МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБПОРТАЛ ШКОЛИ З ІНТЕГРОВАНИМИ ЗАСОБАМИ НАВЧАННЯ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Іван КОГУТ**

«__» _____ 2026 р.

Керівник роботи

старша викладачка _____ **Марина ФАЛЕНКОВА**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Когут Іван

1. Тема кваліфікаційної роботи «Вебпортал для школи з інтегрованими засобами навчання» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат є створена веборієнтована інформаційна система для автоматизації діяльності навчальних закладів, яка поєднує функції публічного представництва школи та внутрішнього управління навчальним процесом.

4. Перелік питань:

– провести аналіз предметної області та існуючих аналогів систем управління навчанням;

– обрати технічний стек та архітектурний підхід;

– спроєктувати структуру бази даних, яка передбачає розділення на центральну базу та окремі бази даних для кожної школи;

– реалізувати механізм автоматичної реєстрації, що включає створення субдомену, генерацію бази даних та виконання міграцій таблиць;

– розробити публічну частину для презентації платформи та публічного сайту для кожної окремої школи;

– розробити закриту частину з функціоналом для ролей: адміністратор, вчитель, учень, батьки;

– забезпечити захист даних та запобігання несанкціонованому доступу між різними школами.

5. Перелік графічних матеріалів: презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «29» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебпортал для школи з інтегрованими засобами навчання**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження технічного завдання на розробку вебпорталу для шкіл	26.12.2025	29.12.2025	*Виконано*
2.	Огляд літератури та аналіз існуючих систем управління навчанням	02.01.2026	19.01.2026	*Виконано*
3.	Складання календарного плану КБР	20.01.2026	21.01.2026	*Виконано*
4.	Визначення ролей користувачів, потоків даних та вимог до ізоляції шкіл	22.01.2026	10.02.2026	*Виконано*
5.	Проєктування схеми бази даних(центральної та клієнтської) та вибір технічного стеку	11.02.2026	05.03.2026	*Виконано*
6.	Моделювання та конструювання ядра системи: налаштування маршрутизації субдоменів та механізму автоматичного створення БД	06.03.2026	25.03.2026	*Виконано*
7.	Кодування функціоналу, розробка Landing Page, тестування безпеки	26.03.2026	15.04.2026	*Виконано*
8.	Отримання відгуку керівника КБР	16.04.2026	18.04.2026	*Виконано*
9.	Оформлення КБР та презентації	19.04.2026	15.05.2026	*Виконано*
10.	Попередній захист	22.06.2026	22.06.2026	*Виконано*
11.	Завершення оформлення КБР та презентації	01.06.2026	09.06.2026	*Виконано*
12.	Рецензування			*Виконано*
13.	Захист кваліфікаційної роботи	22.06.2026	23.06.2026	*Виконано*

Здобувач _____ **Іван КОГУТ**

«__» _____ 2026 р.

Керівник роботи

старша викладачка _____ **Марина ФАЛЕНКОВА**

«__» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

Вебпортал школи з інтегрованими засобами навчання

Здобувач 409 гр.: Когут Іван

Керівник: ст. викладачка Фаленкова Марина

Актуальність роботи зумовлена необхідністю цифровізації освітнього процесу та створення єдиного інформаційного середовища для навчальних закладів.

Мета роботи полягає у розробці та впровадженні інформаційної системи для забезпечення зручної взаємодії між адміністрацією, вчителями, учнями та батьками.

Об’єктом дослідження є процес автоматизації управління шкільними процесами.

Предметом дослідження є методи, моделі та програмні засоби створення мультитенантних систем для автоматизації освітніх процесів.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність обраної теми, визначено мету, основні завдання, об'єкт і предмет дослідження, а також розкрито практичне значення одержаних результатів.

У першому розділі проведено аналіз предметної області та дослідження існуючих аналогів систем управління навчанням, виявлено їхні переваги та недоліки, що дозволило обґрунтувати необхідність створення інтегрованої екосистеми.

У другому розділі здійснено науково-технічне дослідження сучасного стану інструментарію та методів розробки SaaS-систем на основі аналізу фахових публікацій. Сформовано повну специфікацію вимог до програмного забезпечення.

У третьому розділі спроєктовано архітектуру системи з використанням принципу ізоляції баз даних для кожної школи. Побудовано UML-моделі (діаграми

класів, розгортання, послідовності) та розроблено макети інтерфейсу користувача для різних ролей.

У четвертому розділі описано програмну реалізацію порталу на базі стеку Laravel, Vue.js та MS SQL Server. Наведено результати тестування модулів розкладу, журналу та механізму автоматичної реєстрації шкіл.

У висновках узагальнено результати виконаної роботи, підтверджено виконання поставлених на початку завдань та визначено шляхи подальшого розширення функціоналу створеного вебзастосунку.

Кваліфікаційна робота викладена на 87 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 28 найменувань та 3 додатків. Праця містить 10 таблиць та 57 рисунків.

***Ключові слова**: шкільний портал, вебзастосунок, Laravel, Vue.js, мульти-тенантна архітектура, електронний журнал, бази даних, автоматизація освіти.*

ABSTRACT

to the qualifying bachelor's thesis

School web portal with integrated learnings tools

Student of 409 group: Kohut Ivan

Supervisor: Senior Lecturer Falenkova Maryna

The relevance of the work is determined by the need to digitize the educational process and create a unified information environment for educational institutions.

The purpose of the work is to develop and implement an information system to ensure convenient interaction among administration, teachers, students, and parents.

The object of the study is the process of automating school management.

The subject of the study encompasses methods, models, and software tools for creating multi-tenant systems for automating educational processes.

The qualification work consists of an introduction, 4 chapters, conclusions, and a list of references.

In the introduction, the relevance of the chosen topic is justified, the goal, main tasks, object, and subject of the study are defined, and the practical significance of the obtained results is disclosed.

The first sections provide an analysis of the subject area and a study of existing learning management system analogs, identifying their advantages and disadvantages, which allowed for substantiating the necessity of creating an integrated ecosystem.

The second sections carry out a scientific and technical study of the modern state of tools and methods for developing SaaS systems based on the analysis of professional publications. A complete Software Requirements Specification has been formulated.

The third sections design the system architecture using the principle of database isolation for each school. UML models (class, deployment, sequence diagrams) have been constructed, and user interface mockups have been developed for various roles.…

The fourth sections describe the software implementation of the portal based on the Laravel, Vue.js, and MS SQL Server stack. The results of testing the schedule, electronic gradebook, and automatic school registration modules are presented.

The conclusions summarize the results of the work performed, confirm the fulfillment of the tasks set at the beginning, and identify ways for further expanding the functionality of the created web application.

The qualification work is presented on 87 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 28 titles and 3 appendices. The work contains 10 tables and 57 figures.

Keywords: school portal, web application, Laravel, Vue.js, multi-tenant architecture, electronic gradebook, databases, education automation.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ACID – Atomicity, Consistency, Isolation, Durability

API – Application Programming Interface

AWS – Amazon Web Services

CSRF – Cross-Site Request Forgery

GDPR – General Data Protection Regulation

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

MVC – Model-View-Controller

ORM – Object-Relational Mapping

SaaS – Software as a Service

SPA – Single-Page Application

ВСТУП

Сьогодні більшість закладів освіти активно впроваджують цифрові технології для організації навчального процесу. Окрім звичайного вебсайту, школам необхідні інструменти для ведення електронного журналу, управління розкладом занять, обміну інформацією між учителями, учнями та батьками. Тому виникає потреба у створенні єдиного вебпорталу, який об'єднує всі ці можливості в межах однієї системи.

На сьогодні існує багато сервісів для автоматизації освітнього процесу. Проте більшість із них вирішують лише окремі завдання: ведення електронного журналу, організацію дистанційного навчання або публікацію новин школи. Через це заклади освіти змушені використовувати декілька різних платформ одночасно, що ускладнює роботу користувачів. Використання архітектури мультитенантності дозволяє подолати ці технологічні бар'єри, надаючи програмне забезпечення як послугу. Такий підхід забезпечує можливість розгортання ізольованого середовища для нового навчального закладу за хвилини, гарантуючи при цьому безпеку даних.

Метою роботи розробка веборієнтованої інформаційної системи для автоматизації навчального процесу з використанням архітектури Multi-tenancy для підвищення прозорості та ефективності взаємодії між адміністрацією, вчителями, учнями та батьками.

Завдання роботи:

– провести аналіз предметної області та існуючих аналогів систем управління навчанням;

– обрати технічний стек та архітектурний підхід;

– спроєктувати структуру бази даних, яка передбачає розділення на центральну базу та окремі бази даних для кожної школи;

– реалізувати механізм автоматичної реєстрації, що включає створення субдомену, генерацію бази даних та виконання міграцій таблиць;

– розробити публічну частину для презентації платформи та публічного сайту для кожної окремої школи;

– розробити закриту частину з функціоналом для ролей: адміністратор, вчитель, учень, батьки;

– забезпечити захист даних та запобігання несанкціонованому доступу між різними школами.

Об’єктом кваліфікаційної роботи є процес інформаційного забезпечення та управління діяльністю середніх загальноосвітніх навчальних закладів. Предметом дослідження є методи та програмні засоби створення мультитенантних систем для автоматизації освітніх процесів.

Науково-практичне значення одержаних результатів полягає в автоматизації рутинної діяльності адміністрації шкіл, забезпечити повну прозорість навчання через інтегрований електронний журнал та реєстрація нового навчального закладу, відбувається автоматично без втручання розробників. Сфера застосування результатів охоплює державні та приватні заклади загальної середньої освіти, що потребують комплексної цифровізації своєї діяльності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Останніми роками цифрові технології стали невід'ємною частиною роботи навчальних закладів. Школи все частіше використовують електронні журнали, онлайн-сервіси та вебплатформи для організації навчального процесу. Через це виникає потреба у створенні єдиної системи, яка б спрощувала роботу адміністрації, вчителів, учнів і батьків. Досвід останніх років, спричинений глобальними викликами, зокрема переходом на дистанційне та змішане навчання через пандемію та воєнні дії в Україні, став каталізатором, який виявив критичну застарілість традиційних методів роботи з паперовими носіями інформації.

Проблема полягає не лише у фізичній незручності паперового документообігу, а й у відсутності оперативності, низькому рівні прозорості та складності аналітичної обробки даних. Для того, щоб сучасна школа могла ефективно функціонувати, їй необхідний надійний цифровий простір, який би об’єднував ведення електронних журналів, динамічне управління розкладом та забезпечував миттєву комунікацію між усіма учасниками освітнього процесу. Однак існує проблема: впровадження окремих, не пов’язаних між собою програмних продуктів часто не полегшує, а навпаки, ускладнює роботу вчителя, змушуючи його витрачати час на синхронізацію даних між різними платформами.

Ефективне проєктування програмного забезпечення такого рівня вимагає аналізу внутрішньої структури закладу. Без розуміння щоденних бізнес-процесів адміністрації, специфіки взаємодії вчителя з учнем та запитів батьків неможливо побудувати архітектуру.

1.1 Розкриття об’єкта та предмета дослідження

Об’єктом кваліфікаційної роботи є процес інформаційного забезпечення та системного управління діяльністю середніх загальноосвітніх навчальних закладів. В межах дослідження цей процес розглядається не просто як набір рутинних операцій, а як складна система, що функціонує в умовах постійного інформаційного навантаження. Цей процес охоплює задачі, серед яких:

– планування педагогічного навантаження та управління кадровим потенціалом;

– динамічне формування розкладу занять із врахуванням ресурсного забезпечення;

– ведення суворого обліку академічної успішності та відвідуваності здобувачів освіти;

– забезпечення внутрішнього та зовнішнього документообігу, а також організація безперервної інформаційної взаємодії між адміністрацією, викладацьким складом, учнями та батьками.

Аналіз сучасних підходів до цифровізації шкіл показує, що багато навчальних закладів використовують окремі сервіси для різних завдань. Наприклад, одна система застосовується для дистанційного навчання, інша для ведення оцінок, а сайт школи працює окремо. Така децентралізація інфраструктури не лише спричиняє багаторазове дублювання однієї й тієї ж інформації, що збільшує когнітивне навантаження на педагогічний персонал, але й створює критичні вразливості в контексті інформаційної безпеки та захисту персональних даних. Крім того, відсутність єдиної бази даних унеможливлює проведення глибокої аналітики освітнього процесу з боку адміністрації.

Предметом дослідження є методи та програмні засоби створення мультитенантних систем для автоматизації освітніх процесів.

Вирішення проблеми фрагментації базується на відмові від традиційної моделі, яка вимагає розгортання окремого сервера для кожного нового закладу. Натомість застосовується архітектура Multi-tenancy, де програмне забезпечення як послуга (рис. 1.1). Цей підхід забезпечує використання єдиної програмної інфраструктури та спільної кодової бази, але з обов'язковою ізоляцією даних кожної школи на рівні окремих баз даних.

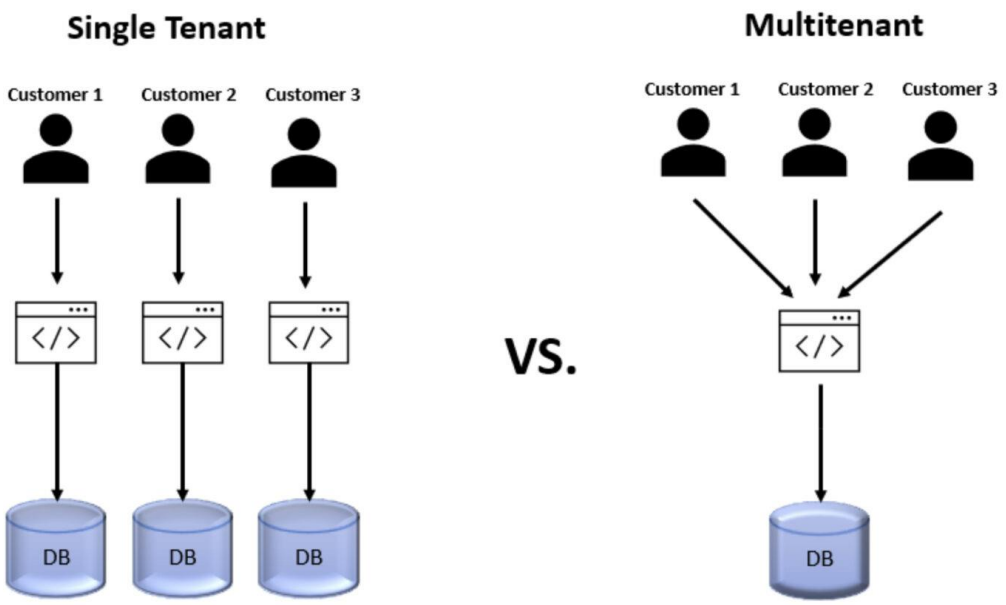

Рисунок 1.1 – Порівняльна схема

Натомість створені даного проєкту, буде використовуватися хмарна архітектура Multi-tenancy, де програмне забезпечення надається як SaaS. Цей підхід забезпечує використання єдиної централізованої програмної інфраструктури та спільної кодової бази для всіх клієнтів платформи. Однак ключовою вимогою до таких систем, особливо в освітній сфері, є суворе розмежування доступу. Тому архітектура передбачає обов'язкову логічну або фізичну ізоляцію даних кожної окремого тенанта на рівні баз даних, що унеможливлює перетин конфіденційної інформації між різними закладами.

Використання мультитенантної архітектури дозволяє обслуговувати декілька навчальних закладів в межах однієї програмної платформи. Застосування алгоритмів маршрутизації піддоменів та методів динамічного управління підключеннями до баз даних дає змогу повністю автоматизувати процес реєстрації нових навчальних закладів. Система здатна генерувати для нового клієнта абсолютно ізольоване цифрове середовище з готовими налаштуваннями за лічені хвилини, без жодного ручного втручання технічних спеціалістів чи адміністраторів сервера. Таким чином, застосування обраного архітектурного підходу не лише

оптимізує витрати на підтримку інфраструктури, але й якісно покращує підходи до управління навчальним закладом.

1.2 Опис і аналіз структурних і функціональних особливостей

Для проєктування інформаційної системи необхідно детально проаналізувати структурні та функціональні особливості. З точки зору системного аналізу, заклад загальної середньої освіти являє собою організацію, діяльність якої супроводжується інтенсивним обміном даними між різними групами користувачів. Основними структурними елементами цього об'єкта є адміністрація, педагогічний колектив, учні, а також їхні батьки. Кожен із цих суб'єктів виконує ролі, що генерують відповідні масиви інформації та потребують індивідуальних рівнів доступу до даних.

Функціональні особливості закладу визначаються наявністю трьох інформаційних потоків: адміністративного, навчального та комунікаційного. Кожен із цих напрямків має власну специфіку бізнес-процесів, які потребують глибокої автоматизації та інтеграції:

– Адміністративний вектор представляє собою рівень стратегічного та операційного планування. Сюди входить формування тарифікаційних списків, розподіл навчальних годин, моніторинг виконання навчальних планів та підготовка звітів для департаментів освіти. На практиці цей процес ініціюється директором або завучем, які формують початкову структуру навчального середовища: реєструють викладачів, створюють учнівські класи та закріплюють за ними класних керівників. Основна складність на цьому етапі полягає в генерації та підтримці динамічності загальношкільного розкладу, який має постійно враховувати навантаження педагогів та оперативні заміни вчителів у реальному часі.

– Навчальний вектор є рівнем щоденної діяльності, де вчитель виступає головним генератором контенту та даних. Проведення уроку розпочинається з відкриття електронного журналу, де система автоматично ідентифікує поточне заняття згідно з розкладом. Педагог фіксує відсутніх учнів, записує тему уроку та

виставляє поточні або тематичні оцінки. Наприкінці заняття вчитель формує домашнє завдання, маючи можливість прикріпити необхідні цифрові матеріали. Усі ці дані мають бути чітко структурованими та нормалізованими, щоб система могла автоматично генерувати їх у статистику успішності без потреби ручного дублювання звітів.

– Комунікаційний вектор формує зовнішній контур системи, який орієнтований на здобувачів освіти та батьків. Батьки потребують оперативного та зручного доступу до інформації про успішність дитини через функціонал електронного щоденника. Вони повинні мати змогу в будь-який момент переглянути поточні оцінки, зауваження викладачів та контролювати відвідуваність. Учні, зі свого боку, отримують доступ до індивідуального розкладу занять та персоналізованої стрічки домашніх завдань. Водночас для самої школи цей вектор слугує інструментом для публічного представлення своїх досягнень, публікації новин та важливих оголошень через інтегрований вебсайт.

Ключовою проблемою, яку має розв’язати система, є забезпечення єдиного та цілісного інформаційного середовища. У сучасних умовах виставлення оцінки або відмітки про відсутність вчителем має миттєво оновити дані в персональному кабінеті батьків, додати запис у базу статистики для директора та, за потреби, відобразити зміни в системі рейтингів на публічному сайті. Саме така синхронність процесів гарантує ефективність впровадження системи в реальні умови роботи закладу освіти.

1.3 Огляд та аналіз існуючих систем

Ринок програмного забезпечення для освітньої сфери пропонує значну кількість систем управління навчанням. Проте, більшість із них вирішують лише точкові завдання (наприклад, організацію тестування або навчальний процес), не утворюючи єдиної управлінської системи для навчального закладу. Для формування вимог до нової системи необхідно проаналізувати найпопулярніші на ринку аналоги.

1.3.1 Google Classroom

Однією з найпоширеніших безкоштовних платформ для організації дистанційного навчання є Google Classroom. На рисунку 1.2, зображене візуальне середовище Google Classroom базується на картковій системі. Кожна картка відображає ізольований навчальний простір (клас або предмет) і містить базову інформацію. Такий мінімалістичний підхід робить платформу інтуїтивно зрозумілою для учнів і вчителів, оскільки фокусує увагу на поточних завданнях. Система тісно інтегрована з екосистемою Google Workspace (Drive, Docs, Meet), що робить її зручною для обміну файлами та проведення відеоуроків.

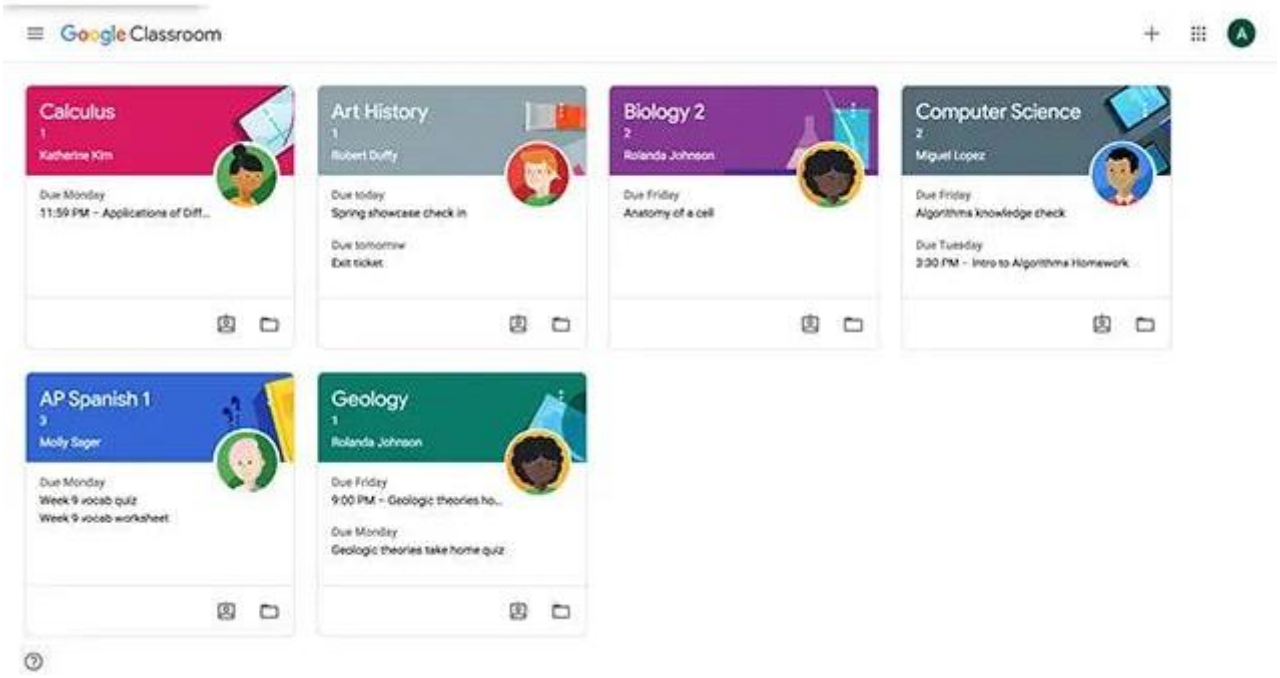

Рисунок 1.2 – Інтерфейс Google Classroom

Має переваги такі як, низький поріг входження, безкоштовний продукт, надійна хмарна інфраструктура. Але відповідно, Google Classroom є інструментом для вчителя, а не системою управління школою. Платформа не має функціоналу для ведення єдиного загальношкільного розкладу, генерації офіційних звітів успішності чи ведення електронного журналу за стандартами української системи освіти. Крім того, система не передбачає наявності інтегрованого публічного сайту-візитки закладу [1]. Детальна характеристика платформи наведені в таблиці 1.1.

