto';
    import { UpdateClassDto } from './dto/class-
update.dto';

    @Controller('classes')
    export class ClassController {
```

```
      constructor(private readonly classService:
ClassService) {}

      @Get()
      async getAllClasses(
        @Query('page') page = '1',
        @Query('limit') limit = '10',
        @Query('search') search: string,
      ) {
        return this.classService.findAll(+page, +limit,
search);
      }

      @Get('/all')
      async getAll() {
        return this.classService.getAll();
      }

      @Post()
        async create(
          @Body() dto: CreateClassDto
        ) {
          return this.classService.create(dto);
        }

        @Put()
        async update(@Body() dto: UpdateClassDto) {
          return this.classService.update(dto);
        }

        @Delete()
        async delete(@Body('id') id: number) {
          return this.classService.delete(id);
        }


        @Get('/select')
        async getSelectClass() {
          return this.classService.getSelectClasses();
        }
    }
```

class.service.ts:

```
    import { Injectable } from '@nestjs/common';
    import { PrismaService } from
'src/prisma/prisma.service';
    import { CreateClassDto } from './dto/class-
create.dto';
    import { UpdateClassDto } from './dto/class-
update.dto';

    @Injectable()
    export class ClassService {
      constructor(private readonly prisma: PrismaService)
{}

      async findAll(page: number, limit: number, search?:
string) {
        const skip = (page - 1) * limit;

        const where: any = {};

        if (search) {
          where.OR = [
            {
              name: {
                contains: search,
                mode: 'insensitive',
              },
            },
          ];
        }

        const [classes, count] = await
this.prisma.$transaction([
          this.prisma.class.findMany({
            where,
            include: {
              announcements: true,
              events: true,
              lessons: true,
              students: true,
              supervisor: true,
            },
            skip,
```

```
      take: limit,
    }),
    this.prisma.class.count({ where }),
  ]);

  return { classes, count };
}

async getAll() {
  const [classes] = await this.prisma.$transaction([
    this.prisma.class.findMany(),
  ]);

  return classes;
}

public async create(dto: CreateClassDto) {
  const classes = await this.prisma.class.create({
    data: {
      ...dto
    },
  });

  return classes;
}

public async update(dto: UpdateClassDto) {
  return this.prisma.class.update({
    where: { id: dto.id },
    data: {
      name: dto.name,
      capacity: dto.capacity,
      supervisorId: dto.supervisorId,
      gradeId: dto.gradeId
    },
  });
}

public async delete(id: number) {
  return this.prisma.class.delete({
    where: { id },
  });
}
```

```
    async getSelectClasses() {
      const [classes] = await this.prisma.$transaction([
        this.prisma.class.findMany({
          include: { _count: { select: { students: true
} } },
        })
      ]);


      return classes;
    }
  }
```

subject.module.ts:

```
  import { Module } from '@nestjs/common';
  import { SubjectService } from './subject.service';
  import { SubjectController } from
'./subject.controller';


  @Module({
    controllers: [SubjectController],
    providers: [SubjectService],
  })
  export class SubjectModule {}
```

subject.controller.ts:

```
  import { Body, Controller, Delete, Get, Post, Put,
Query } from '@nestjs/common';
  import { SubjectService } from './subject.service';
  import { CreateSubjectDto } from './dto/subject-
create.dto';
  import { UpdateSubjectDto } from './dto/subject-
update.dto';


  @Controller('subject')
  export class SubjectController {
```

```
      constructor(private readonly subjectService:
SubjectService) {}

      @Get()
      async getAllSubjects(
        @Query('page') page = '1',
        @Query('limit') limit = '10',
        @Query('search') search: string,
      ) {
        return this.subjectService.findAll(+page, +limit,
search);
      }

      @Get('/all')
      async getAll() {
        return this.subjectService.getAll();
      }

      @Post()
      async create(
        @Body() dto: CreateSubjectDto
      ) {
        return this.subjectService.create(dto);
      }

      @Put()
      async update(@Body() dto: UpdateSubjectDto) {
        return this.subjectService.update(dto);
      }

      @Delete()
      async delete(@Body('id') id: number) {
        return this.subjectService.delete(id);
      }

      @Get('/select')
      async getSelectSubject() {
        return this.subjectService.getSelectSubject();
      }
    }
```

subject.service.ts:

```
import { Injectable } from '@nestjs/common';
import { PrismaService } from 'src/prisma/prisma.service';
import { CreateSubjectDto } from './dto/subject-create.dto';
import { UpdateSubjectDto } from './dto/subject-update.dto';

@Injectable()
export class SubjectService {
    constructor(private readonly prisma: PrismaService) {}

    async findAll(page: number, limit: number, search?: string) {
        const skip = (page - 1) * limit;
        const where: any = {};

        if (search) {
        where.OR = [
            {
            name: {
                contains: search,
                mode: 'insensitive',
            },
            },
        ];
        }

        const [subjects, count] = await this.prisma.$transaction([
            this.prisma.subject.findMany({
                where,
                include: {
                    lessons: true,
                    teachers: true,
                },
                skip,
                take: limit,
            }),
            this.prisma.subject.count({ where }),
```

```
            ]);

            return { subjects, count };
        }

        async getAll() {
            const [subjects] = await
this.prisma.$transaction([
                this.prisma.subject.findMany(),
            ]);

            return subjects;
        }

        public async create(dto: CreateSubjectDto) {
            const teachers = await
this.prisma.teacher.findMany({
                    where: {
                        id: {
                            in: dto.teachers,
                        },
                    },
                    select: { id: true },
            });

            const subject = await
this.prisma.subject.create({
                data: {
                    name: dto.name,
                    teachers: {
                        connect: teachers.map((teacher) =>
({ id: teacher.id })),
                    },
                },
            });

            return subject;
      }

        public async update(dto: UpdateSubjectDto) {
            const teachers = await
this.prisma.teacher.findMany({
                where: {
                    id: {
```

```
                        in: dto.teachers,
                    },
                },
                select: { id: true },
            });

            if (teachers.length === 0) {
                throw new Error('Учителя не найдены');
            }

            return this.prisma.subject.update({
                where: { id: dto.id },
                data: {
                    name: dto.name,
                    teachers: {
                        connect: teachers.map((teacher) =>
({ id: teacher.id })),
                    },
                },
            });
        }

        public async delete(id: number) {
            return this.prisma.subject.delete({
                where: { id },
            });
        }

        async getSelectSubject() {
            const [subjects] = await
this.prisma.$transaction([
                this.prisma.subject.findMany({
                    select: { id: true, name: true },
                }),
            ]);


            return subjects;
        }
    }
```