Таблиця 1.1 – Характеристика та аналіз системи Google Classroom

Назва	Google Classroom
Виробник	Google
Архітектура	Хмарна мікросервісна архітектура.
Мова реалізації	Angular, Go, C++
Основні функції	Створення та управління класами, розподіл завдань, оцінки та зворотній зв’язок, email та push-сповіщення, мобільна підтримка.
Переваги	Безкоштовно для шкіл із Google Workspace for Education. Простий та інтуїтивний інтерфейс для новачків. Глибока інтеграція з екосистемою Google (Drive, Docs).
Недоліки	Залежність від Google. Відсутність повноцінного функціоналу для адміністрування школи.

Отже, незважаючи на зручність для організації безпосередньої взаємодії між учителем та учнем, Google Classroom не може розглядатися як комплексне рішення для автоматизації управління всім навчальним закладом через відсутність необхідного інструментарію для адміністрації школи

1.3.2 Schoology

Schoology – це система управління навчанням корпоративного рівня, яка широко використовується в навчальних закладах США та країн Європи. На відміну від більш простих аналогів, система дозволяє кастомізацію під конкретний навчальний заклад, проте базова структура інформаційної панелі все одно залишається орієнтованою на ізольовані навчальні дисципліни. Загальний вигляд панелі курсів у середовищі Schoology подано на рисунку 1.4. Вона пропонує гнучке управління ролями користувачів та можливість створення складних навчальних курсів із глибокою аналітикою успішності учнів, інтерфейс.

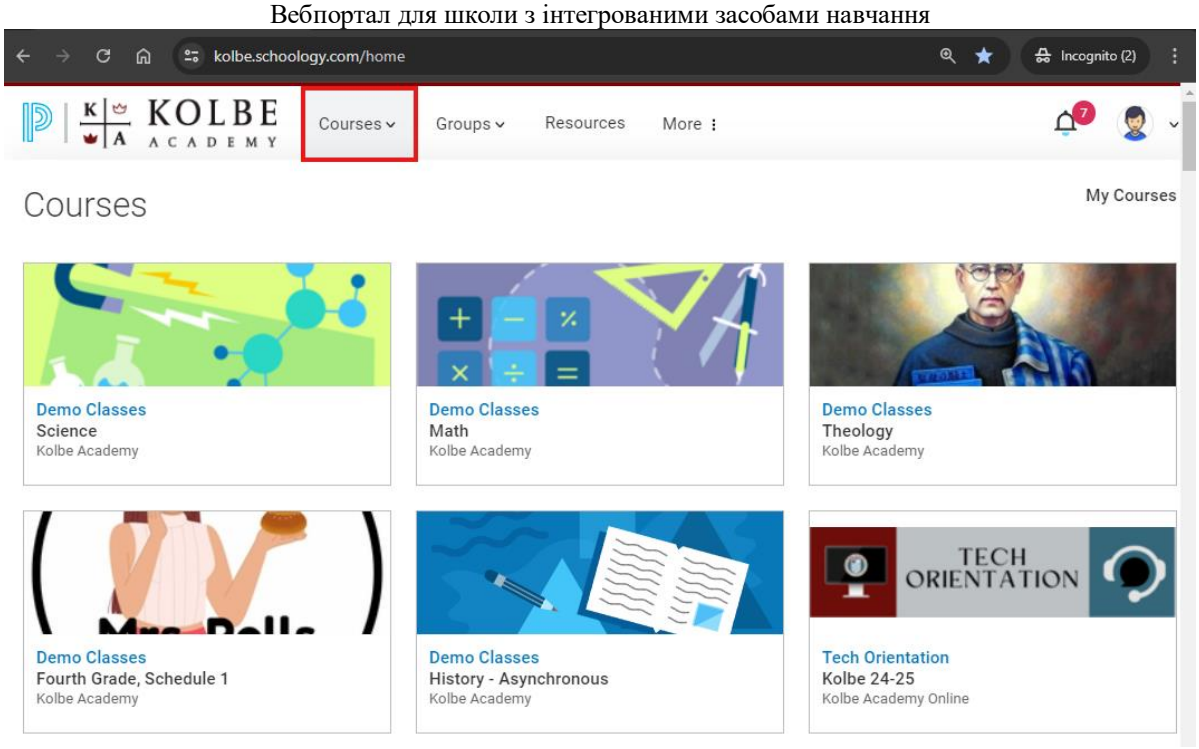

Рисунок 1.3 – Інтерфейс Schoology

Має переваги такі як, детальна статистика навчального процесу, високий рівень безпеки даних, інструменти для гейміфікації(обмежена) та надійна система тестування. Головною проблемою платформи є її орієнтація на західну модель освіти. Адаптація системи під українську 12-бальну шкалу та структуру навчального року (чверті, семестри) є дуже складною. Крім того, Schoology є комерційним продуктом із високою вартістю ліцензування для державних шкіл [2]. Зведені технічні та функціональні характеристики системи Schoology наведено в таблиці 1.2.

Таблиця 1.2 – Характеристика та аналіз системи Schoology

Назва	Schoology
Виробник	PowerSchool

Кінець таблиці 1.2

Архітектура	Клієнт-серверна архітектура, з хмарними сервісами.
Мова реалізації	Angular, PHP, Java
Основні функції	Ролі та права доступу, розклад і календар, домашнє завдання і оцінки, модуль для публічних сторінок, можливості для комунікації(групи за інтересами, стрічка новин).
Переваги	Гнучкість ролей, інтуїтивно зрозумілий інтерфейс(схожий на Facebook), звіти про успішність, адаптивність для всіх пристроїв.
Недоліки	Платна модель, складність налаштування для маленьких шкіл, логіка системи орієнтовна на американську систему.

Таким чином, Schoology демонструє високий рівень функціональності та безпеки, проте її адаптація під вітчизняні освітні стандарти потребує значних зусиль, що робить її малопридатною для масового впровадження в українських школах.

1.3.3 Prosvita(Просвіта)

Просвіта є офіційною державною платформою, що робить її надійним інструментом для українських шкіл. Оскільки вона розроблена спеціально під українську систему освіти, всі функції, електронний журнал, розклад, оцінювання, відповідають державним стандартам і нормативам. Це значно спрощує роботу вчителів і адміністрації, адже не потрібно адаптувати іноземні платформи під місцеві вимоги.

Архітектура інтерфейсу системи Просвіта, яка зображена на рисунку 1.4, докорінно відрізняється від західних аналогів. Бічна навігаційна панель є значно ширшою і містить інструменти для комплексного адміністрування школи: керування розкладом, модулі для взаємодії з батьками, формування звітів та контроль відвідуваності. Але, табличний підхід із надмірною щільністю інформації

та елементів управління (кнопки додавання колонок, відеодзвінків тощо) робить інтерфейс візуально важким. Необхідність постійного горизонтального та вертикального прокручування великих масивів даних значно підвищує когнітивне навантаження на педагогічний персонал [3].

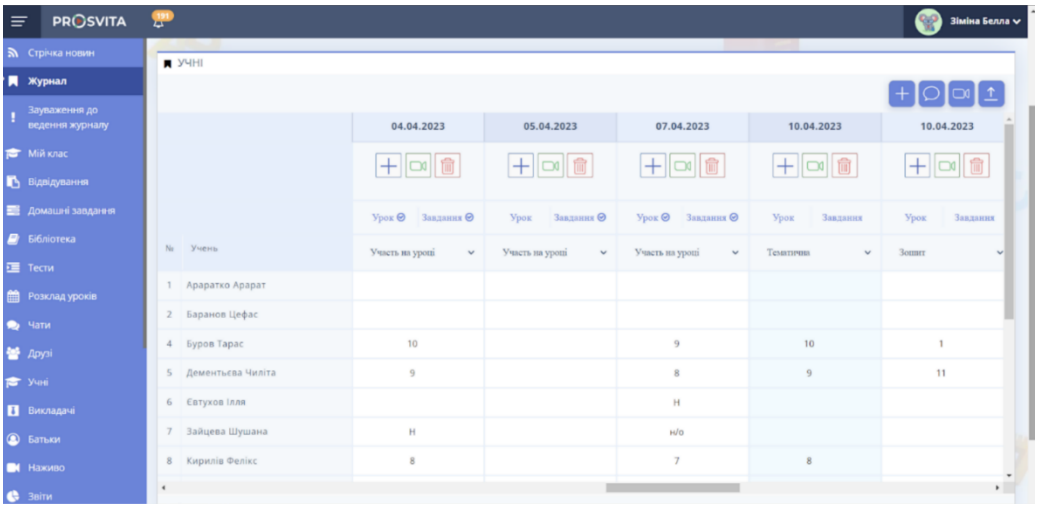

Рисунок 1.4 – Інтерфейс Prosvita

Система має функціональні обмеження, що перешкоджають створенню повноцінної освітньої екосистеми. Застарілий інтерфейс користувача ускладнює адаптацію педагогічного персоналу. Також, присутня обмеженість аналітичного апарату. Система надає переважно базову статистику успішності та відвідуваності, не маючи засобів глибокої аналітики для потреб адміністрації закладу. Технічні особливості, ключові переваги та недоліки платформи систематизовано в таблиці 1.3.

Таблиця 1.3 – Характеристика та аналіз системи Prosvita

Назва	Prosvita
Виробник	ТОВ «НОВІ ОСВІТНІ ТЕХНОЛОГІЇ».
Архітектура	Клієнт-серверна архітектура, з хмарними сервісами

Кінець таблиці 1.3

Мова реалізації	React, PHP/Java
Основні функції	Ролі та права доступу, цифровий календар та розклад, із фільтрами, домашнє завдання та оцінки, система сповіщень, чати для спілкування між учнями, вчителями та батьками
Переваги	Унікальна система мотивації «Вчись-Грай-Заробляй», де учні отримують віртуальну валюту (кристали) за оцінки та активність, яку можна витрачати на аватари чи реальні товари в магазині школи; повна безкоштовність для державних закладів; сучасний дизайн.
Недоліки	Гейміфікація може відволікати учнів від навчання; складна система налаштування для адміністратора на початку роботи.

Система Просвіта є найбільш адаптованою до українських нормативних вимог, однак перевантажений інтерфейс створюють бар'єри для ефективного щоденного використання. Це підтверджує необхідність розробки більш сучасної, оптимізованої та швидкої альтернативи.

Висновки до розділу 1

У межах першого розділу було проаналізовано особливості роботи сучасних закладів освіти та існуючі програмні рішення для автоматизації навчального процесу. Аналіз показав, що використання декількох незалежних сервісів створює труднощі в роботі адміністрації, учителів та батьків. Необхідність постійної ручної синхронізації інформації між різними платформами суттєво перевантажує педагогічний та адміністративний персонал, збільшує когнітивне навантаження на вчителів та знижує загальну ефективність роботи закладу.

Аналіз існуючих систем управління навчанням показав, що жоден із популярних аналогів не задовольняє повною мірою потреби вітчизняних шкіл.

Зокрема, Google Classroom є зручним інструментом для організації локального навчального процесу, але зовсім не пристосований для комплексного управління школою. Платформа Schoology, попри потужний функціонал, орієнтована на західну освітню модель. Вітчизняна система Просвіта, хоча й відповідає українським нормативам, має перевантажений застарілий інтерфейс та обмежені аналітичні можливості, що ускладнює її щоденне використання

З огляду на це, обґрунтовано потребу у розробці веборієнтованої інформаційної системи з використанням архітектури Multi-tenancy, яка усуне виявлені недоліки. Застосування такого підходу з ізоляцією даних на рівні окремих баз гарантує високий рівень безпеки інформації кожного навчального закладу при використанні спільної кодової бази. Створення єдиного цифрового простору для адміністрації, вчителів, учнів та батьків забезпечить прозорість освітнього процесу та суттєво підвищить загальну ефективність управління школою.

2 НАУКОВО-МЕТОДИЧНИЙ АНАЛІЗ

2.1 Аналіз сучасного стану моделей, методів та інструментарію розробки SaaS-платформ

Поширення дистанційного та змішаного навчання сприяло активному розвитку освітніх вебплатформ. Такі системи повинні забезпечувати стабільну роботу, захист даних користувачів та можливість обслуговування великої кількості навчальних закладів. Сучасний вебпорталу школи з інтегрованими засобами навчання більше не є просто інформаційним ресурсом або статичною візиткою навчального закладу. Це складна система, яка охоплює системи управління навчанням, інструменти, електронні журнали, розклади та модулі інтеграції зі сторонніми сервісами [4]. Впровадження моделі програмного забезпечення як послуги стало галузевим стандартом для розгортання таких комплексних платформ. Використання SaaS-моделі дозволяє централізовано управляти оновленнями програмного забезпечення, оптимізувати витрати на інфраструктуру та забезпечувати високу доступність сервісів. Однак розгортання платформи, яка обслуговує сотні незалежних навчальних закладів, ставить перед розробниками низку викликів щодо архітектури зберігання даних, інформаційної безпеки та вибору технологічного стека [5, 6, 7].

Одним із найпоширеніших підходів до побудови SaaS-систем є використання мультитенантної архітектури. Вона дозволяє обслуговувати декілька незалежних клієнтів за допомогою єдиної програмної платформи та спільної кодової бази. Такий підхід зменшує витрати на підтримку системи та спрощує її подальше масштабування. Дослідження у сфері хмарних обчислень виділяють три фундаментальні підходи до організації даних у мультитенантних системах [8]:

– спільна база даних та спільна схема. У моделі де спільна база даних та спільна схема (Shared Database, Shared Schema), всі тенанти зберігають свої дані в одній фізичній базі даних та в одних і тих самих таблицях. Логічна ізоляція забезпечується виключно на рівні рядків шляхом додавання ідентифікатора, наприклад, стовпця tenant_id, до кожної таблиці системи. Будь-який запит до бази

даних повинен містити умову WHERE tenant_id = X. Перевагою цього підходу є простота реалізації та ефективне використання ресурсів сервера. Проте ізоляція даних забезпечується виключно програмними механізмами. Через це будь-яка помилка у логіці застосунку може призвести до доступу користувачів до інформації інших тенантів. Крім того, виникає гостра проблема «галасливого сусіда», коли ресурсоємні запити від одного великого тенанта монополізують ресурси всієї бази даних, суттєво сповільнюючи роботу системи для всіх інших шкіл [9];

– спільна база даних, окремі схеми (Shared Database, Separate Schemas): У цьому підході дані всіх тенантів також зберігаються на одному сервері баз даних, але кожен тенант отримує власну логічну схему. Наприклад, таблиця користувачів першої школи матиме адресу tenant_1.users, а другої – tenant_2.users. Використання окремих схем підвищує рівень ізоляції даних порівняно зі спільною схемою. Дані кожного тенанта зберігаються окремо, що зменшує ризик випадкового доступу до інформації інших клієнтів. Водночас усі схеми залишаються в межах одного сервера баз даних, тому проблема спільного використання ресурсів повністю не зникає. Крім того, при досягненні масштабу в тисячі тенантів, виконання міграцій перетворюється на надзвичайно складний і тривалий процес [10, 11];

– окрема база даних для кожного тенанта (Database-per-Tenant): Цей підхід, передбачає створення повністю ізольованої, фізично або логічно виділеної бази даних для кожного клієнта. В освітніх системах, де конфіденційність даних учнів є критично важливою вимогою.[12].

2.2 Переваги моделі Database-per-Tenant

Перевага даної моделі для розробки, спирається на низку стратегічних та інженерних переваг, які повністю нівелюють додаткові інфраструктурні витрати:

– Використання окремих баз даних зменшує ризик доступу до даних інших навчальних закладів навіть у випадку помилок у програмному коді. Будь-яка вразливість на рівні програмного коду (наприклад, SQL-ін'єкція або помилка авторизації) матиме обмежений радіус ураження – зловмисник фізично не зможе

отримати доступ до даних інших навчальних закладів, оскільки підключення до їхніх баз не ініційоване в поточному контексті застосунку.

– В освітньому процесі часто трапляються помилки, пов'язані з людським фактором (наприклад, адміністратор школи випадково видаляє всю інформацію про класи або оцінки). При використанні поточної моделі відновлення з резервної копії, здійснюється лише для бази даних конкретної школи, не впливаючи на доступність та цілісність даних інших тенантів порталу. У випадку спільної бази даних, точкове відновлення записів одного тенанта з бекапу без зупинки системи та блокування інших користувачів є надзвичайно складним і ризикованим.

– Коли кожна школа має власну базу даних, апаратні ресурси можуть бути чітко розподілені, обмежені або масштабовані незалежно. Якщо одна велика гімназія генерує величезне навантаження під час проведення масового онлайн-тестування, її базу даних можна ізольовано перенести на більш потужний сервер або кластер, абсолютно не зачіпаючи бази даних менших шкіл, які можуть спільно розміщуватися на дешевшому обладнанні [13].

Проведений аналіз показав, що для освітньої системи, де зберігаються персональні дані учнів та працівників школи, доцільно використовувати модель Database-per-Tenant. Такий підхід забезпечує кращу ізоляцію даних і знижує ризик доступу до інформації сторонніх користувачів.

2.3 Безпека даних у хмарних освітніх системах

Оскільки освітні системи працюють із персональними даними учнів, батьків та працівників школи, питання інформаційної безпеки є одним із ключових під час розробки програмного забезпечення. Зберігання журналів успішності, профілів, домашніх адрес та контактних даних учнів, робить освітні платформи надзвичайно привабливою та вразливою мішенню для кібератак, соціальної інженерії та викрадення особистостей [14].

Глобальні нормативно-правові акти, такі як GDPR в Європейському Союзі, FERPA у США або локальні закони про захист персональних даних в Україні,

вимагають від розробників SaaS дотримання жорстких політик щодо збору, обробки та зберігання інформації [15].

Під час проєктування системи доцільно враховувати сучасні підходи до захисту персональних даних, які використовуються в міжнародній практиці:

– механізми захисту даних повинні бути нерозривно вплетені в архітектуру системи. У контексті розробки це означає, що налаштування приватності (наприклад, видимість профілю учня) мають бути максимально закритими від моменту створення акаунта. З інженерної точки зору, використання моделі Database-per-Tenant є прямим втіленням концепції Privacy by Design, оскільки ізоляція даних відбувається на фундаментальному структурному рівні, а не покладається на програмну логіку додатку [15];

– система повинна обробляти та зберігати виключно той мінімальний обсяг персональних даних, який є критично необхідним для надання освітніх послуг. Це передбачає розробку оптимізованих структур баз даних без надлишкових полів та реалізацію автоматизованих процесів знищення застарілої інформації (наприклад, видалення профілю після випуску учня) [16];

– хмарні бази даних та застосунки повинні використовувати надійні методи шифрування (наприклад, AES-256) для даних та протоколи TLS 1.3 для передачі даних мережею. Обов'язковою є реалізація методів криптографічного хешування (наприклад, алгоритми bcrypt або Argon2 для зберігання паролів) [16].

2.4 Практичні механізми забезпечення безпеки у хмарних базах даних

Під час використання хмарної інфраструктури відповідальність за безпеку системи розподіляється між постачальником хмарних послуг та розробниками програмного забезпечення. Провайдер забезпечує захист фізичної інфраструктури, тоді як налаштування доступу, автентифікація користувачів та захист даних залишаються відповідальністю розробників системи [17].

Для надійного захисту кожної ізольованої бази даних у шкільному порталі застосовуються наступні інженерні практики:

– Динамічне управління доступом на основі ролей є критичним компонентом. Користувачі поділяються на чіткі категорії (учень, батько, вчитель, адміністратор школи). Кожен HTTP-запит до back-end має проходити сувору верифікацію Middleware, де перевіряється не лише ідентифікатор користувача, але й його належність до поточного контексту тенанта. Вчитель школи «А» фізично не може отримати токен доступу, дійсний для ресурсів школи «Б», оскільки його запит буде відхилено на рівні визначення тенанта [18].

– Відповідно до вимог GDPR щодо забезпечення прозорості, усі події доступу, модифікації або видалення конфіденційної інформації повинні безперервно фіксуватися в спеціалізованих захищених журналах аудиту. Логування має бути структурованим (наприклад, у форматі JSON) і обов'язково містити ідентифікатор тенанта. Це дозволить адміністраторам платформи оперативно розслідувати інциденти та відповідати вимогам регуляторів щодо термінової звітності у випадку виявлення витоку даних [19].

2.5 Сучасні вебархітектури

Технологічний стек, обраний для розробки користувацького інтерфейсу та back-end, впливає не лише на швидкість та легкість розробки продукту, але й на фінальну продуктивність системи, її підтримку та користувацький досвід. Впродовж останнього десятиліття індустрія веброзробки пройшла етап бурхливої еволюції, змінюючи парадигми від монолітних додатків до повністю розподілених мікросервісних рішень [20].

Історично вебдодатки будувалися за класичним патерном MVC з повним рендерингом на стороні сервера. У цій парадигмі (наприклад, класичний Laravel з використанням шаблонізатора Blade) браузер користувача надсилає запит, сервер обробляє його, звертається до бази даних і повертає повністю сформовану HTML-сторінку. При кожному переході за посиланням процес повторюється, викликаючи перезавантаження сторінки. Це забезпечує чудову пошукову оптимізацію та швидкий час першого відображення, але робить застосунок менш інтерактивним і

незручним для інтенсивного введення даних, що критично для складних дашбордів або систем оцінювання в освітньому порталі.

Для розв'язання проблеми інтерактивності індустрія перейшла до архітектури односторінкових застосунків. У цьому випадку сервер (наприклад, Node.js або Laravel) виступає виключно як RESTful повертаючи дані у форматі JSON. Вся відповідальність за відображення, маршрутизацію та обробку стану лягає на потужні фронтенд-фреймворки на базі JavaScript, такі як React, Angular або Vue.js [21].

Однак розробка повністю розділеного SPA має певні складнощі:

– розробникам доводиться описувати правила валідації даних, маршрутизацію та перевірку прав доступу як на клієнті (наприклад, у Vue Router), так і на сервері [22];

– на клієнті потрібно розгортати і підтримувати складні менеджери глобального стану (Vuex, Redux, Pinia) для синхронізації отриманих через API даних з інтерфейсом [23];

– замість надійних, перевірених часом вбудованих захищених HTTP-only сесій (Session-based Auth) із вбудованими CSRF-токенами, доводиться реалізовувати складні механізми на основі JWT-токенів [24].

2.6 Концепція «сучасний моноліт»

Для розробки освітнього порталу школи обрано інноваційну архітектуру, яка елегантно поєднує найкращі риси класичного серверного MVC та клієнтського SPA за допомогою інструменту Inertia.js. Цей підхід в сучасній інженерії часто називають Сучасним монолітом. Вона дозволяє розробникам зберігати монолітну структуру коду, використовуючи при цьому сучасні реактивні інтерфейси [25].

Laravel виступає як надзвичайно надійний, багатофункціональний back-end фреймворк, що відповідає за всі серверні процеси: підключення та взаємодію з базою даних через Eloquent ORM, міграції, обробку складної бізнес-логіки та забезпечення безпеки. MVC-архітектура Laravel з її багатою екосистемою ідеально підходить для побудови надійних корпоративних та освітніх систем. Особливо

важливою є гнучкість Laravel у контексті мультитенантності: back-end може легко перехоплювати запити на рівні Middleware, визначати піддомен школи та перемикати підключення до відповідної бази даних, залишаючи бізнес-логіку контролерів абсолютно незмінною [26].

Vue.js інтегрується як прогресивний JavaScript-фреймворк для створення інтерактивних, реактивних та модульних компонентів користувацького інтерфейсу. Використання концепції віртуального DOM у Vue.js забезпечує блискавичне оновлення лише тих частин сторінки, які дійсно змінилися внаслідок дій користувача (наприклад, додавання оцінки в електронний журнал або відправка повідомлення в чат). Це гарантує неймовірно плавний користувацький досвід без перевантаження сторінки [27].

Inertia.js виконує роль мосту, який з'єднує Laravel та Vue.js. Замість створення та підтримки окремого RESTful API, використовувати класичні контролери та маршрути Laravel.

Принцип роботи протоколу Inertia.js виглядає наступним чином:

1. при першому переході користувача на сайт (HTTP GET запит) Inertia генерує та повертає повну HTML-сторінку. Цей початковий HTML містить кореневий та data-атрибут із серіалізованими JSON-даними, необхідними для рендерингу Vue-компонента;
2. коли користувач натискає по посиланнях всередині додатку, Inertia перехоплює ці кліки і робить асинхронні запити до сервера додаючи спеціальний заголовок X-Inertia: true;
3. контролер Laravel, розпізнаючи цей заголовок, повертає не HTML-розмітку, а легкий JSON-об'єкт, який містить лише назву необхідного Vue-компонента та нові дані для нього;
4. клієнтська бібліотека Inertia отримує цей JSON і динамічно замінює поточний Vue-компонент на новий, оновлюючи при цьому історію браузера за допомогою HTML5 History API [28].

2.7 Специфікація вимог до програмного забезпечення

1. Призначення та межі проєкту:

1.1. призначення системи: Вебпортал школи призначений для комплексної автоматизації освітніх та управлінських процесів у закладах загальної середньої освіти. Система розробляється за моделлю SaaS і забезпечує єдиний цифровий екопростір для ведення електронних журналів, управління розкладом занять, організації комунікації та моніторингу успішності;

1.2. погодження, що ухвалені в програмній документації:

– специфікація розроблена з урахуванням нормативних вимог Міністерства освіти і науки України щодо ведення електронного класного журналу та оцінювання учнів;

– ухвалено використання мультитенантної архітектури з фізичною ізоляцією баз даних для забезпечення відповідності міжнародним стандартам захисту персональних даних;

– ухвалено архітектура «Сучасного моноліту» з використанням інструменту Inertia.js для забезпечення реактивності інтерфейсу без створення окремого REST API.

1.3. межі проєкту ПЗ:

– проєкт охоплює розробку клієнт-серверного вебзастосунку з автоматизованою системою маршрутизації піддоменів;

– проєкт включає модуль автоматичного створення бази даних при реєстрації нової школи;

– проєкт НЕ включає розробку нативних мобільних додатків для iOS/Android;

2. Загальний опис:

2.1. сфера застосування: система орієнтована на використання в державних та приватних закладах загальної середньої освіти. Платформа застосовується для організації як очного, так і дистанційного або змішаного форматів навчання.

2.2. характеристики користувачів:

– системний адміністратор, який здійснює глобальний моніторинг серверів, керує підписками шкіл;

– адміністратор школи, керує навчальним процесом конкретної школи, створює класи, розклад, реєструє вчителів;

– вчитель веде електронний журнал, виставляє оцінки, відмічає відсутніх, задає домашні завдання;

– учень переглядає свій розклад, домашні завдання та оцінки;

– батьки: здійснюють моніторинг успішності та відвідуваності виключно своєї дитини.

2.3. загальна структура і склад системи:

– глобальне ядро: центральна база даних, яка містить інформацію про всі зареєстровані тенанти та їхні піддомени;

– клієнтська частина: реактивний інтерфейс користувача, побудований на фреймворку Vue.js 3;

– серверна частина: монолітний застосунок на фреймворку Laravel, що забезпечує бізнес-логіку та взаємодію з БД;

– сполучна ланка: протокол Inertia.js для рендерингу компонентів без перезавантаження сторінки.

2.4. загальні обмеження:

– для роботи з вебпорталом необхідне стабільне підключення до мережі Інтернет;

– кількість підключених шкіл обмежується лише обсягом дискового простору та оперативної пам'яті основного сервера бази даних;

– для забезпечення ізоляції, користувач однієї школи не може бути авторизований у системі іншої школи під тим самим сесійним токеном.

3. Функції системи:

3.1. функція системи «Автоматизована реєстрація школи»:

3.1.1. опис функції: дозволяє представнику школи зареєструвати свій заклад на головній сторінці платформи, після чого система автоматично розгортає для нього готовий портал;

3.1.2. вхідна і вихідна інформація:

– вхідна: назва закладу, бажаний піддомен, ПІБ директора, email, пароль;

– вихідна: посилання на створений портал, статус генерації бази даних.

3.1.3. функціональні вимоги:

– перевірка унікальності бажаного піддомену;

– автоматичне створення ізольованої бази даних на сервері;

– автоматичний запуск міграцій структури таблиць у новій базі;

– створення облікового запису адміністратор школи у щойно створеній БД.

3.2. функція системи «Управління навчальною структурою»:

3.2.1. опис функції: надання інструментарію адміністратору школи для конструювання навчального процесу;

3.2.2. вхідна і вихідна інформація:

– вхідна: списки учнів, вчителів, перелік навчальних предметів, структура навчального року (семестри);

– вихідна: згенеровані облікові записи користувачів, сформовані класи.

3.2.3. функціональні вимоги:

– можливість масового імпорту списку учнів;

– призначення класного керівника для кожного класу;

– формування розкладу уроків із валідацією на перехрещення часу для одного вчителя або кабінету.

3.3. функція системи: Ведення електронного журналу

3.3.1. опис функції: основний робочий інструмент вчителя для обліку знань та відвідуваності;

3.3.2. вхідна і вихідна інформація:

– вхідна: ID учня, ID уроку, оцінка (1-12), тип оцінки (поточна, тематична, семестрова), відмітка про відсутність;

– вихідна: оновлена таблиця журналу, автоматично розрахований середній бал учня.

3.3.3. функціональні вимоги:

– вчитель має доступ лише до тих журналів, які закріплені за ним у розкладі;

– можливість редагування оцінки обмежена часовим вікном;

– миттєва передача виставленої оцінки в персональний кабінет учня/батьків.

3.4. функція системи «Моніторинг успішності»:

3.4.1. опис функції: доступ здобувачів освіти до результатів навчання;

3.4.2. вхідна і вихідна інформація:

– вхідна: запит на перегляд (фільтрація за предметом або тижнем);

– вихідна: електронний щоденник (розклад, оцінки, домашні завдання, коментарі вчителя).

3.4.3. функціональні вимоги:

– строге розмежування доступу: батьки бачать статистику лише своєї дитини;

– можливість завантаження файлів із виконаним домашнім завданням.

4. Вимоги до інформаційного забезпечення:

4.1. Джерела і зміст вхідної інформації (даних):

– дані про шкільну структуру (кабінети, предмети) генеруються адміністрацією школи;

– дані про успішність та домашні завдання генеруються вчителями;

– системні логи генеруються автоматично ядром Laravel.

4.2. нормативно-довідкова інформація (класифікатори, довідники тощо):

– стандартна шкала оцінювання: 1-12 балів;

– класифікатор типів оцінок: поточна, самостійна робота, контрольна робота, тематична, семестрова, річна.

4.3. вимоги до способів організації, збереження та ведення інформації:

– дані кожної школи зберігаються у фізично ізольованих реляційних базах даних;

– паролі користувачів не зберігаються у відкритому вигляді (використовується алгоритм хешування bcrypt);

– резервне копіювання баз даних здійснюється щотижнево.

5. Вимоги до технічного забезпечення:

– сервер СКБД: щонайменше 4 ядра CPU, 8 ГБ RAM;

– вебсервер: щонайменше 2 ядра CPU, 4 ГБ RAM, SSD-накопичувач;

– пристрої користувачів: ПК, ноутбук, планшет або смартфон із сучасним веббраузером (Google Chrome, Safari, Firefox).

6. Вимоги до програмного забезпечення:

6.1. архітектура програмної системи:

– мультитенантна архітектура;

– архітектура «Сучасного моноліту» з використанням патерну MVC на сервері та SPA на клієнті.

6.2. системне програмне забезпечення:

– операційна система сервера: Linux(Ubuntu Server);

– вебсервер: Nginx або Apache.

6.3. мережне програмне забезпечення:

– протоколи HTTP/2, підтримка SSL/TLS шифрування (HTTPS).

6.4. програмне забезпечення ведення інформаційної бази:

– СКБД: MySQL 8.0+ або PostgreSQL 14.

6.5. мова і технологія розробки ПЗ:

– backend: PHP 8.2+, фреймворк Laravel 11;

– frontend: JavaScript, фреймворк Vue.js 3;

– зв'язуючий протокол: Inertia.js;

– стилізація: Tailwind CSS.

7. Вимоги до зовнішніх інтерфейсів:

7.1. інтерфейс користувача:

– повністю адаптивний дизайн, що коректно відображається на роздільних здатностях від 320px (мобільні) до 4K (десктопи);

– основна мова інтерфейсу – українська;

– наявність зрозумілої системи сповіщень про успішні або помилкові дії.

7.2. апаратний інтерфейс:

– специфічних вимог немає. Взаємодія відбувається через стандартні пристрої введення.

7.3. програмний інтерфейс:

– використання внутрішнього API Inertia.js для передачі JSON-даних між Laravel та Vue-компонентами.

7.4. комунікаційний протокол:

– взаємодія між клієнтом та сервером здійснюється виключно за захищеним протоколом HTTPS для запобігання перехопленню конфіденційних освітніх даних.

8. Властивості програмного забезпечення:

8.1. доступність:

Система повинна забезпечувати безперебійну роботу в режимі 24/7. Технічні роботи виконуються у нічний час.

8.2. супроводжуваність:

Код системи розроблений з дотриманням принципів SOLID та стандартів кодування PSR-12, що дозволяє легко додавати нові модулі без руйнування існуючої архітектури.

8.3. переносимість:

Система повинна мати можливість розгортання в контейнерному середовищі , що дозволяє швидко переносити її між різними хмарними провайдерами.

8.4. продуктивність:

Час рендерингу клієнтського інтерфейсу не повинен перевищувати 500 мілісекунд

8.5. надійність:

Система повинна підтримувати ACID. У разі помилки під час складного процесу (наприклад, масового переведення учнів у наступний клас), всі зміни мають бути автоматично скасовані.

8.6. безпека:

– вбудований захист від csrf та xss атак засобами фреймворку laravel;

– управління доступом за допомогою політик для перевірки належності ресурсу конкретному користувачу та тенанту.

9. Інші вимоги:

– система повинна вести журнали аудиту, фіксуючи, хто і коли змінив оцінку в електронному журналі або видалив користувача;

– публічні сторінки порталу (візитки шкіл) повинні бути оптимізовані для пошукових систем.

Висновки до розділу 2

У другому розділі проведено глибокий науково-методичний аналіз підходів до проєктування сучасних освітніх SaaS-платформ. Дослідження архітектурних патернів підтвердило, що для забезпечення найвищого рівня безпеки та масштабованості системи оптимальним вибором є мультитенантна архітектура з фізичною ізоляцією даних за моделлю Database-per-Tenant. Цей підхід дозволяє повністю нівелювати ризики перехресного витоку конфіденційної інформації між школами та ефективно вирішує проблему розподілу апаратних ресурсів при зростанні навантаження на систему.

Особливу увагу приділено питанням інформаційної безпеки та захисту персональних даних в умовах хмарного розгортання. Визначено, що архітектура вебпорталу повинна відповідати суворим міжнародним нормам, зокрема регламенту GDPR. Для реалізації концепції Privacy by Design обґрунтовано необхідність застосування динамічного управління доступом на основі ролей,

надійних методів шифрування та ведення структурованих журналів аудиту, що в комплексі гарантує цілісність та конфіденційність освітніх даних учасників навчального процесу.

Аналіз еволюції вебархітектур довів недоцільність використання як класичного серверного рендерингу, так і повністю розділених SPA-додатків через їхні обмеження у складності підтримки та налаштуванні безпеки. Натомість обґрунтовано вибір концепції «Сучасного моноліту» на базі фреймворків Laravel, Vue.js та сполучного протоколу Inertia.js. Такий технологічний стек усуває потребу в розробці та підтримці окремого REST API, забезпечує високу реактивність інтерфейсу користувача та безшовно інтегрується з обраною моделлю маршрутизації ізольованих баз даних.

На основі проведеного теоретичного аналізу сформовано детальну специфікацію вимог до програмного забезпечення. Визначені функціональні та нефункціональні вимоги чітко регламентують поведінку системи, межі проєкту, рівні доступу користувачів та технічні характеристики. Отримана специфікація разом з обраним технологічним інструментарієм створюють надійний інженерний фундамент.

3 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ

3.1 Проєктування системи засобами UML

Процесу безпосереднього написання коду інформаційної системи обов'язково передує етап архітектурного проєктування. Засобами UML забезпечується візуалізація структурних компонентів та логіки їхньої роботи. Структуризація вимог до SaaS-порталу досягається створенням відповідних графічних моделей. Застосування об'єктно-орієнтованого підходу формує надійний фундамент для побудови масштабованого програмного продукту, мінімізуючи ймовірність архітектурних помилок.

Повноцінний опис розроблюваної освітньої платформи реалізується через побудову взаємопов'язаних діаграм, згрупованих за ключовими системними аспектами:

– документування функціональних вимог, алгоритмів виконання бізнес-процесів та меж доступу для різних користувачів здійснюється за допомогою діаграми прецедентів та діаграми діяльності;

– статична структура системи, базові сутності предметної області, їхні атрибути, а також ієрархічне групування просторів імен відображаються на діаграмі класів і діаграмі пакетів;

– динаміка обміну повідомленнями між об'єктами під час виконання операцій та життєвий цикл окремих сутностей деталізуються діаграмою взаємодії та діаграмою станів;

– організація програмних модулів, фізична схема апаратного забезпечення та розміщення вузлів у хмарному середовищі ілюструється діаграмою компонентів разом із діаграмою розгортання.

З огляду на специфіку архітектури Multi-tenancy, створення комплексу графічних моделей є критично необхідним етапом. Чітке технічне завдання для реалізації back-end сервісів та розробки клієнтського інтерфейсу формується саме завдяки візуальному представленню бізнес-логіки.

3.1.1 Діаграма прецедентів (варіантів) використання

Функціональні можливості акторів та межі створюваного програмного продукту чітко визначаються діаграмою прецедентів (варіантів) використання, наведеною на рисунку 3.1. Побудова діаграми відіграє важливу архітектурну роль у процесі розробки освітнього SaaS-порталу. Структуризація вимог до механізмів суворого рольового контролю доступу забезпечується шляхом ретельного документування базових сценаріїв використання. Таке подання системних процесів створює надійну концептуальну основу для подальшого проєктування серверної логіки.

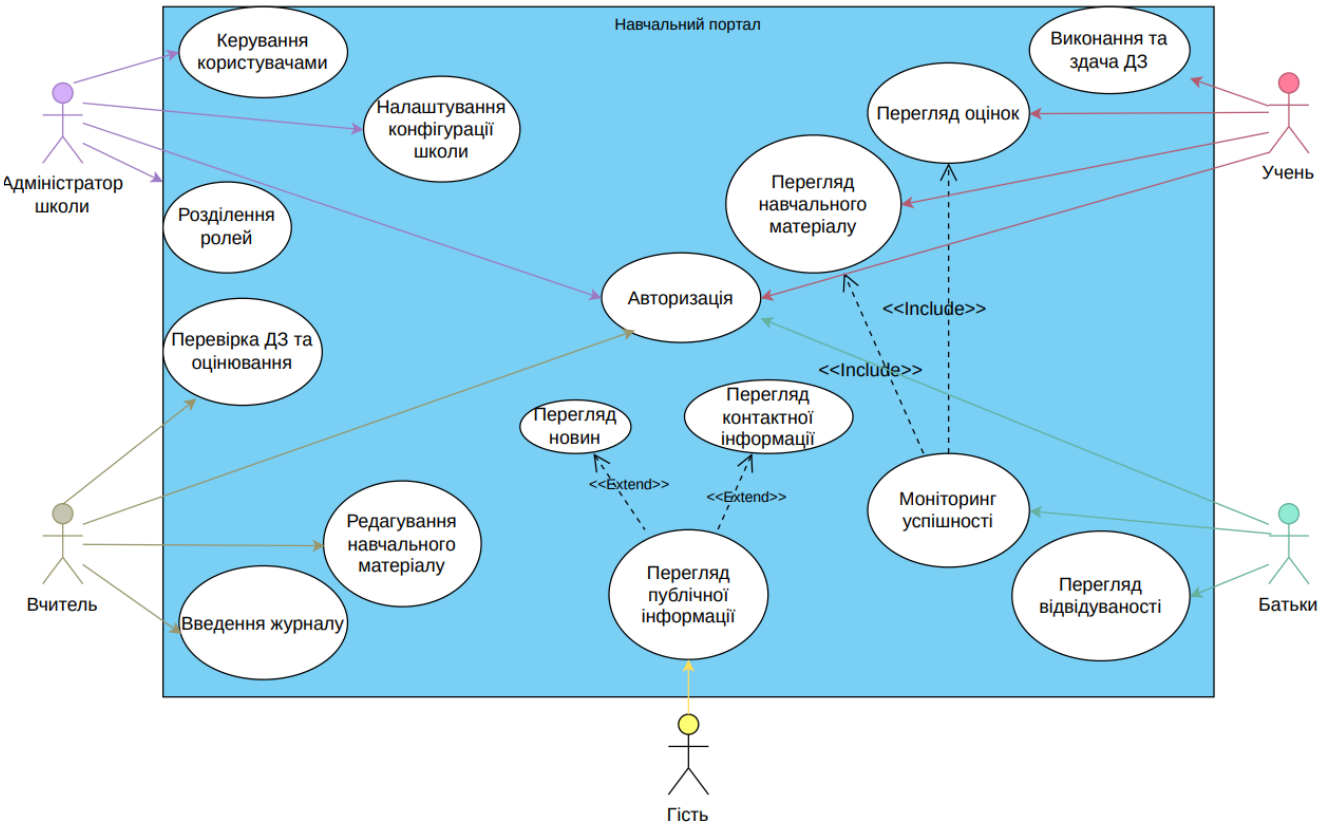

Рисунок 3.1 – Діаграма прецедентів

Діаграма прецедентів демонструє основні функції навчального порталу та взаємодію користувачів із системою. Процес авторизації обов'язково проходиться всіма зареєстрованими учасниками: шкільними адміністраторами, вчителями, учнями та батьками. Облікові записи створюються адміністратором, який також

налаштовує параметри школи. Вчитель веде електронний журнал, перевіряє домашні завдання, виставляє оцінки та завантажує навчальні матеріали.

Учні переглядають наданий контент і здають виконані роботи. Відвідуваність уроків та загальна успішність дитини контролюються батьками. Залежність типу «include» показує включення перегляду оцінок і матеріалів до процесу батьківського моніторингу. Звичайний гість порталу отримує доступ до базових публічних відомостей. Перегляд новин і читання контактної інформації виступають елементами цієї відкритої сторінки

3.1.2 Діаграма взаємодій

Діаграми взаємодії наочно ілюструють хронологічну послідовність обміну повідомленнями між компонентами під час виконання конкретних бізнес-процесів, демонстрації наведена на рисунку 3.2. Завдяки таким графічним моделям деталізується внутрішня логіка функціонування розроблюваного освітнього порталу. Цей візуальний інструмент використовується для глибокого розуміння життєвого циклу окремих операцій клієнт-серверного обміну.

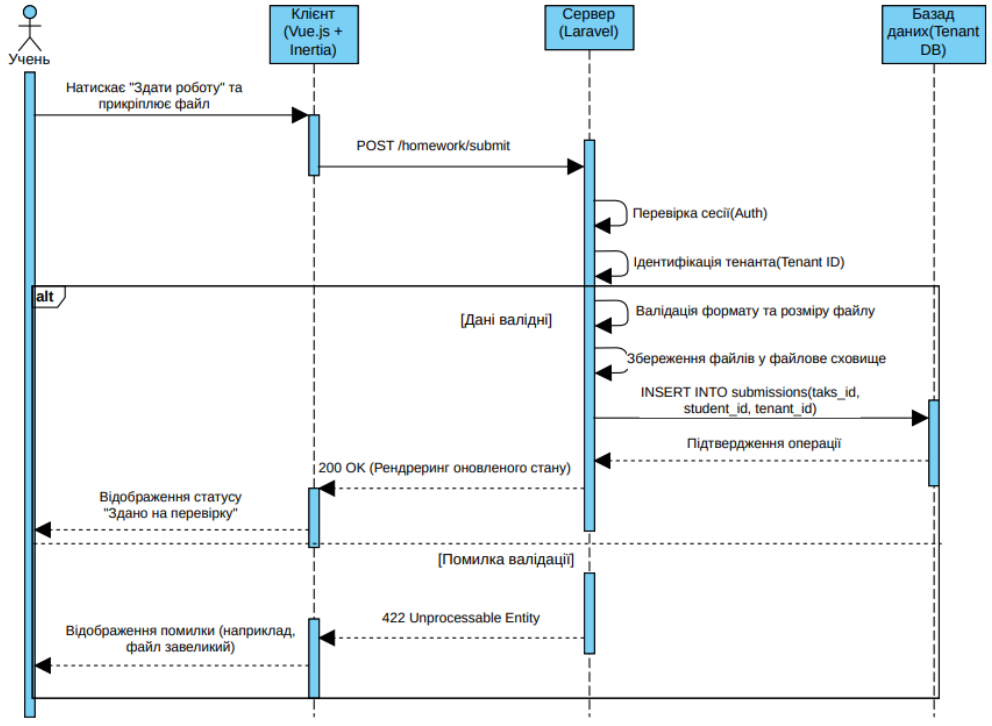

Рисунок 3.2 – Діаграма взаємодій

На діаграмі показано покроковий процес того, як система обробляє завантаження виконаного домашнього завдання. Усе починається з того, що учень прикріплює файл в інтерфейсі та натискає кнопку здачі, після чого застосунок відправляє дані на сервер. Отримавши запит, back-end одразу перевіряє, чи авторизований цей користувач, і визначає, до якої саме школи він належить (Tenant ID). Далі логіка роботи розділяється на два можливі варіанти залежно від того, чи відповідає завантажений файл вимогам до розміру та формату. Якщо відповідає, сервер зберігає роботу в ізольовану базу даних і миттєво оновлює статус на екрані учня, а в разі помилки – відхиляє запит і виводить відповідне попередження.

3.1.3 Діаграма класів

Наступним кроком у проєктуванні є побудова діаграми класів, яка детально розкриває внутрішню статичну структуру нашої системи, як це показано на рисунку 3.3. У випадку освітнього порталу вона допомагає чітко виділити головні сутності: школи, користувачів, навчальні матеріали та логічні зв'язки між ними. Завдяки такій візуалізації можливо заздалегідь побачити всі архітектурні залежності, що робить подальшу розробку передбачуваною. Це також дозволяє уникнути багатьох помилок ще до того, як почнеться написання back-end коду.

Основою підсистеми є абстрактний суперклас User, який об’єдную загальні ідентифікаційні та авторизаційні атрибути (електронна пошта, пароль, базова роль). Від нього реалізовано відношення успадкування до трьох похідних класів: Teacher, Student та SchoolParent. Такий підхід дозволяє винести специфічні дані (наприклад, дату народження учня або кваліфікацію вчителя) у відповідні дочірні класи, уникаючи надмірності даних.

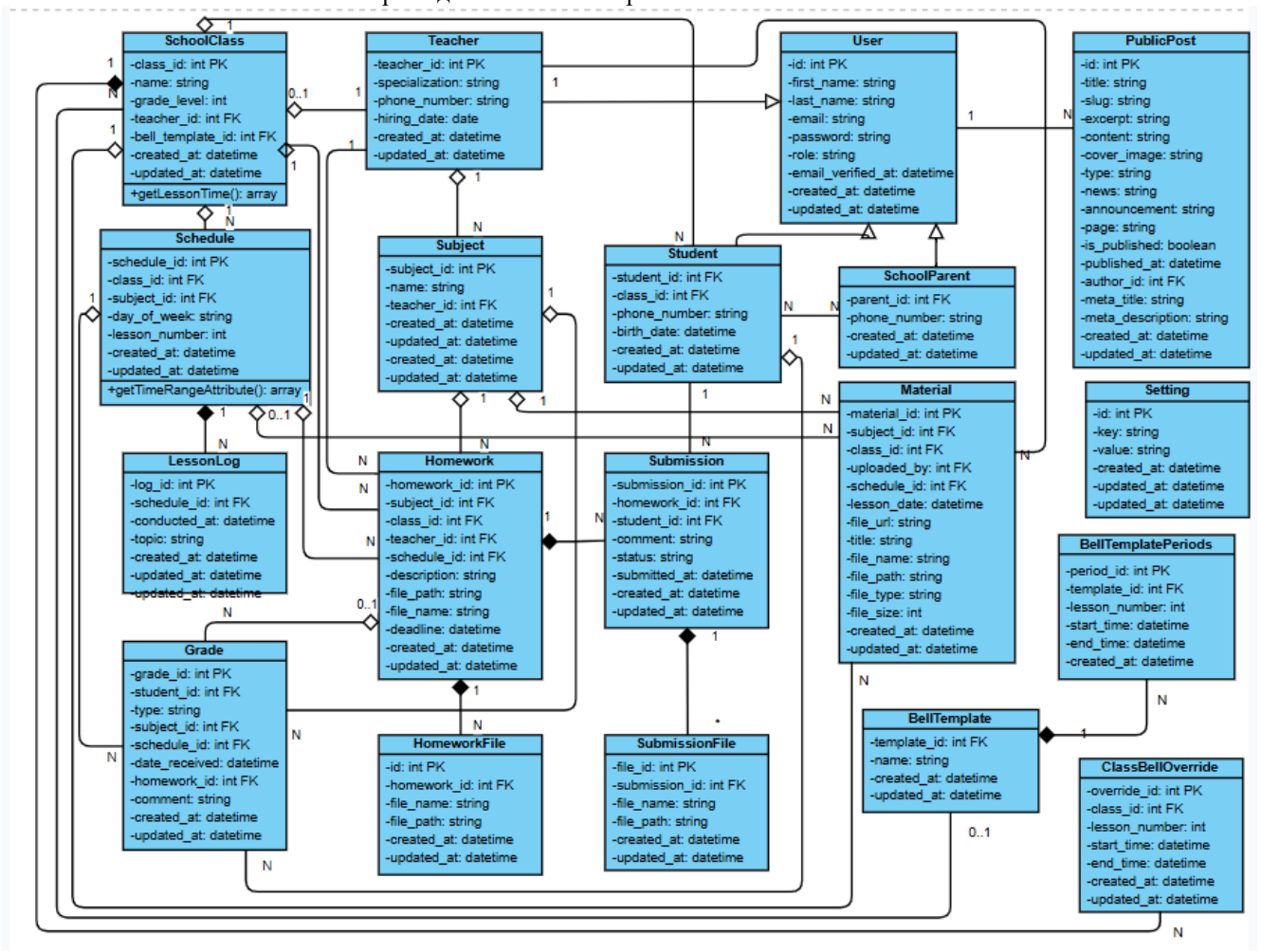

Рисунок 3.3 – Діаграма класів

Структурна організація закладу базується на сутності SchoolClass, яка агрегує об'єкти класу Student та має зв'язок типу асоціації з класним керівником. Академічна діяльність регламентується класом Subject. Центральною ланкою планування виступає клас Schedule, який об'єднує предмет, клас та час проведення заняття. Для фіксації фактично проведених уроків передбачено сутність LessonLog, що має зв'язок композицію із розкладом.

Сутність Homework дозволяє додавати файлові вкладення через відношення композиції. Учні взаємодіють із цим модулем, створюючи екземпляри класу Submission, до яких також можуть додаватися робочі файли.

Оцінювання успішності реалізовано через клас Grade. Ця сутність має гнучку архітектуру зв'язків. Вона в обов'язковому порядку асоційована з учнем, проте може бути прив'язана як до конкретного завдання, так і до загального розкладу чи предмета. Це дозволяє системі фіксувати різні типи оцінок: за домашні роботи,

активність на уроці або підсумкові контрольні бали. Додатково передбачено клас Material для збереження навчальних матеріалів та методичних посібників із прив'язкою до предметів або конкретних уроків.

Базові розклади зберігаються в класі BellTemplate, що містить окремі часові періоди BellTemplatePeriods. Шкільні класи за замовчуванням посилаються на ці загальні шаблони. Однак для обробки виняткових ситуацій (наприклад, індивідуального графіка для окремого класу) впроваджено сутність ClassBellOverride, яка дозволяє гнучка перевизначити час початку та завершення уроків без дублювання базових розкладів.

Інформаційна підтримка навчального закладу забезпечується сутністю PublicPost, яка відповідає за публікацію новин та оголошень авторизованими співробітниками. Глобальне конфігурування порталу здійснюється через клас Setting, який зберігає системні параметри у форматі ключ-значення.

3.1.4 Діаграма діяльності

Наступним кроком є моделювання динамічної поведінки системи. Для візуалізації реальних бізнес-процесів та алгоритмів платформи найкраще підходить діаграма діяльності. Вона допомагає крок за кроком проілюструвати послідовність операцій, розгалуження сценаріїв та паралельні завдання. Створення такої схеми робить роботу ключових модулів, наприклад, процес завантаження навчальних матеріалів чи виставлення оцінок, максимально зрозумілою, що продемонстровано на рисунку 3.4.

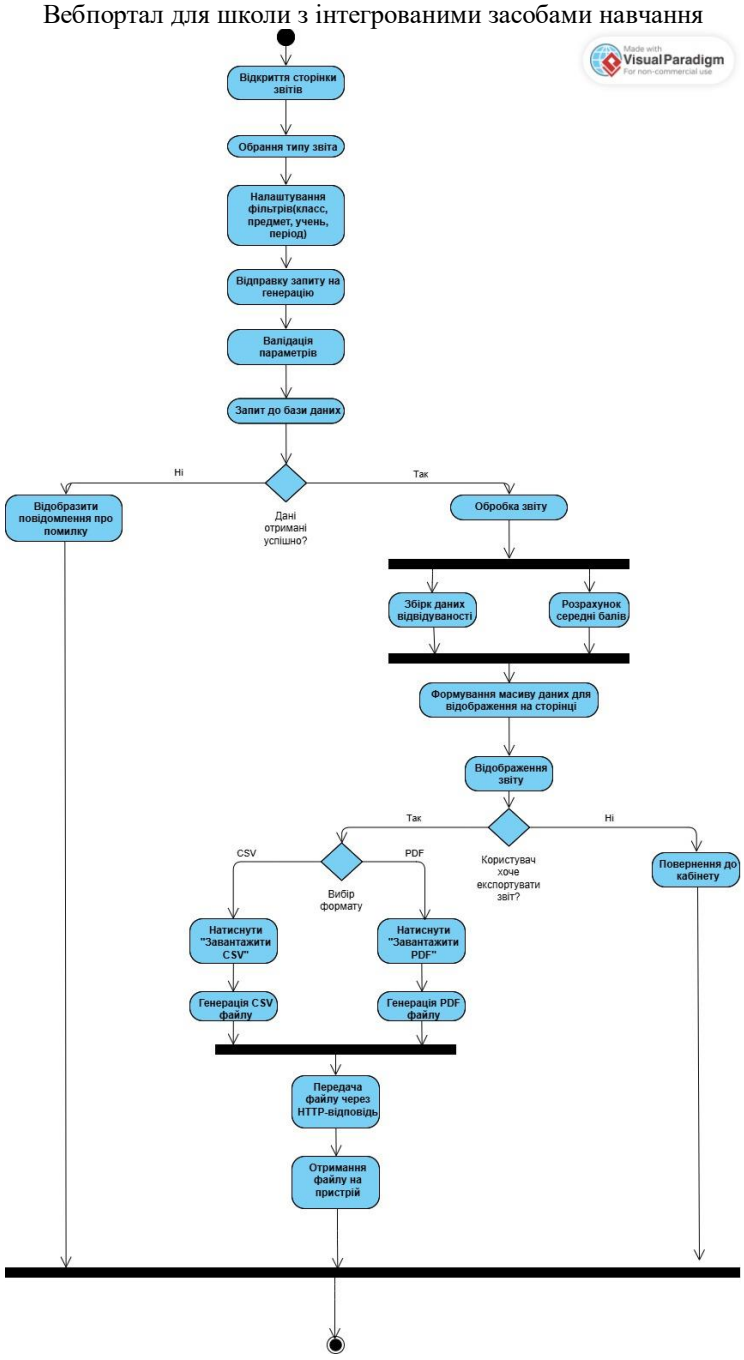

Рисунок 3.4 – Діаграма діяльності

На рисунку 3.4 зображено діаграму діяльності, яка детально моделює алгоритм роботи, модуля генерації звітів користувачем. Бізнес-процес ініціюється на клієнтській частині, де користувач переходить на сторінку звітів, обирає потрібний тип документа та налаштовує відповідні фільтри, такі як конкретний клас, навчальний предмет, учень або визначений часовий період. Сформований запит передається на серверну частину, яка першочергово виконує строгу валідацію вхідних параметрів. Після успішної перевірки система здійснює звернення до бази даних. На цьому етапі алгоритм містить вузол прийняття

рішення. Якщо за вказаними критеріями інформації не знайдено або виникає помилка, процес переривається, користувачу відображається відповідне сповіщення, і потік управління переходить до фінальної стадії. У разі ж успішного отримання даних система ініціює блок обробки звіту, ключовою особливістю якого є використання ліній синхронізації, розгалуження та злиття потоків. Це дозволяє паралельно або в межах незалежних транзакцій виконувати ресурсномісткі операції зі збору даних про відвідуваність та розрахунку середніх балів, після чого результати об'єднуються у єдиний структурований масив.

Готовий аналітичний звіт відображається на екрані користувача для попереднього ознайомлення, після чого алгоритм пропонує варіативність подальших дій. Користувач має змогу завершити поточну роботу зі звітом або ініціювати процедуру експорту документа. Якщо обрано експорт, система обробляє додаткове логічне розгалуження залежно від бажаного формату файлу (PDF або CSV). Відповідно до зробленого вибору, сервер генерує цільовий документ та передає його клієнту за допомогою HTTP-відповіді. Процес успішно завершується фактичним завантаженням файлу на пристрій користувача.

3.1.5 Діаграма компонентів

Для відображення системи з погляду її фізичної організації та поділу на самостійні програмні блоки застосовується діаграма компонентів Зазначена модель ілюструє групування вихідного коду в окремі функціональні модулі, що підлягають автономній збірці, розгортанню та повторному використанню. Візуалізація компонентної структури надає змогу чітко визначити межі відповідальності кожної частини системи, а також інтерфейси, за допомогою яких забезпечується взаємодія та обмін даними між ними під час виконання програмного коду, що детально проілюстровано на рисунку 3.5.

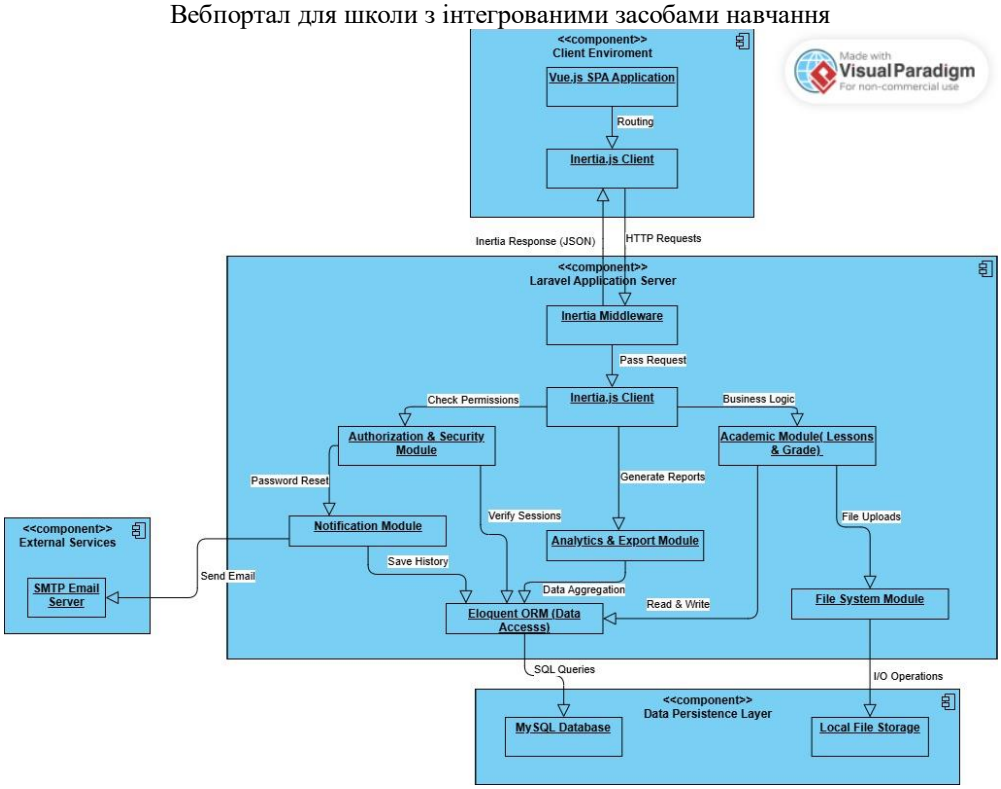

Рисунок 3.5 – Діаграма компонентів

Наведена UML-діаграму компонентів, ілюструє багаторівневу структурну архітектуру розробленої освітньої SaaS-платформи. Програмний комплекс логічно розподілено на чотири ключові підсистеми: клієнтське середовище, сервер додатків, рівень збереження даних та зовнішні інтеграційні сервіси. На стороні клієнта функціонує вебзастосунок формату SPA, побудований на базі фреймворку Vue.js. Взаємодія із серверною частиною здійснюється за допомогою інструментарію Inertia.js. Завдяки такому рішенню клієнт надсилає стандартні HTTP-запити та отримує готові структуровані відповіді у форматі JSON, зберігаючи при цьому високу швидкість рендерингу.

Головним обчислювальним вузлом системи є Laravel Application Server. Вхідні клієнтські запити обробляються компонентом Inertia Middleware, після чого маршрутизатор передає управління відповідним модулям бізнес-логіки. Архітектура сервера містить автономні функціональні блоки: модуль авторизації та безпеки, академічний модуль для керування навчальним процесом (уроками та оцінками), а також підсистему аналітики для генерації звітів. Окрему роль відіграє модуль сповіщень, який не лише фіксує події у системі, але й асинхронно взаємодіє

із зовнішнім поштовим SMTP-сервером для відправки електронних листів користувачам. Важливою характеристикою моделі є те, що всі компоненти бізнес-логіки повністю ізольовані від прямого контакту з базою даних: агрегація, читання та запис інформації виконуються виключно через Eloquent ORM. Фізичне збереження інформації реалізовано на іншому рівні, який об'єднує реляційну СКБД MySQL для роботи зі структурованими даними за допомогою SQL-запитів та локальне дискове сховище для виконання операцій вводу-виводу при роботі з користувацькими файлами.

3.1.6 Діаграма розгортання

Діаграма розгортання являє собою структурну UML-модель, що визначає топологію апаратного забезпечення та специфіку розміщення на ньому програмних компонентів. Головна мета цієї схеми полягає у проєкції логічної архітектури, створеної на етапі проєктування, на реальні фізичні пристрої або віртуальні сервери. У межах розподілених систем така візуалізація наочно демонструє точний розподіл програмного забезпечення між окремими обчислювальними вузлами.

Саме програмне забезпечення на цій діаграмі фіксується у вигляді конкретних артефактів (архівів, виконуваних файлів чи бібліотек), які завантажуються у цільове середовище виконання. Оскільки сучасні інфраструктури зазвичай складаються з багатьох ізольованих серверів та клієнтських пристроїв, модель також обов'язково ілюструє шляхи зв'язку.

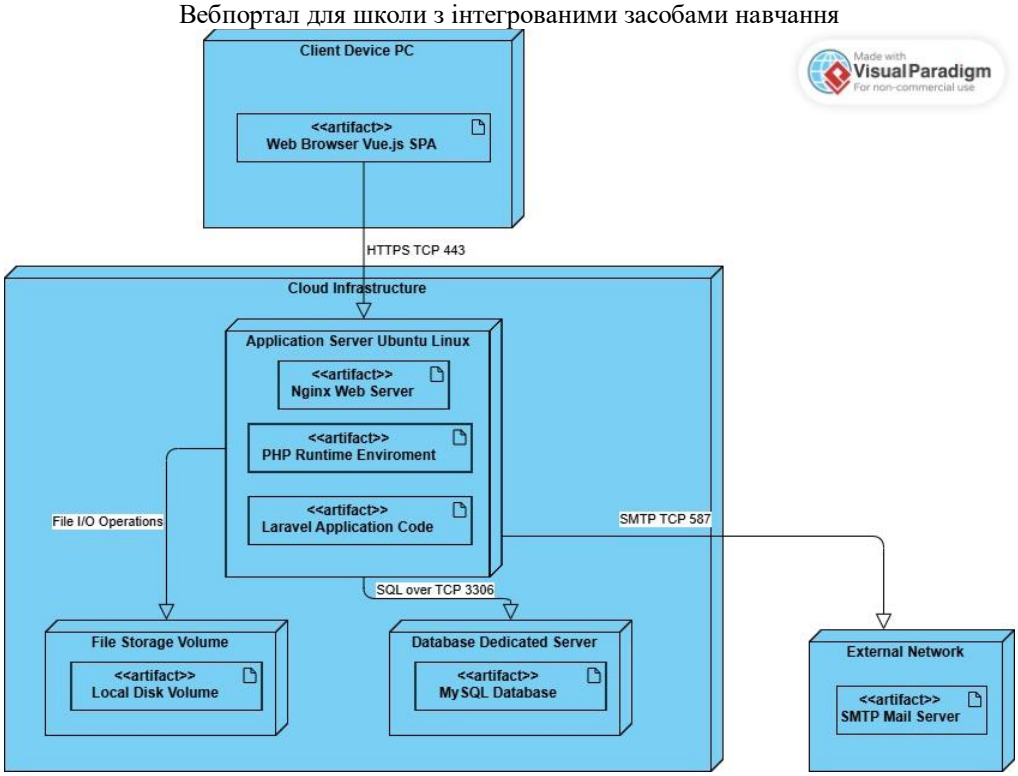

Рисунок 3.6 – Діаграма розгортання

Ця модель візуалізує розподіл виконуваних програмних артефактів між апаратними обчислювальними вузлами системи. На рівні взаємодії з користувачем виділено вузол клієнтського пристрою, де основним середовищем виконання виступає веббраузер. Саме в ньому розгортається та функціонує артефакт клієнтської частини платформи, SPA-застосунок на базі фреймворку Vue.js. Комунікація між пристроєм користувача та центральною хмарною інфраструктурою здійснюється виключно через захищений протокол HTTPS із використанням порту TCP 443, що гарантує надійне шифрування трафіку.

Обчислювальним центром системи є хмарна інфраструктура, базовим вузлом якої виступає сервер додатків під керуванням операційної системи Ubuntu Linux. На цьому сервері розгорнуто три ключові артефакти: вебсервер Nginx для обробки вхідних з'єднань, середовище виконання PHP та безпосередньо вихідний код застосунку Laravel. З метою забезпечення стійкості до високих навантажень архітектура передбачає розподіл ресурсів: рівень збереження даних винесено на окремий виділений сервер із розгорнутою СКБД MySQL, взаємодія з якою відбувається за допомогою SQL-запитів через внутрішній порт TCP 3306. Збереження завантажених матеріалів через вузл файлового сховища, до якого

застосунок звертається через системні виклики вводу-виводу (File I/O Operations). Додатково на діаграмі відображено інтеграцію системи із зовнішньою мережею, куди сервер додатків маршрутизує запити через порт TCP 587 до незалежного SMTP-сервера для забезпечення безперебійної доставки email-сповіщень.

3.2 Сценарії використання системи

Метод сценаріїв використання є базовим інструментом для опису того, як саме користувач взаємодіє із системою для вирішення своїх завдань. Цей підхід відіграє ключову роль в аналізі вимог, оскільки він перетворює загальні побажання на конкретну послідовність дій та системних відповідей. Завдяки цьому вся команда отримує єдине та чітке бачення майбутнього продукту. Сценарії допомагають зрозуміти, які саме функції платформа має виконувати, а які виходять за її межі. При цьому опис завжди ведеться з погляду кінцевого користувача: фіксується те, що робить система, свідомо уникаючи технічних деталей того, «як» це реалізовано на рівні програмного коду.

Як продемонстровано в таблиці 3.1, кожен такий варіант використання враховує різні шляхи досягнення результату. Базовий сценарій описує ідеальний розвиток ситуації, коли користувач виконує операцію успішно, без жодних помилок і найкоротшим шляхом. Однак під час роботи часто виникають нестандартні ситуації: неправильно введені дані, відсутність необхідних прав доступу чи переривання операції. Усі ці винятки та перешкоди фіксуються як альтернативні варіанти розвитку подій. Це дає змогу заздалегідь продумати реакцію платформи на помилки та зробити її більш надійною у реальних умовах.

Крім того, подібний формат опису суттєво спрощує роботу всім учасникам процесу розробки. Програмісти отримують однозначне розуміння функціоналу, а фахівці з тестування можуть використовувати ці описані кроки як готову основу для створення тест-кейсів, що значно економить час. Оскільки кожен Use Case безпосередньо відповідає певній бізнес-потребі (наприклад, оцінюванню учнів чи завантаженню матеріалів), стає набагато легше перевіряти, чи виконує платформа свої головні завдання.

Таблиця 3.1 – Сценарій редагування публічного контенту

Usecase section	**Comment**
Назва прецеденту	Редагування публічного контенту сайту школи
Сфера дії	Модуль управління інформаційним наповненням мультиорендної освітньої платформи
Рівень	Успішне оновлення інформаційних сторінок або стрічки новин конкретного навчального закладу (орендаря)
Головний актор	Адміністратор школи
Зацікавлені особи та інтереси	– адміністратор школи прагне оперативно опублікувати актуальні відомості в межах свого цифрового простору; – відвідувачі сайту мають потребу у перегляді свіжої інформації про установу.
Передумови	Користувач успішно авторизувався в системі з відповідними правами доступу.
Гарантії успіху	– наявність у користувача веббраузера з підтримкою сучасних стандартів javascript та html5; – стабільне підключення клієнтського пристрою до мережі інтернет; – успішна валідація та збереження внесених змін в ізольованій базі даних відповідного орендаря; – коректна реактивна синхронізація стану клієнтського інтерфейсу після отримання відповіді від сервера.
Основний сценарій успіху	– адміністратор здійснює перехід до адміністративної панелі у розділ управління контентом; – система за допомогою маршрутизації клієнтської частини динамічно рендерить інтерфейс без повного перезавантаження сторінки; – користувач обирає необхідну категорію публікацій та активує форму додавання нового матеріалу; – введення текстового вмісту та заголовка реалізується через інтегровані компоненти візуального редактора; – здійснюється вибір та завантаження графічного медіафайлу для обкладинки матеріалу; –

Кінець таблиці 3.1

Основний сценарій успіху	– серверна частина платформи проводить валідацію отриманого запиту, оптимізує зображення та фіксує об'єкт в ізольованій базі даних поточного тенанта; оновлені дані миттєво відображаються на публічній вебсторінці відповідного закладу.
Розширення	Помилка валідації форми: – одне або кілька обов'язкових полів залишаються незаповненими користувачем; – платформа повертає помилку обробки форми, підсвічує некоректні поля в інтерфейсі та виводить відповідні підказки. невдала спроба завантаження медіафайлу: – графічний об'єкт перевищує встановлений ліміт розміру або має непідтримуваний формат; – система блокує подальше завантаження файла на сервер, скасовує операцію та демонструє користувачеві попереджувальне повідомлення.
Спеціальні вимоги	Швидкість передачі даних для клієнтського підключення повинна становити не менше 60 кб/сек.
Список технологічних варіантів	– використання компонентної архітектури клієнтської частини на базі зв'язки vue.js та inertia.js з адаптивною стилізацією через tailwind css; – обробка запитів та збереження даних засобами інструментарію laravel eloquent orm із підключенням серверних бібліотек обробки зображень.
Частота виникнення	30%
Різне	Чи потрібно реалізовувати автоматичне стиснення, зміну розміру та оптимізацію графічних файлів на стороні сервера перед їхнім збереженням у сховище?

Сценарій ілюструє покрокову взаємодію адміністратора закладу із системою під час створення чи оновлення новинних матеріалів, підкреслюючи специфіку роботи в multi-tenant архітектурі, де дані кожного навчального закладу зберігаються повністю ізольовано. Окрім базового алгоритму успішної публікації,

специфікація визначає обов'язкові передумови ініціалізації процесу, зокрема наявність успішної авторизації з відповідними правами доступу. Також детально регламентується поведінка back-end частини у разі виникнення виняткових ситуацій, таких як невідповідність даних правилам валідації форми або спроба завантаження некоректних графічних файлів. Додатково в описі акцентується увага на технологічному аспекті реалізації: використанні реактивного клієнтського інтерфейсу на базі Vue.js та Inertia.js, що гарантує миттєве відображення внесених змін без необхідності повного перезавантаження вебсторінки

Таблиця 3.2 – Сценарій перевірки домашнього завдання вчителем

Usecase section	**Comment**
Назва прецеденту	Оцінювання та перевірка виконаного домашнього завдання
Сфера дії	Модуль управління навчальним процесом
Рівень	Успішне виставлення балів, надання зворотного зв'язку та закриття завдання
Головний актор	Вчитель
Зацікавлені особи та інтереси	– вчитель прагне зручно переглянути надіслані файли, оцінити рівень знань та залишити коментарі до помилок; – учень зацікавлений у швидкому отриманні об'єктивної оцінки та роз'яснень щодо своєї роботи.
Передумови	– вчитель успішно авторизований у системі з відповідними правами доступу; – учень надіслав роботу, і вона має статус «очікує перевірки».
Гарантії успіху	– введена оцінка пройшла валідацію та коректно збережена в базі даних; – статус роботи змінено на «оцінено»; – учень отримав автоматичне системне сповіщення про перевірку.
Основний сценарій успіху	– вчитель переходить до розділу «перевірка робіт» в адміністративній панелі порталу; – система відображає перелік учнів, які надіслали відповіді на обране завдання;

Кінець таблиці 3.2

Основний сценарій успіху	– вчитель відкриває картку конкретної роботи учня; – знайомиться з текстом відповіді або завантажує прикріплені файли для детального аналізу; – вводить підсумковий бал у відповідне поле та, за бажанням, додає текстовий коментар; – натискає кнопку «зберегти результати перевірки»; – сервер обробляє запит, фіксує оцінку в базі даних та змінює статус роботи; – модуль сповіщень ініціює відправку повідомлення учню про виставлену оцінку.
Розширення	Некоректний формат оцінки:: – вчитель вводить бал, що перевищує максимально допустимий для цього або вводить текст замість цифр; – система блокує збереження, підсвічує поле помилки та вимагає ввести коректне значення. повернення на доопрацювання: – вчитель виявляє, що робота виконана неправильно або файл учня пошкоджений/не відкривається; – замість виставлення балів, вчитель залишає коментар з описом проблеми та натискає кнопку «повернути на доопрацювання»; – система знімає статус «очікує перевірки» та повертає завдання учню для повторної здачі
Спеціальні вимоги	– швидкість передачі даних для клієнтського підключення повинна становити не менше 60 кб/сек; – інтерфейс перевірки повинен підтримувати швидку навігацію між роботами різних учнів без повного перезавантаження сторінки.
Список техн. варіантів	– реалізація інтерфейсу за допомогою vue.js та inertia.js для швидкого перемикання між учнями; – використання патерну observer у laravel для автоматичного тригерування події відправлення сповіщення після збереження оцінки.
Частота виникнення	80%
Різне	Чи зберігається історія змін оцінок для огляду?

Сценарій описує послідовність дій викладача під час огляду надісланих учнівських робіт. Даний сценарій відіграє ключову роль у забезпеченні зворотного зв’язку в освітньому процесі. Базовий алгоритм передбачає перегляд прикріплених матеріалів, виставлення балів та додавання пояснювальних коментарів. Разом з тим, опис враховує і нестандартні варіанти розвитку подій. Серед них – спрацювання механізмів валідації при спробі збереження некоректних оцінок (наприклад, поза межами встановленої шкали), а також процедура повернення завдання учню на доопрацювання у разі виявлення суттєвих недоліків чи пошкоджених файлів. Успішне завершення цих дій логічно замикає цикл роботи із завданням, після чого система автоматично генерує сповіщення для інформування учня про результати перевірки.

3.3 Проєктування логічної та фізичної структури бази даних

У зв'язку з реалізацією multi-tenant архітектури за принципом Database-per-Tenant, загальна система збереження інформації вебпорталу логічно та фізично розділена на два автономні рівні: центральну базу даних платформи та ізольовані бази даних окремих навчальних закладів.

Центральна база даних виступає головним координаційним вузлом платформи. Вона відповідає за реєстрацію нових орендарів, збереження їхніх системних конфігурацій та динамічну маршрутизацію мережевих запитів за доменними іменами. Основними сутностями цього рівня є тенанти та домени. Логічна та фізична структура цих таблиць, а також реляційні зв'язки між ними відображені на схематичній діаграмі центральної бази даних (рис. 3.9).

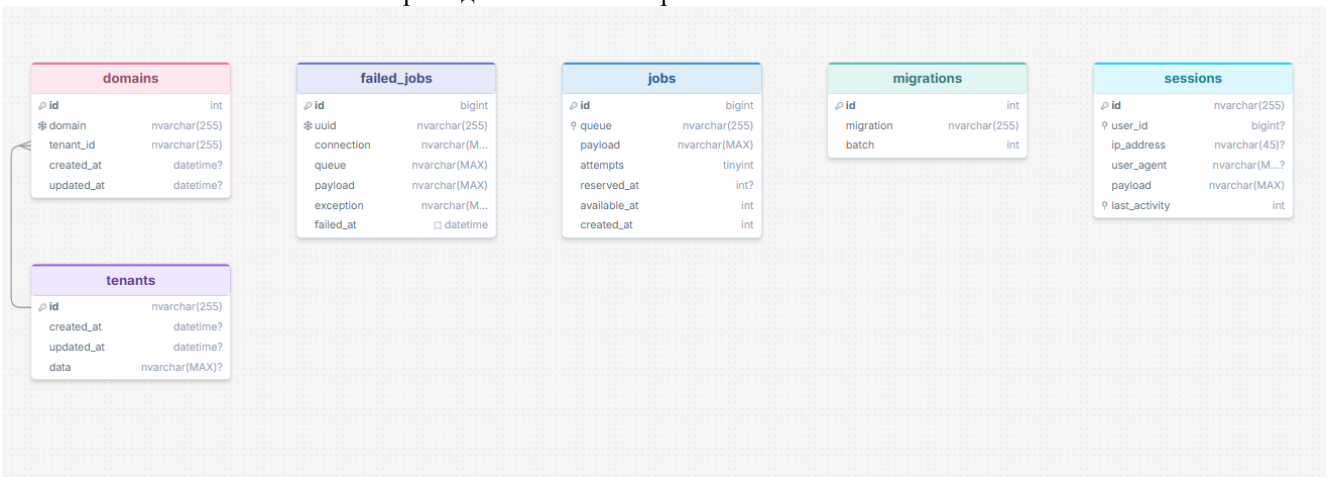

Рисунок 3.7 – Діаграма бази даних SchoolDatabase

Таблиця tenants використовує рядковий унікальний ідентифікатор id (тип nvarchar(255)) як первинний ключ, а також містить поле data (тип nvarchar(max)), призначене для збереження динамічних метаданих та індивідуальних системних налаштувань конкретної школи у форматі JSON. Таблиця domains слугує для прив'язки унікальних вебадрес до відповідних шкіл. Вона містить автоматичний первинний ключ id, текстове поле domain із суворим обмеженням унікальності та зовнішній ключ tenant_id, який посилається на відповідний запис у таблиці tenants. Для забезпечення цілісності інформації на рівні СКБД MS SQL Server для зовнішнього ключа налаштовано правила каскадного оновлення та видалення що автоматично очищує пов'язані домени при видаленні об'єкта школи.

Важливою особливістю є те, що всі таблиці, системні індекси та реляційні зв'язки як у центральній базі даних, так і в базах даних орендарів, не створюються вручну через графічний інтерфейс СКБД. Їхнє формування, модифікація та контроль версій структури повністю делеговані механізму міграцій фреймворку Laravel. Системна таблиця migrations фіксує життєвий цикл розгортання бази даних. Це дозволяє повністю автоматизувати процес підготовки інфраструктури, гарантує абсолютну ідентичність схем даних для всіх нових тенантів під час їхнього динамічного створення та спрощує подальше супроводження платформи.

Логічна та фізична структура бази даних окремого навчального закладу спроєктована у третій нормальній формі для забезпечення цілісності та мінімізації надмірності даних. Ядром архітектури є система профілів користувачів.

Абстрактна інформація (ім'я, електронна пошта, пароль, базова роль) зберігається у центральній таблиці users. Деталізовані дані, що відповідають специфіці ролей, реалізовано через зв'язки «один-до-одного» з відповідними дочірніми таблицями: teachers (збереження спеціалізації та дати працевлаштування), students (прив'язка до класу та дата народження) і parents. Для відображення родинних зв'язків між батьками та учнями використовується проміжна таблиця parent_student, що реалізує відношення Many-to-Many.

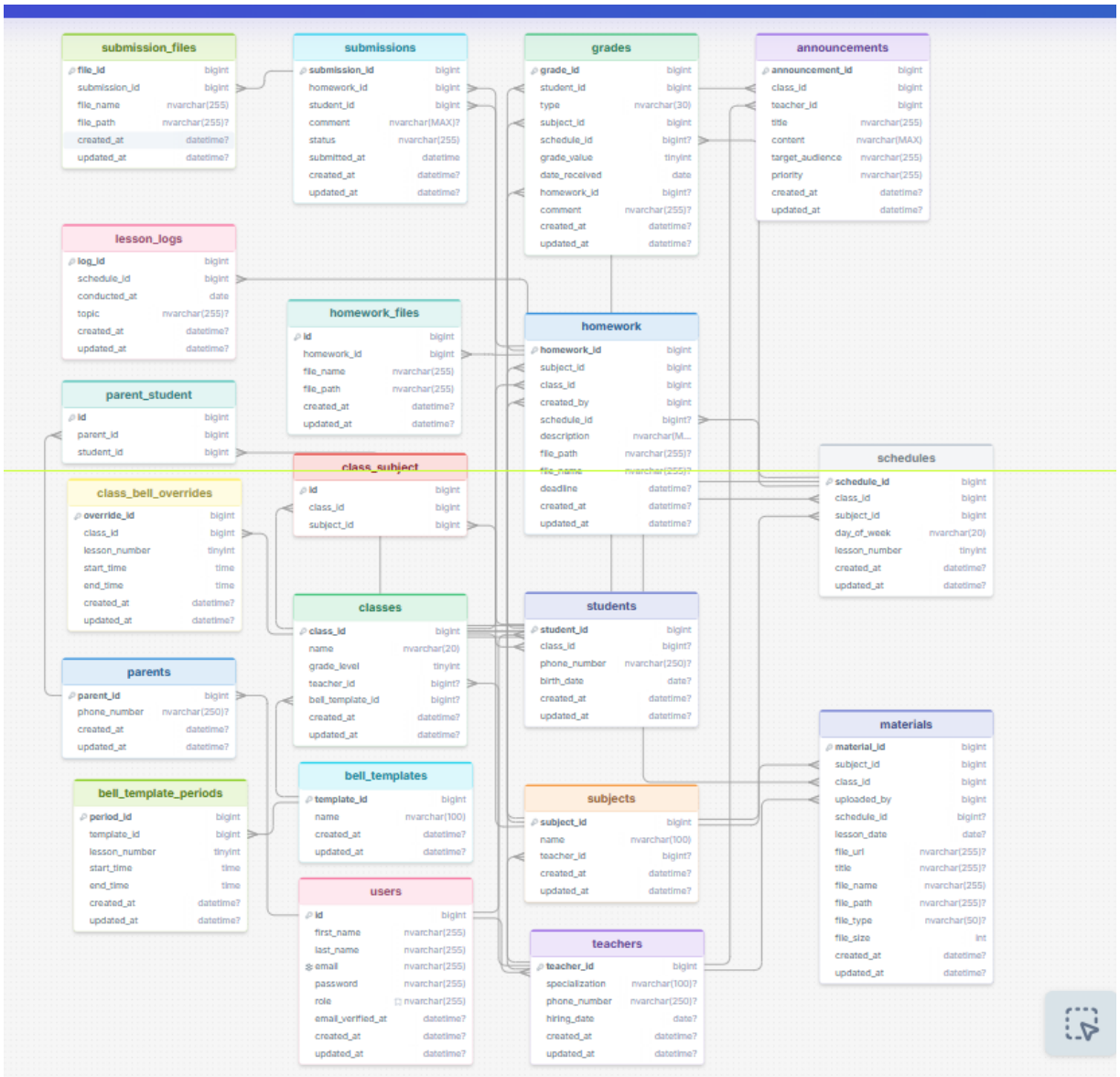

Рисунок 3.8 – Діаграма ізольованої бази даних навчального закладу

Навчальний процес модулюється складною ієрархією реляційних зв'язків. Базовою структурною одиницею є таблиця classes, яка агрегує учнів та має зв'язок

із класним керівником. Розклад занять фіксується у таблиці schedules, що об'єднує предмети, класи та конкретний час проведення (на основі шаблонів дзвінків із таблиць bell_templates та bell_template_periods). Виставлення балів реалізовано через гнучку таблицю grades, яка дозволяє фіксувати оцінки як за виконання поточних завдань (через зовнішній ключ homework_id), так і за активність на уроці (через schedule_id). Відповіді учнів на домашні завдання та прикріплені файли зберігаються в таблицях submissions та submission_files відповідно. Візуалізовану ER-діаграму бази даних навчального закладу, що відображає ключові сутності та зв'язки між ними, наведено на рисунку 3.8.

3.4 Проєктування користувацького інтерфейсу

Першим етапом взаємодії майбутніх клієнтів (навчальних закладів) із розробленою SaaS-платформою є головна Landing Page. Вона виконує функцію візитної картки продукту та головної точки входу для ініціалізації нових орендарів у системі. Дизайн цієї сторінки спроєктовано з акцентом на максимальну інформативність, що дозволяє користувачеві швидко ознайомитися з можливостями платформи та розпочати роботу без залучення служби технічної підтримки.

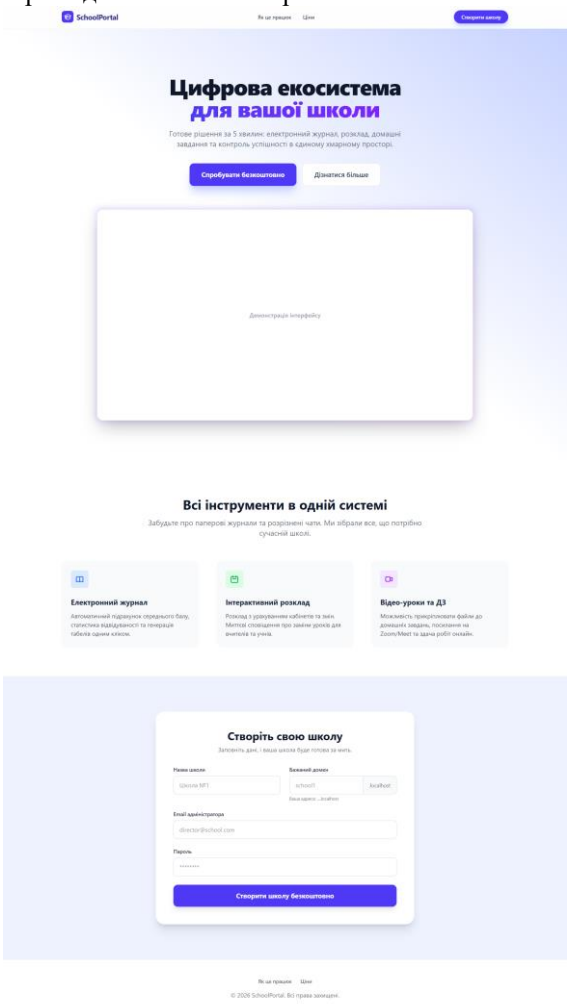

Рисунок 3.9 – Інтерфейс Landing Page

Верхній інформаційний блок виконано у світлій колірній схемі з використанням м'яких фіолетово-синіх градієнтів, які асоціюються з технологічністю. Центральне місце займає чітка ціннісна пропозиція та контрастні кнопки заклику до дії. Нижче розташовано модульний блок презентації ключового функціоналу: електронного журналу та інтерактивного розкладу. Використання мінімалістичних іконок та достатньої кількості вільного простору між елементами знижує когнітивне навантаження на користувача (рис. 3.9).

Після успішної реєстрації та розгортання ізольованої бази даних, система автоматично генерує унікальну публічну сторінку для новоствореного навчального закладу. Цей інтерфейс виконує роль офіційного вебсайту школи в мережі Інтернет і слугує точкою доступу як для неавторизованих відвідувачів, так і для учасників освітнього процесу.

Дизайн публічної сторінки школи спроєктовано з використанням фірмового зеленого кольору, що надає інтерфейсу академічності. Головний екран містить привітання, персоналізовану назву закладу (яка динамічно підвантажується з бази даних поточного тенанта) та кнопку швидкого переходу до останніх публікацій. У верхній частині екрана передбачено закріплену панель навігації, яка забезпечує швидкий доступ до розділів новин, інформації про заклад та контактних даних. Важливим елементом навігаційної панелі є кнопка «Увійти в кабінет», яка ініціює процес авторизації користувача (рис. 3.12).

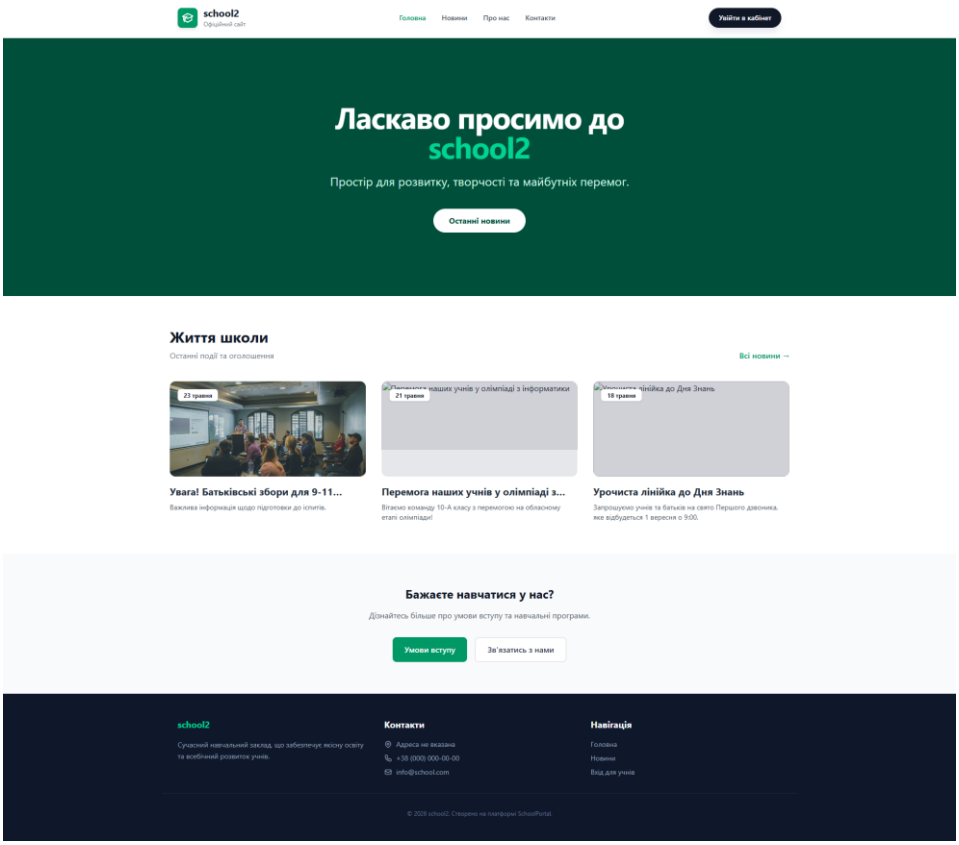

Рисунок 3.12 – Інтерфейс головної сторінки публічного вебсайту школи

Центральним інформаційним модулем публічного сайту є стрічка новин та подій (рис. 3.12). Інтерфейс цього блоку реалізовано у вигляді адаптивної сітки карток. Кожна картка новини огортає тематичне зображення, дату публікації, заголовок та короткий анонс події. Використання мікроанімацій при наведенні курсора на картки робить інтерфейс більш інтерактивним та спонукає користувача до взаємодії. Контент для цього розділу управляється адміністрацією школи через

закриту панель управління, що гарантує актуальність інформації на публічному сайті.

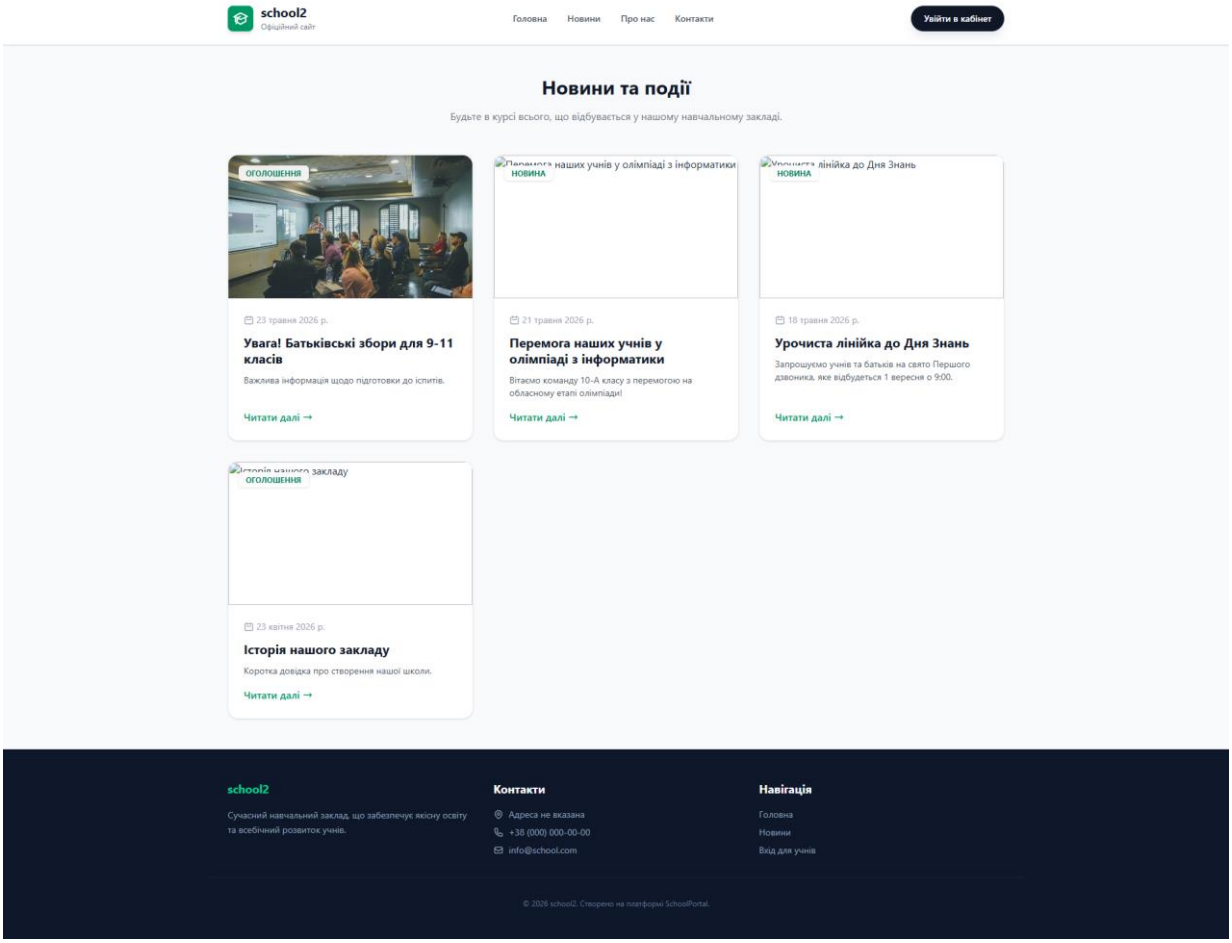

Рисунок 3.13 – Інтерфейс відображення новин

Для забезпечення зворотного зв'язку між відвідувачами сайту (наприклад, батьками потенційних учнів) та адміністрацією закладу, спроєктовано окремий інтерфейс розділу контактів (рис. 3.13). Він візуально поділений на два логічні блоки: інформаційну панель із фізичною адресою, телефоном та інтегрованою картою (Google Maps), а також форму зворотного зв'язку. Форма містить базові поля для введення імені, електронної пошти та тексту повідомлення, оснащена клієнтською валідацією засобами Vue.js для запобігання відправці некоректних даних. Загальна стилістика сторінок підтримується єдиним підвалом сайту, який дублює навігаційні посилання та містить інформацію про авторські права розробленої платформи.

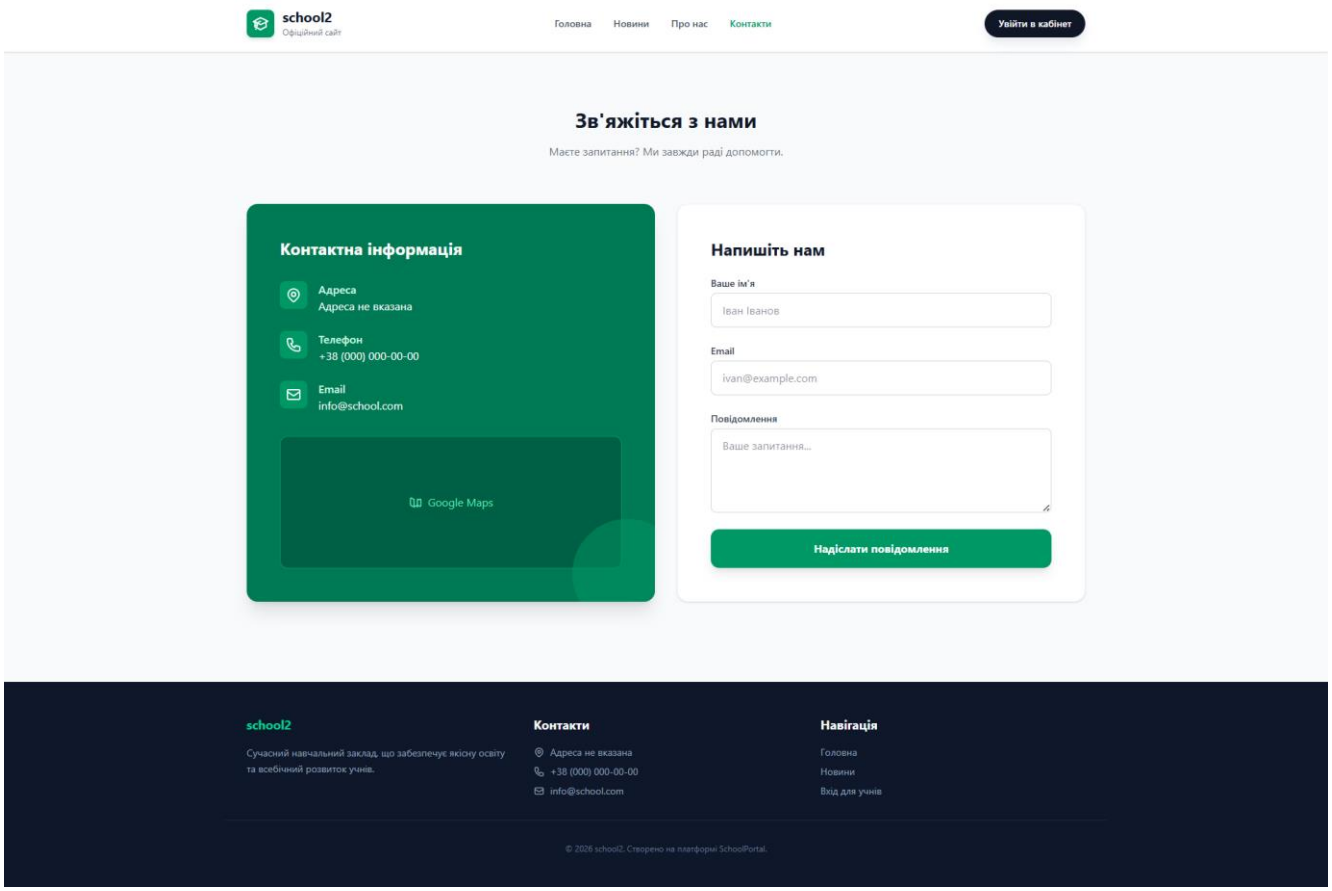

Рисунок 3.13 – Інтерфейс сторінки зворотного зв’язку та контактної інформації

Сторінка «Про нас» слугує для формування іміджу навчального закладу та презентації його основних досягнень, що реалізовано через поєднання статичних інформаційних блоків та динамічних елементів візуалізації даних (рис. 3.14). Верхня частина сторінки містить інформативний віджет статистики, який за допомогою великих контрастних шрифтів на фірмовому зеленому фоні відображає ключові показники діяльності школи, такі як кількість учнів, викладацького складу, років історії та відсотковий показник успішності. Основний контентний блок розділений на смислові зони: ліворуч розміщено медіаконтент, що ілюструє шкільне життя, а праворуч текстове звернення директора, оформлене з використанням іменної фотокартки та посадових метаданих, що додає персоналізації та сприяє побудові довірливих відносин із відвідувачами сайту. Окремим функціональним вузлом сторінки є блок «Адміністрація», де представлено команду керівництва та ключових працівників закладу у вигляді сітки з круглими портретними фотографіями, іменами та посадами. Такий

інтерфейс забезпечує швидку ідентифікацію персоналу та підтримує концепцію відкритості освітнього простору, а завдяки адаптивній верстці Tailwind CSS всі елементи сторінки автоматично масштабуються залежно від роздільної здатності пристрою, зберігаючи при цьому читабельність текстів та пропорційність графічних блоків.

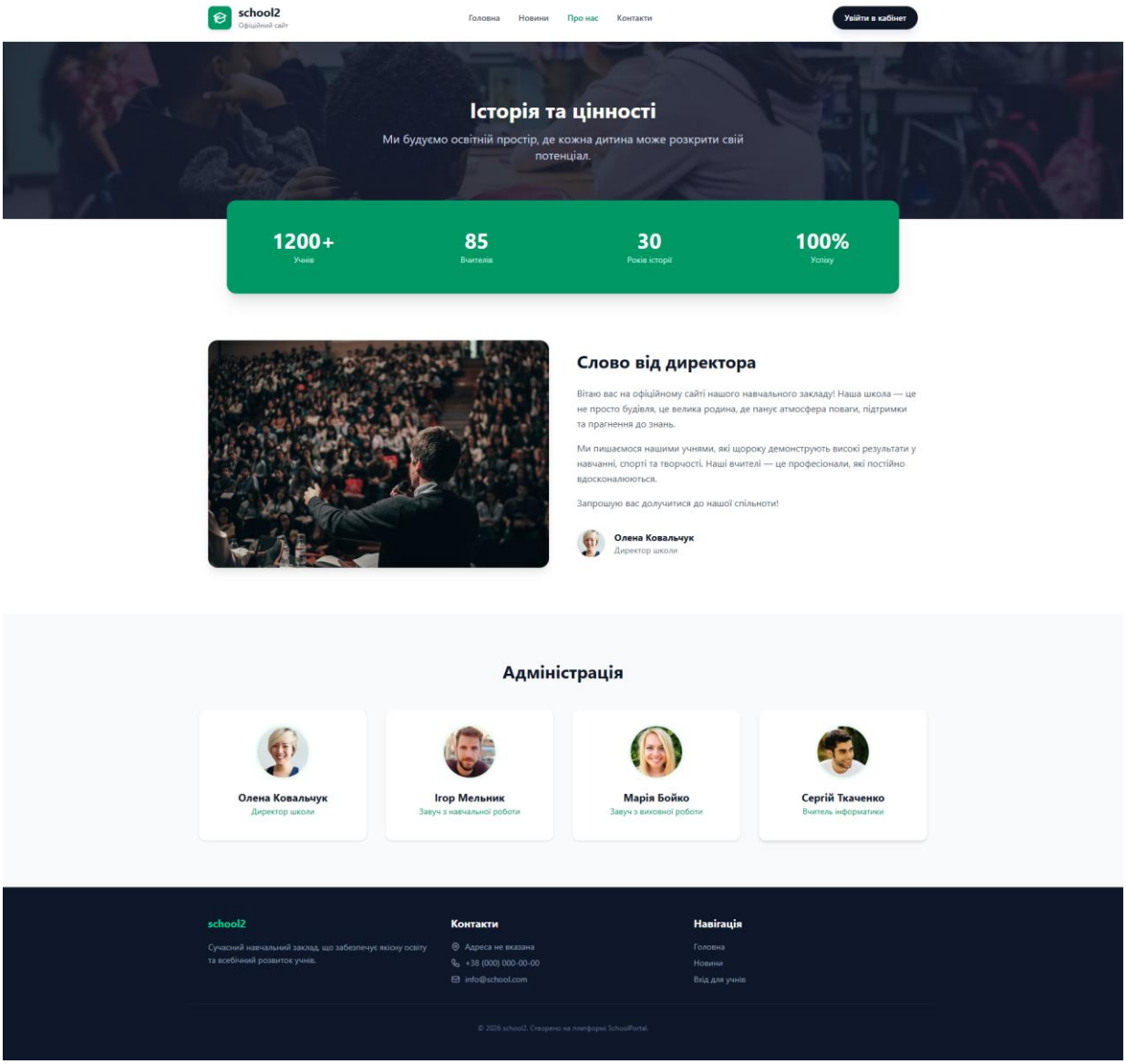

Рисунок 3.14 – Інтерфейс сторінки «Про нас»

Сторінка авторизації спроєктована з дотриманням мінімалізму, фокусуючи увагу користувача на центральному функціональному блоці входу до системи (рис. 3.15). Інтерфейс містить лише необхідні поля для введення електронної адреси та пароля, прапорець опції «Запам’ятати мене» та кнопку підтвердження,

при цьому назва навчального закладу динамічно підтягується з бази даних поточного тенанта, що забезпечує персоналізацію доступу. Використання достатньої кількості вільного простору навколо форми та нейтральна кольорова гама мінімізують відволікаючі чинники, створюючи безпечне та зручне середовище для початку роботи з особистим кабінетом.

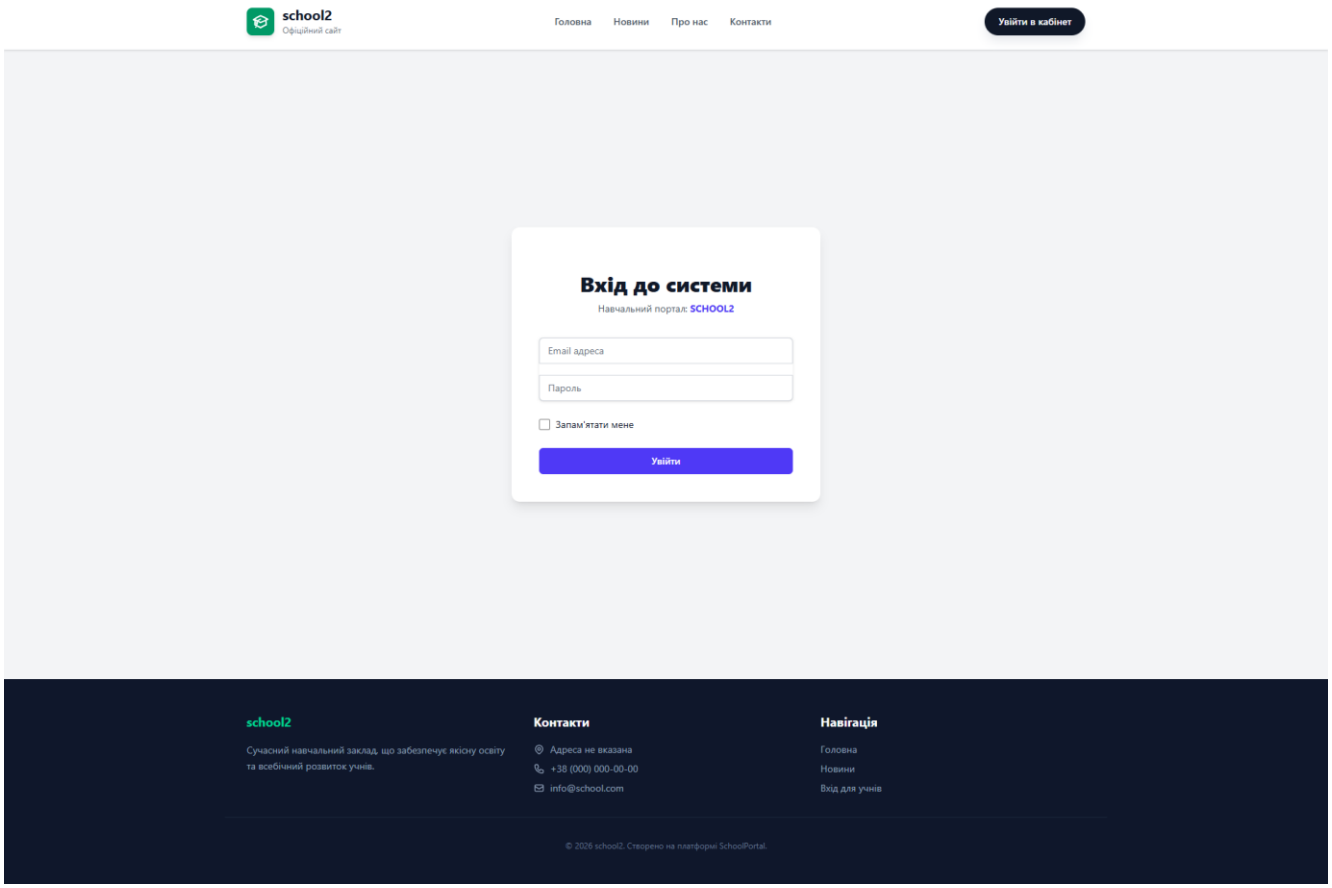

Рисунок 3.15 – Інтерфейс форми авторизації

Центральним елементом закритої частини платформи є панель управління адміністратора, яка забезпечує швидкий доступ до моніторингу стану системи та виконання ключових операцій. Інтерфейс побудовано на основі двопанельної архітектури: зліва розташовано вертикальну навігаційну панель із чітким групуванням посилань на розділи управління користувачами, навчальним процесом та налаштуваннями порталу, що дозволяє адміністратору миттєво перемикатися між ключовими функціональними модулями (рис. 3.16). Основна робоча область містить верхній інформаційний рядок із показниками актуальної статистики (кількість учнів, вчителів, класів та предметів), що дозволяє оцінити

масштаб системи без необхідності відкриття окремих звітів. Центральний блок інтерфейсу реалізовано у вигляді набору карток для ініціації найбільш затребуваних адміністративних завдань створення облікових записів або додавання публікацій, що суттєво оптимізує щоденну роботу. Додатково праворуч розміщено стрічку, яка у хронологічному порядку відображає історію змін у системі, забезпечуючи адміністратору контроль над операціями, виконаними іншими користувачами. Така компоновка робочого простору відповідає принципам системної ергономіки, мінімізуючи час на пошук необхідних інструментів та забезпечуючи високий рівень наочності стану освітньої екосистеми.

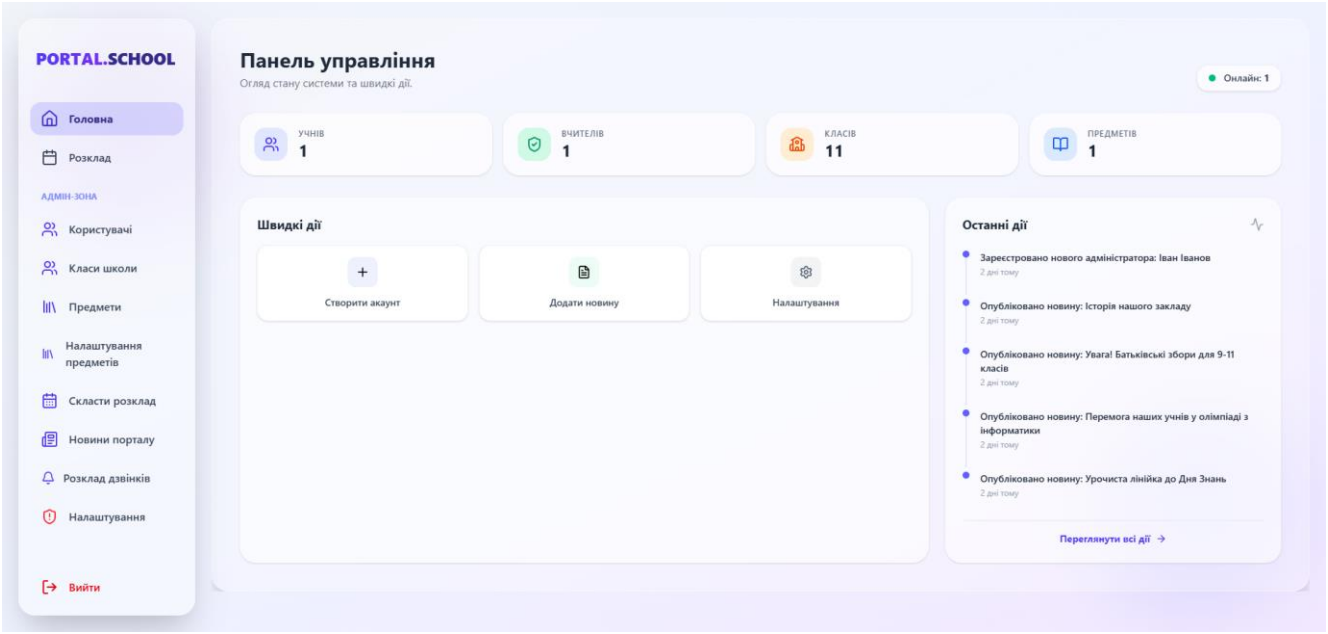

Рисунок 3.16 – Інтерфейс панелі управління адміністратор школи

Сторінка перегляду розкладу є інформаційним вузлом освітнього процесу, доступ до якого реалізовано з урахуванням ролевої моделі доступу (рис. 3.17). Для учнів та їхніх батьків інтерфейс розкладу автоматично адаптується та відображає дані, прив'язані до конкретного класу, до якого належить учень, що усуває потребу в додатковій фільтрації та спрощує користувацький досвід. Натомість для вчителів та адміністраторів система надає ширші можливості, дозволяючи вільно переглядати розклад занять будь-якого класу в межах школи. Інтерфейс модуля візуально імітує класичний паперовий щоденник, що забезпечує високу інтуїтивність сприйняття, та включає функціональні поля для фіксації предметів,

домашніх завдань і оцінок, гарантуючи прозорість організації освітнього процесу для всіх учасників системи.

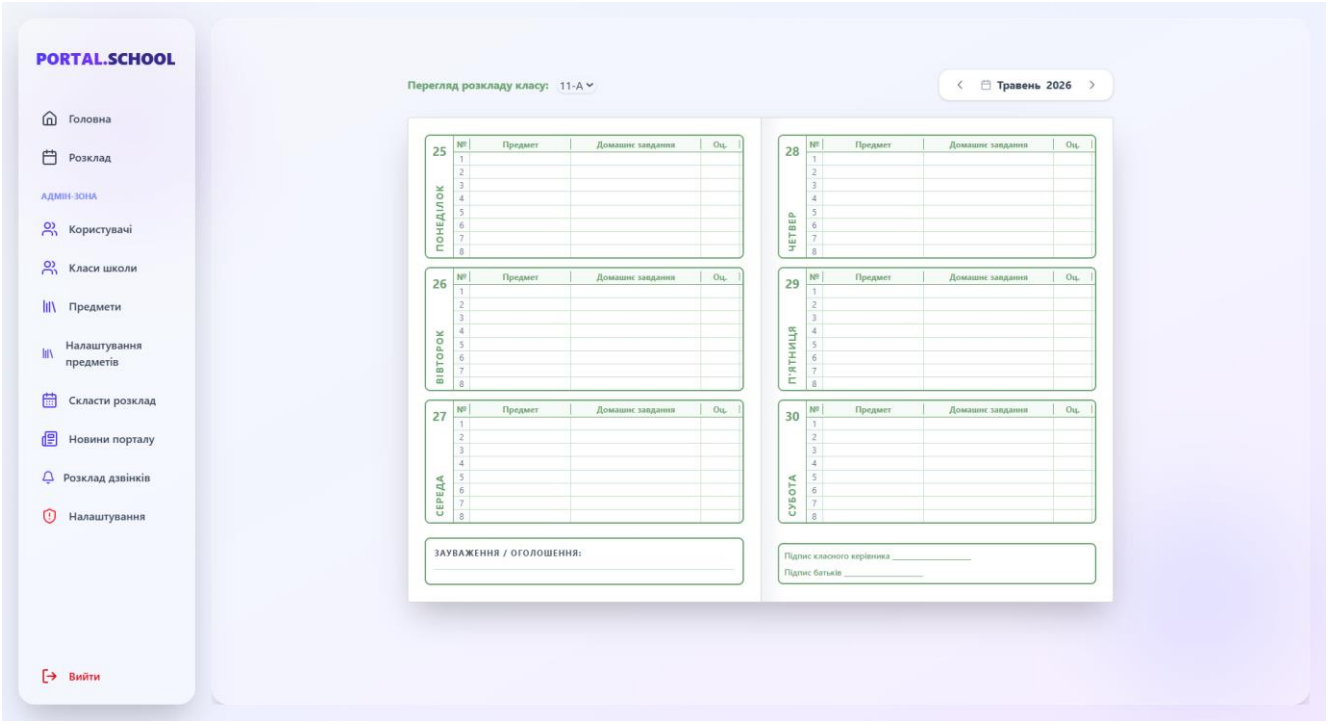

Рисунок 3.17 – Інтерфейс розкладу занять

Панель управління вчителя спроєктована як персоналізований робочий простір, що забезпечує миттєвий доступ до поточного розкладу та пріоритетних завдань. Верхній блок вітання містить інтерактивну картку, яка за допомогою контрастного фіолетового фону акцентує увагу на найближчому навчальному занятті, надаючи вчителю швидкий доступ до початку уроку, інформацію про клас та кабінет (рис 3.18). Поруч розміщено індикатор денного навантаження, що візуалізує прогрес виконання навчального плану, та блок «Потребує уваги», де система автоматично сповіщає про недоліки в журналі, наприклад, про незаповнені теми попередніх занять. Основну частину робочої області займає інтерактивний розклад на день, який дозволяє вчителю бачити структуру свого графіку та оперативно розпочинати роботу. Такий інтерфейс мінімізує час на планування, фокусуючи увагу викладача на навчальному процесі та виправленні адміністративних пропусків.

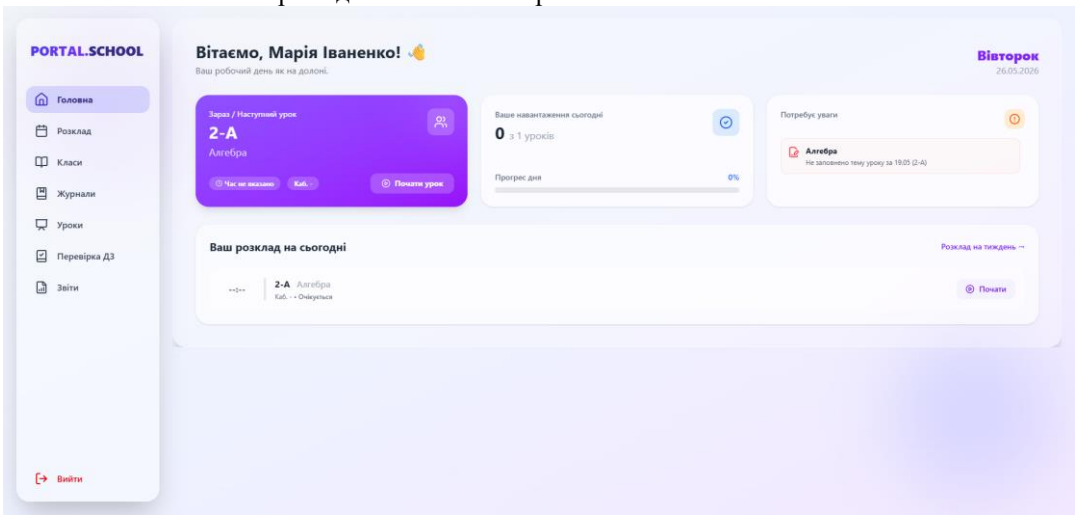

Рисунок 3.18 – Інтерфейс головної панелі управління вчителя

Панель управління учня це персоналізований навчальний хаб, що забезпечує швидкий огляд актуальної успішності та розпорядку дня. Інтерфейс містить блок з деталями проведення, індикатор активних домашніх завдань та графічний прогрес-бар середнього бала, що дозволяє учню миттєво оцінити свій поточний рівень успішності (рис. 3.19). Компактний список уроків на сьогодні структурує навчальний час, перетворюючи дашборд на ефективний інструмент самоорганізації та швидкого доступу до ключових навчальних матеріалів.

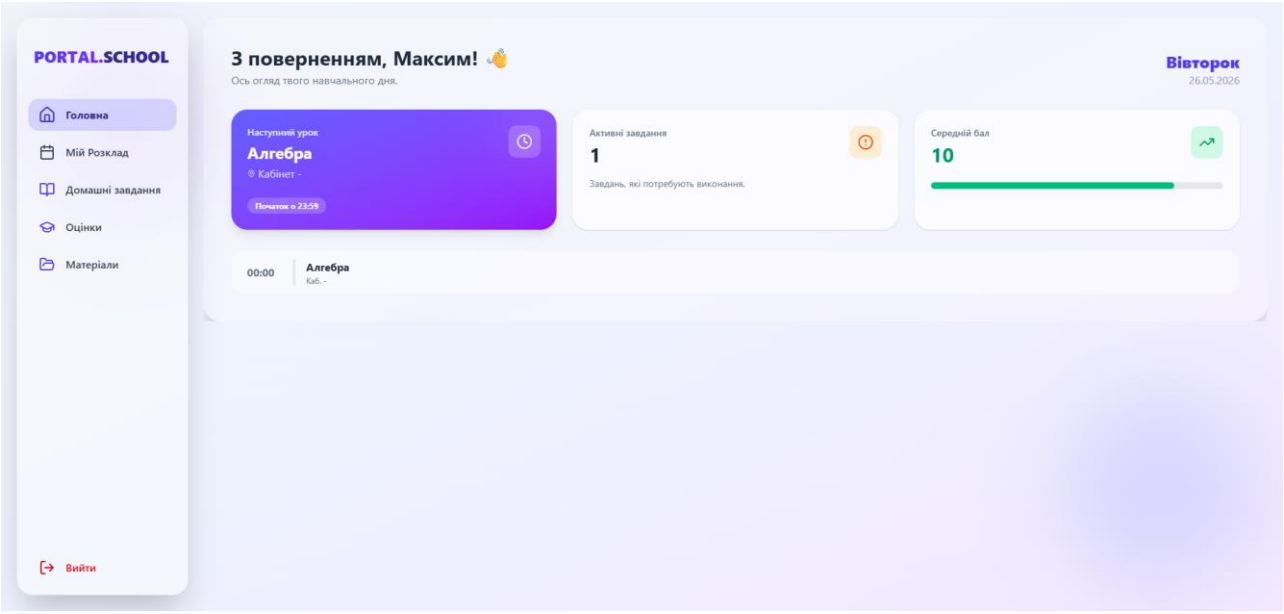

Рисунок 3.19 – Інтерфейс головної панелі управління учня

Панель управління батьків є спеціалізованою сторінкою для оперативного моніторингу успішності та стану навчання дітей. Інтерфейс включає блок, який

дозволяє в реальному часі відстежувати поточне заняття дитини, картку з останньою отриманою оцінкою для швидкого реагування на успіхи, а також зону для сповіщення про наявність заборгованостей чи важливих нотаток (рис. 3.20).

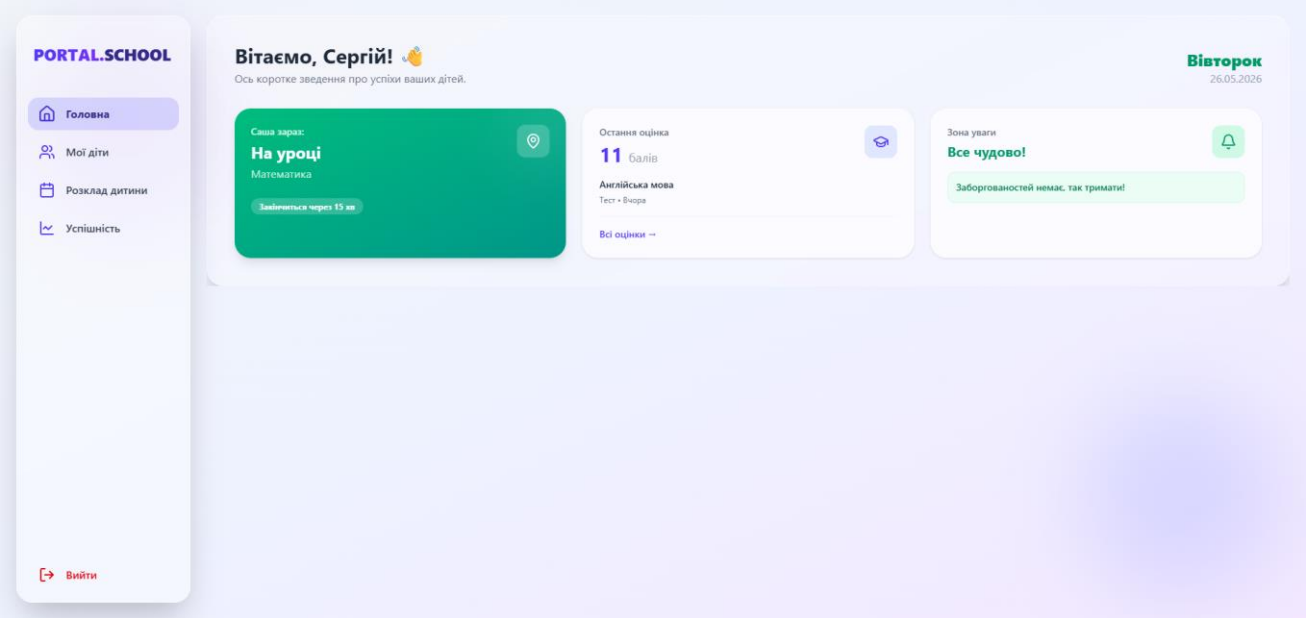

Рисунок 3.20 – Інтерфейс головної панелі управління учня

Така компоновка забезпечує батькам комплексний і лаконічний огляд навчального процесу без необхідності детального аналізу журналів, що значно спрощує процес контролю та комунікації зі школою.

Висновки до розділу 3

У третьому розділі проведено комплексне архітектурне проєктування та моделювання освітньої SaaS-платформи. Завдяки використанню методології об'єктно-орієнтованого аналізу та інструментарію UML розроблено набір взаємопов'язаних графічних моделей: діаграми прецедентів, класів, діяльності, взаємодії, компонентів та розгортання. Це дозволило чітко візуалізувати статичну структуру та динамічну поведінку системи, визначити межі відповідальності програмних модулів та спроєктувати надійний фундамент для реалізації ізольованої multi-tenant архітектури.

Важливим кроком стала деталізація функціональних вимог через розробку та опис сценаріїв використання. На прикладі базових бізнес-процесів, таких як редагування публічного контенту та перевірка домашніх завдань викладачем,

формалізовано алгоритми взаємодії користувачів із платформою. Врахування як основних, так і альтернативних варіантів розвитку подій (наприклад, спроби завантаження некоректних файлів, порушення дедлайнів або помилки валідації) забезпечило підґрунтя для створення стійкої до відмов системи із продуманим та безпечним користувацьким досвідом.

Особливу увагу приділено проєктуванню бази даних, де використано концепцію фізичної ізоляції даних кожного навчального закладу. Така модель у поєднанні з автоматизованою системою міграцій Laravel не лише забезпечує найвищий рівень безпеки та конфіденційності інформації орендарів, але й мінімізує вплив людського фактора під час ініціалізації нових робочих просторів. Описана структура даних у третій нормальній формі гарантує цілісність, відсутність надмірності та високу швидкість обробки запитів.

Проєктування користувацького інтерфейсу ґрунтувалося на принципах людино-орієнтованого дизайну та компонентної архітектури. Використання сучасного стека технологій, дозволило реалізувати реактивний та адаптивний інтерфейс, що забезпечує безшовну взаємодію користувачів із системою незалежно від типу пристрою.

Рольова адаптивність інтерфейсу виступає ключовим інструментом персоналізації освітнього простору. Система динамічно трансформує інформаційне наповнення робочих областей залежно від прав доступу: від адміністративних інструментів налаштування порталу до спеціалізованих дашбордів вчителя, учня та батьків.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Архітектурне проєктування системи та аналіз технологічного стеку

Проєктування сучасної освітньої SaaS-платформи вимагає виваженого підходу до вибору архітектурних рішень та інструментів розробки. Головне завдання на цьому етапі полягає у створенні надійної системи, яка працюватиме швидко, легко масштабуватиметься та гарантуватиме безпеку персональних даних. Для досягнення цих цілей за основу проєкту обрано класичну багаторівневу клієнт-серверну архітектуру(рис. 4.1). Такий підхід дає змогу логічно розділити програму на три незалежні частини: візуальний інтерфейс користувача, сервер обробки бізнес-логіки та рівень збереження інформації. Завдяки цьому забезпечується висока автономність модулів. Відповідно, розробник отримує можливість оновлювати, тестувати або розширювати окремі компоненти без ризику порушити стабільну роботу всієї платформи.

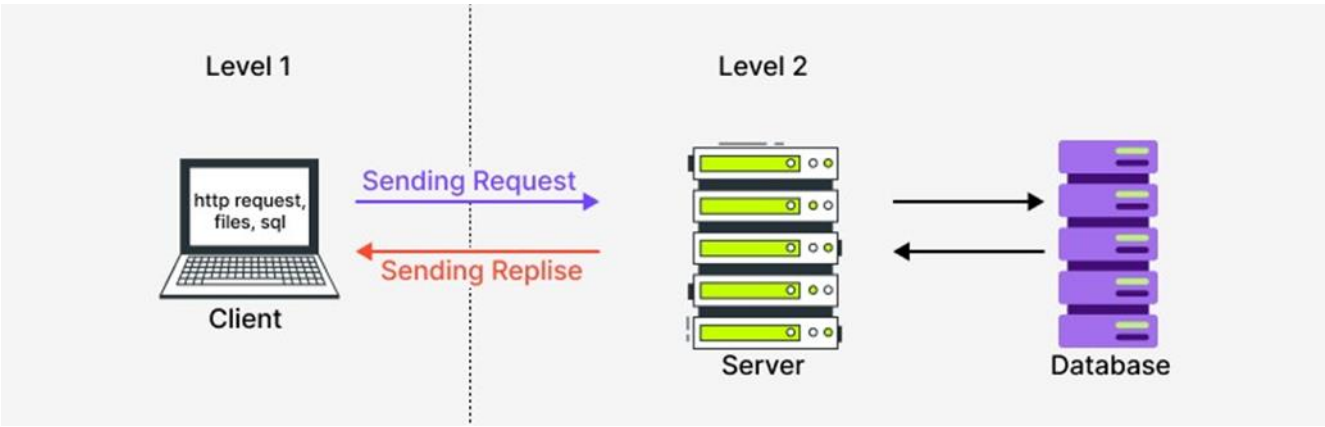

Рисунок 4.1 – Схема багаторівневої архітектури

Ключовою особливістю спроєктованої системи є застосування multi-tenant архітектури. Оскільки створений освітній портал має одночасно обслуговувати багато незалежних шкіл, критично важливо реалізувати надійний механізм ізоляції їхньої інформації. У межах цієї моделі кожен навчальний заклад отримує власний закритий робочий простір із чітким розмежуванням прав доступу. Щоб забезпечити максимальну конфіденційність академічних даних оцінок, розкладу та особистих справ учнів, архітектура використовує суворе розділення інформації

безпосередньо на рівні бази даних. Такий метод гарантує, що записи різних шкіл ізольовані одне від одного, що повністю виключає ризик випадкового витоку інформації та відповідає актуальним вимогам безпеки.

Для реалізації back-end частини обрано сучасний фреймворк Laravel, який функціонує на базі мови програмування PHP. Вибір цього інструментарію зумовлений високою швидкістю розробки та наявністю великої кількості вбудованих компонентів, що дозволяє уникнути написання базового функціоналу з нуля. Платформа надає готові та надійні інструменти для маршрутизації HTTP-запитів, керування сесіями та автентифікації користувачів із різними ролями та правами доступу. Особлива увага при виборі Laravel була приділена питанням безпеки, оскільки освітній портал оперує конфіденційними даними. Фреймворк містить автоматичні механізми захисту від найбільш поширених мережевих вразливостей, зокрема інструменти для підробки міжсайтових запитів (CSRF-токени) та автоматичне екранування вхідних даних для запобігання SQL-ін’єкціям, що суттєво підвищує загальний рівень захищеності програмного продукту.

Невід'ємною частиною обраного серверного стеку є об'єктно-реляційне відображення Eloquent ORM, яке реалізує архітектурний патерн Active Record. Використання Eloquent ORM дозволяє повністю абстрагуватися від написання прямих SQL-запитів до бази даних, замінюючи їх об'єктно-орієнтованим синтаксисом. Це не лише прискорює процес програмування, а й робить код більш чистим, гнучким та придатним для подальшого супроводження. У контексті розробки освітньої системи цей інструмент має критично важливе значення для моделювання та підтримки складних багаторівневих взаємозв'язків між сутностями. Завдяки вбудованим методам опису зв'язків, серверна частина платформи здатна ефективно та швидко оперувати залежностями в ланцюжках даних, гарантуючи цілісність інформації та високу швидкість обробки запитів.

Важливим архітектурним рішенням під час проєктування системи стала використання інструментарію Inertia.js, який виконує функцію сполучної ланки між back-end та front-end частинами. Використання цієї технології дозволяє реалізувати повноцінний вебзастосунок односторінкового типу, зберігаючи при

цьому звичну серверну маршрутизацію. Такий підхід дає змогу контролерам серверної частини передавати дані безпосередньо у front-end компоненти. Це суттєво зменшує загальний обсяг коду, звільняє систему від необхідності підтримки складної клієнтської логіки керування станом і прискорює загальний цикл розробки без втрати кінцевої продуктивності продукту.

Для побудови візуального інтерфейсу обрано JavaScript-фреймворк Vue.js. Його фундаментальна перевага полягає у компонентній архітектурі: весь користувацький інтерфейс декомпозується на ізольовані та незалежні блоки, такі як таблиці успішності, форми завантаження домашніх завдань чи навігаційні панелі. Ці компоненти підлягають багаторазовому використанню, що полегшує підтримку та масштабування інтерфейсу. Реактивна природа Vue.js гарантує миттєве оновлення інформації на екрані при зміні вхідних даних без повного перезавантаження вебсторінки. Це формує сучасний, інтерактивний та безперервний користувацький досвід, що є необхідною умовою для комфортної роботи учнів та викладачів з освітнім порталом.

Враховуючи високі вимоги до надійності, структурованості та безпеки академічних даних, у ролі системи керуванням базами даних обрано Microsoft SQL Server. Специфіка функціонування освітнього закладу передбачає роботу з чітко регламентованими сутностями (навчальні плани, профілі користувачів, журнали успішності) та розгалуженими зв'язками між ними.

MS SQL Server забезпечує високий рівень продуктивності та повністю відповідає принципам транзакційної цілісності ACID. Це гарантує безпомилковість операцій під час паралельного доступу до бази, наприклад, коли кілька викладачів одночасно оновлюють оцінки. Крім того, потужний функціонал цієї СКБД дозволяє надійно реалізувати політику розмежування доступу та механізми логічної ізоляції записів для кожного окремого навчального закладу в межах multu-tenancy архітектури спроєктованої SaaS-платформи.

Розгортання серверної частини платформи здійснюється на базі операційної системи Ubuntu Server із застосуванням технології контейнеризації Docker, що гарантує надійну ізоляцію компонентів та абсолютну ідентичність середовищ

розробки й виконання. Для ефективної обробки мережевих запитів і маршрутизації трафіку використовується високопродуктивний вебсервер Nginx. Автоматизація життєвого циклу проєкту, включаючи тестування, збірку та безпечний реліз оновлень без зупинки роботи порталу, реалізована за допомогою CI/CD платформи GitHub Actions.

4.2 Проєктування реляційної моделі даних та специфікація сутностей

Проєктування ядра інформаційної системи вимагає жорсткої структури, де дані гарантовано ізольовані, а зв'язки між ними залишаються абсолютно прозорими. Фундаментом SaaS-платформи слугує реляційна модель бази даних, управління якою здійснюється через ORM Eloquent, де кожна таблиця має власний клас, що диктує правила обробки інформації, інкапсулює атрибути, захищає їх від масового присвоєння та формує граф об'єктів без написання сирих SQL-запитів.

Ключові сутності платформи, їх атрибути та логіка:

– User. Базовий вузол авторизації, який зберігає фундаментальні атрибути користувача, такі як ім'я, електронна пошта та роль, причому пароль хешується автоматично на рівні моделі через масив $casts, а токени доступу генеруються вбудованим трейтом HasApiTokens. Від цього ядра логічно відгалужуються конкретні профілі через декларативні методи зв'язків hasOne, формуючи чітку ієрархію між загальним акаунтом та конкретною роллю вчителя, учня або батька.

– Teacher. Профіль викладача характеризується жорстко заданим первинним ключем teacher_id із вимкненим автоінкрементом, що гарантує точну ідентифікацію працівника в системі. Атрибути фіксують спеціалізацію та дату найму, тоді як номер телефону шифрується безпосередньо під час запису завдяки кастомізації масиву $casts, що надійно захищає персональні дані.

– Student. Цифровий профіль учня не лише містить базові атрибути як дати народження та зашифрованого номера телефону, але й виступає вузлом зв'язків. Метод parents() реалізує зв'язок «багато до багатьох» із сутністю SchoolParent через проміжну зведену таблицю parent_student, що дозволяє одному

учню мати кількох опікунів. Водночас метод grades() формує відношення «один до багатьох», відкриваючи прямий доступ до колекції всіх оцінок використовуючи student_id, а зворотний зв'язок belongsTo у методах schoolClass() та user() жорстко фіксує належність учня до конкретного класу та глобального облікового запису.

– Grade. Академічна оцінка фіксує точний контекст навчального процесу, зберігаючи сам бал, тип роботи, дату отримання та коментар викладача. Для забезпечення абсолютної ідентифікації модель використовує чотири методи belongsTo. Завдяки цьому кожна оцінка через відповідні зовнішні ключі прив'язується до одного конкретного учня, однієї дисципліни, конкретного місця в розкладу та виконаного домашнього завдання.

– SchoolClass. Організаційна одиниця зберігає назву, паралель та керує розкладом дзвінків, виступаючи центром перетину багатьох сутностей. Модель включає в себе колекцію учнів через зв'язок hasMany, закріплює за собою класного керівника через belongsTo та прив'язує навчальні дисципліни за допомогою зведеної таблиці відношенням belongsToMany. Складна бізнес-логіка часових рамок реалізована в методі getLessonTime(), який спочатку перевіряє наявність ручних перевизначень графіка через зв'язок hasMany (bellOverrides), і лише за їх відсутності звертається до стандартного шаблону через belongsTo (bellTemplate).

– Schedule. Запис в розкладі фіксують день тижня та номер уроку, деталізуючи контекст навчального процесу через зворотні зв'язки belongsTo до сутностей класу та дисципліни. Це гарантує, що кожен елемент розкладу через поля class_id та subject_id належить лише одному класу і стосується одного предмета. При цьому модель динамічно додає віртуальний атрибут time_range під час серіалізації в JSON, метод getTimeRangeAttribute() делегує обчислення точного часу уроку батьківському об'єкту класу, що дозволяє системі безшовно інтегрувати дані та миттєво відображати їх у клієнтському інтерфейсі без додаткових запитів до бази.

Спроєктована архітектура моделей ліквідує потребу в звичайних SQL-запитах завдяки декларативному опису зв'язків. Класи перестають бути пасивними сховищами даних. Вони перетворюються на активні системні вузли, які

інкапсулюють бізнес-логіку, керують шифруванням та гарантують транзакційну цілісність.

Практична реалізація реляційної структури наведена у додатку А. Ця модель керує життєвим циклом домашніх завдань, жорстко пов'язуючи навчальний розклад, прикріплені файли та успішність учнів.

Архітектура класу Homework базується на строгій відповідності фізичній моделі даних. Властивості $table та $primaryKey примусово перевизначають типову поведінку Laravel. Вони встановлюють явну прив'язку до існуючої таблиці homework та її первинного ключа homework_id. Захист цілісності інформації реалізовано через масив $fillable. Цей інструмент жорстко регламентує перелік атрибутів, дозволених для масового заповнення. Будь-які сторонні параметри із запиту автоматично ігноруються рівнем абстракції. Метод casts() забезпечує необхідну типізацію. Він трансформує текстовий формат поля deadline у повноцінний об'єкт дати під час читання запису з бази.

Ключова відповідальність моделі полягає у формуванні графа зв'язків за допомогою декларативних методів Eloquent. Відношення типу belongsTo визначають структурні залежності об'єкта. Звернення до schoolClass(), subject(), creator() та schedule() ідентифікує точний контекст завдання: класс, предмет, автора та місце в розкладі. Водночас відношення hasMany формують ієрархічну підпорядкованість. Методи submissions(), files() та grades() перетворюють модель на центральний вузол. Один об'єкт класу містить посилання на всі завантажені матеріали, здані роботи учнів та виставлені бали.

Представлена реалізація інкапсулює складну реляційну логіку всередині об'єктно-орієнтованого інтерфейсу. Абстракція ORM повністю виключає необхідність конструювання багаторядкових SQL-запитів для виконання операцій об'єднання.

4.3 Проєктування контролерів та маршрутизація запитів

Після проєктування структури бази даних та моделей, наступним критичним етапом розробки є реалізація бізнес-логіки системи та механізмів обробки

користувацьких запитів. За виконання цих завдань відповідають підсистема маршрутизації (routing) та контролери (controllers).

Маршрутизація виступає єдиною точкою входу для всіх HTTP-запитів, зіставляючи URL-адреси з відповідними методами контролерів. У контексті розроблюваного багаторівневого SaaS-додатку маршрутизатор також відіграє важливу роль у розподілі доступу, використовуючи middleware для ідентифікації поточного орендаря (навчального закладу) та перевірки прав користувача.

Контролери, у свою чергу, отримують валідовані запити, взаємодіють з моделями бази даних для отримання або модифікації інформації та формують фінальну відповідь. Головною особливістю системи є використання бібліотеки Inertia.js, яка суттєво змінює підхід до формування відповідей. Замість рендерингу традиційних HTML-шаблонів або побудови повноцінного REST API, контролери повертають об'єкти Inertia. Це дозволяє безпосередньо передавати серверні дані у компоненти клієнтського інтерфейсу на базі Vue.js, забезпечуючи високу швидкість та плавність роботи системи за принципом Single Page Application (SPA).

Реалізація створення нового ізольованого простору (орендаря) виконується завдяки RegisterTenantController. Головною архітектурною особливістю методу store є чітке розмежування між роботою з центральною базою даних та базою даних конкретного орендаря. Після успішної валідації вхідних параметрів та перевірки унікальності ідентифікатора за допомогою правила unique:tenants,id, система створює запис сутності Tenant та реєструє відповідне доменне ім'я. Лістинг коду коду RegisterTenantController додано нижче.

```
class RegisterTenantController extends Controller
{
  public function store(Request $request)
    {
        $rules = [
            'school_name' => 'required|string|max:255',
            'domain' => 'required|string|max:63|unique:tenants,id|regex:/^[a-zA-
Z0-9]+$/',
            'email' => 'required|string|email|max:255',
            'password' => ['required', 'confirmed', Password::defaults()],
        ]
        $messages = [
            'domain.unique' => 'Школа з таким доменом (посиланням) вже існує! Будь
ласка, придумайте інший.',
            'domain.regex' => 'Домен може містити лише латинські літери та цифри,
без пробілів та спецсимволів.',
            'domain.required' => 'Вкажіть домен для школи.',
```

```
            'school_name.required' => 'Назва школи є обов\'язковою.',
            'email.required' => 'Введіть email адресу адміністратора.',
            'password.confirmed' => 'Паролі не співпадають. Спробуйте ще раз.',
        ];
        $request->validate($rules, $messages);

        $fullDomain = $request->domain . '.' . 'localhost';

        $tenant = Tenant::create([
            'id' => $request->domain,
            'data' => [
                'school_name' => $request->school_name,
            ]
        ]);

        $tenant->domains()->create([
            'domain' => $fullDomain
        ]);

        $tenant->run(function () use ($request) {
            User::create([
                'first_name' => 'Іван',
                'last_name' => 'Іванов',
                'role' => 'admin',
                'email' => $request->email,
                'password' => Hash::make($request->password),
            ]);
        });
        return Inertia::location("http://" . $fullDomain . ":8000");
    }
}
```

Для забезпечення ізоляції даних створення облікового запису первинного адміністратора закладу виконується всередині замикання методу $tenant->run(). Цей підхід дозволяє динамічно переключити контекст з'єднання з базою даних, гарантуючи, що інформація про адміністратора буде записана виключно в локальну таблицю users відповідного орендаря, без ризику витоку даних у загальний стек системи. Завершення процесу реєстрації супроводжується використанням методу Inertia::location(), що забезпечує коректне перенаправлення SPA-додатка на новостворений субдомен для подальшої авторизації.

Для забезпечення безшовної інтеграції між серверною частиною на базі фреймворку Laravel та клієнтським SPA на базі Vue.js відкинуто класичний підхід з використанням Blade-шаблонів на користь технології Inertia.js. Організація передачі даних у розроблених контролерах базується на виклику методу Inertia::render(), який приймає назву відповідного фронтенд-компонента та props.

Наведені лістинги коду демонструють два основні сценарії взаємодії. У випадку UserController реалізовано передачу динамічної колекції користувачів із

вбудованою пагинацією та збереженням стану фільтрації (withQueryString), що дозволяє клієнтському компоненту миттєво реагувати на дії адміністратора без перезавантаження сторінки. Лістинг коду методу index класа UserController наведено нижче.

```
public function index(Request $request)
    {
        $query = User::with(['teacher', 'student.schoolClass', 'parent']);

        if ($request->has('role') && $request->role !== 'all') {
            $query->where('role', $request->role);
        }

        $users = $query->orderBy('created_at', 'desc')
                       ->paginate(15)
                       ->withQueryString();

        return Inertia::render('Tenant/Dashboard/Admin/Users/Index', [
            'users' => $users,
            'filters' => $request->only('role'),
        ]);
    }
```

У ScheduleController продемонстровано підхід комплексної агрегації даних. Контролер формує складний багатокомпонентний об'єкт відповіді, який містить розклад, масиви оцінок та домашніх завдань, адаптовані під конкретну роль авторизованого користувача. Такий підхід дозволяє уникнути проєктування громіздкого REST API та забезпечує збереження монолітної структури розробки при отриманні всіх переваг сучасного SPA-інтерфейсу. . Лістинг коду частини методу index класа ScheduleController наведено нижче.

```
return Inertia::render('Tenant/Dashboard/Schedule/Index', [
            'schedule' => $schedules,
            'classes' => $allClasses,
            'currentClass' => $currentClass,
            'userRole' => $user->role,
            'grades' => $grades,
            'homeworks' => $homeworks,
            'parentChildren' => $parentChildren,
            'currentStudentId' => $studentId,
        ]);
```

Процес отримання та обробки даних на стороні клієнтського інтерфейсу реалізується за допомогою макросу defineProps. Наведений лістинг ілюструє прийом комплексного об'єкта відповіді від серверної частини додатка. Головною архітектурною перевагою такого підходу є те, що props, сформовані на сервері, автоматично трансформуються в реактивні властивості компонента. Лістинг коду клієнтської частини коду , продемонстровано нижче.

```
<script setup>
const props = defineProps({
    schedule: Object,
    currentClass: Object,
    userRole: String
    ...
});
const lessonSlots = [1, 2, 3, 4, 5, 6, 7, 8];
const getLesson = (dayKey, lessonNumber) => {
    const dayLessons = props.schedule[dayKey];
    if (!dayLessons) return null;

    return dayLessons.find(lesson => lesson.lesson_number == lessonNumber);
};
...
</script>
<template>
...
    <div class="diary-container">
        <h2>Розклад для класу: {{ currentClass?.name }}</h2>

        <div v-for="num in lessonSlots" :key="num" class="lesson-row">
            <span class="lesson-number">{{ num }}</span>

            <span class="subject-name">
                {{ getLesson('monday', num)?.subject?.name || 'Немає уроку' }}
            </span>
        </div>
    </div>
...
</template>
```

Для динамічного мапінгу даних у сітку інтерфейсу (шаблон щоденника) використовується допоміжна функція getLesson. Як показано в секції <template>, цей метод викликається безпосередньо всередині ітераційного циклу v-for. Він виконує пошук відповідного запису в реактивному масиві за ключем дня тижня та порядковим номером уроку. Якщо урок існує, система миттєво рендерить його параметри (наприклад, subject.name), в іншому випадку виводиться значення за замовчуванням. Такий підхід забезпечує високу швидкість роботи інтерфейсу.

4.4 Реалізація підсистеми маршрутизації та розмежування доступу

У мультитенантних SaaS-системах маршрутизатор виконує складну функцію диспетчеризації на рівні доменів. Оскільки розроблена платформа одночасно обслуговує публічний сайт сервісу та безліч ізольованих навчальних закладів, конфігурація маршрутів фізично розділена на два незалежних потоки: центральні маршрути (routes/web.php) та маршрути орендарів (routes/tenant.php).

Центральна маршрутизація відповідає виключно за головний домен платформи. Як показано в лістингу який наведено нижче, використовується цикл для перебору дозволених центральних доменів із конфігурації. У цій зоні відсутні захищені персональні дані. Тут розташовані публічні сторінки (лендінг) та єдина критична точка входу – POST-запит на створення нового навчального закладу (RegisterTenantController), після виконання якого користувач маршрутизується вже на власний субдомен.

```
foreach(config('tenancy.central_domains') as $domain) {
    Route::domain($domain)->group(function () {
        Route::get('/', [PageController::class, 'homeService'])->name('home');

        Route::get('/how-it-works', [PageController::class, 'howItWorks'])-
>name('how-it-works');
        Route::get('/pricing', [PageController::class, 'pricing'])-
>name('pricing');

        Route::post('/register-tenant', [RegisterTenantController::class,
'store'])->name('tenant.register');

        Route::fallback(function () {
            return Inertia::render('Landing/Home');
        })->name('error');
    });
}
```

Обробка запитів усередині зареєстрованих шкіл (субдоменів) делегована файлу routes/tenant.php. Лістинг коду фрагмента маршутизації орендаря наведено нижче. Ключовим елементом безпеки тут є застосування класу InitializeTenancyByDomain. Цей middleware автоматично перехоплює HTTP-запит, аналізує його субдомен (наприклад, school1.localhost), ідентифікує орендаря та динамічно перемикає з'єднання з центральної бази даних на ізольовану локальну базу конкретної школи. Додатковий клас PreventAccessFromCentralDomains блокує будь-які спроби звернутися до шкільного функціоналу з головного домену.

```
Route::middleware([
    'web',
    InitializeTenancyByDomain::class,
    PreventAccessFromCentralDomains::class,
])->group(function () {
    ...
    Route::get('/', [PageController::class, 'publicSchoolPage'])->name('public-
home');
    ...
    Route::middleware('auth')->group(function () {
        Route::prefix('portal')->group(function () {
            ...
```

```
            Route::get('/dashboard', [HomeDashboardController::class, 'index'])->name('dashboard');

            Route::controller(UserController::class)->prefix('users')->group(function () {
                Route::get('/', 'index')->name('users.index');
                Route::post('/', 'store')->name('users.store');
            });

            Route::controller(TeacherHomeworkController::class)->prefix('teacher')->group(function () {
                Route::get('/homeworks', 'index')->name('teacher.homeworks.index');
            });

            Route::controller(StudentHomeworkController::class)->prefix('student')->group(function () {
                Route::get('/homework', 'index')->name('student.homework.index');
            });
            ...
        });
    });
});
```

Базовий middleware auth відсікає неавторизовані сесії. Далі маршрути організовані в ізольовані рольові групи за допомогою методів prefix() та controller(). Замість створення складних перевірок усередині кожного контролера, система виділяє окремі API-простори: /portal/teacher/... для викладачів, /portal/student/... для учнів та /portal/parent/... для батьків. Такий підхід мінімізує ризик перетину прав доступу і дозволяє контролерам виконувати виключно бізнес-логіку обробки даних.

4.5 Методи та засоби забезпечення якості програмного продукту

У сучасній інженерії програмного забезпечення тестування є важливою частиною життєвого циклу розробки. Складність сучасних вебсистем, особливо таких, що базуються на архітектурі мультитенантності, вимагає гарантій того, що будь-яка зміна в одному модулі системи не призведе до появи нових помилок.

Feature-тестування це підхід перевірки працездатності окремих функціональних можливостей програмного забезпечення з метою підтвердження їхньої відповідності встановленим технічним вимогам. На відміну від модульного тестування, цей підхід аналізує функціонал у контексті взаємодії всіх рівнів системи, включаючи обробку користувацьких запитів та бізнес-логіку. Головною

метою цього виду тестування є гарантія того, що впроваджені зміни або новий функціонал працюють коректно, не створюють критичних помилок.

Застосуванням цього підходу є перевірка важливого процесу – реєстрації нових навчальних закладів. Тест TenantOnboardingTest імітує повний цикл створення орендаря у системі. Лістинг коду тесту, наведено нижче. Він виконує запит, аналогічний діям користувача, а потім верифікує ключові результати: наявність запису про новий заклад у центральній базі даних, автоматичне налаштування окремого домену та коректну ініціалізацію облікового запису адміністратора всередині ізольованої бази даних конкретного закладу.

```
class TenantOnboardingTest extends TestCase
{
    ...
    public function test_new_school_can_register_and_admin_is_created(): void
    {
        $domain = strtolower(fake()->unique()->lexify('??????school'));
        $email = fake()->unique()->safeEmail();

        $payload = [
            'school_name' => fake()->company() . ' School',
            'domain' => $domain,
            'email' => $email,
            'password' => 'SecurePass123!',
            'password_confirmation' => 'SecurePass123!',
        ];

        $response = $this->post('http://localhost/register-tenant', $payload);

        $response->assertSessionHasNoErrors();
        $response->assertStatus(302);

        $this->assertDatabaseHas('tenants', [
            'id' => $domain,
        ]);

        $this->assertDatabaseHas('domains', [
            'domain' => $domain . '.localhost',
            'tenant_id' => $domain,
        ]);

        $tenant = Tenant::find($domain);

        $tenant->run(function () use ($email) {
            $this->assertDatabaseHas('users', [
                'email' => $email,
                'role' => 'admin',
            ]);
        });

    }
}
```

Використання методу $tenant->run() гарантує, що процедура налаштування виконується саме в межах відокремленого середовища, що є обов’язковою

вимогою для безпеки мультитенантної архітектури. Успішне проходження тесту підтверджує автоматизацію процесу розгортання та надійність розділення даних між школами

Наступним етапом функціонального тестування є перевірка механізмів розмежування прав доступу користувачів до ресурсів системи. Тест RoleAccessTest забезпечує контроль безпеки, підтверджуючи, що користувачі можуть взаємодіяти лише з тими розділами порталу, які передбачені їхніми ролями. У межах цього тесту реалізовано два сценарію. Сценарій test_student_cannot_access_teacher_routes перевіряє захист від несанкціонованого втручання, підтверджуючи, що спроба учня перейти до вчительського інтерфейсу призводить до блокування запиту з кодом помилки 403 Forbidden. Лістинг коду сценарію наведено нижче

```
public function test_student_cannot_access_teacher_routes(): void
    {
        $tenantId = 'test-school-' . Str::random(5);
        $domain = $tenantId . '.localhost';

        $tenant = Tenant::create(['id' => $tenantId]);
        $tenant->domains()->create(['domain' => $domain]);

        $tenant->run(function () use ($domain) {
            $student = User::factory()->student()->create();
            $response = $this->actingAs($student)-
>get("http://{$domain}/portal/teacher/homeworks");
            $response->assertStatus(403);
        });
    }
```

Сценарій test_teacher_can_access_teacher_routes демонструє, що авторизований викладач отримує безперешкодний доступ до професійного розділу зі статусом 200 OK. Використання методу actingAs дозволяє точно відтворити поведінку кожного користувача, що в результаті доводить ефективність механізму розмежування ролей та гарантує захист даних у системі. Лістинг коду сценарію наведено нижче.

```
public function test_teacher_can_access_teacher_routes(): void
    {
        $tenantId = 'test-school-' . Str::random(5);
        $domain = $tenantId . '.localhost';

        $tenant = Tenant::create(['id' => $tenantId]);
        $tenant->domains()->create(['domain' => $domain]);

        $tenant->run(function () use ($domain) {
            $teacher = User::factory()->teacher()->create();
            $response = $this->actingAs($teacher)-
>get("http://{$domain}/portal/teacher/homeworks");
```

```
            $response->assertStatus(200);
        });
    }
```

Важливим аспектом мультитенантної архітектури є забезпечення суворої ізоляції даних між різними орендарями (навчальними закладами). Витік даних між орендарями є неприпустимим порушенням безпеки, тому система повинна гарантувати, що обліковий запис користувача, зареєстрований в одному закладі, не може бути використаний для автентифікації в іншому, навіть за збігу ідентифікаторів. Для перевірки цього рівня безпеки розроблено CrossTenantLeakageTest. Лістинг коду цього тесту наведено у додатку Б.

Тест моделює спробу несанкціонованого доступу, створюючи два незалежні середовища – «школу А» та «школу Б». У базі даних «школи Б» створюється обліковий запис користувача, після чого здійснюється спроба автентифікації цього користувача через портал «школи А». Використовуючи метод assertSessionHasErrors, система перевіряє, чи відхиляє запит автентифікації через невідповідність контексту домену, а assertDatabaseMissing підтверджує відсутність даних про користувача з іншої бази в поточному середовищі. Таким чином, тест гарантує, що механізм автентифікації жорстко прив’язаний до контексту поточного орендаря, гарантуючи повну цілісність даних

Наступним етапом є перевірки коректності політик доступу до функціоналу виставлення оцінок. Тест TeacherGradingLogicTest забезпечує перевірку надійності розмежування прав доступу між викладачами. Лістинг коду даного тесту наведено в додатку В Його логіка базується на моделюванні спроби несанкціонованого оцінювання, коли один викладач намагається виставити оцінку за роботу, створену іншим колегою.

В ході тестування перевіряються два сценарії. Спроба стороннього викладача вплинути на оцінювання має бути заблокована системою з кодом 403 Forbidden, тоді як для автора завдання, система повинна успішно опрацювати запит. Використання методу actingAs дозволяє точно імітувати дії різних користувачів, підтверджуючи, що політики авторизації жорстко контролюють повноваження в межах навчальних дисциплін, що унеможливлює неправомірне втручання в процес виставлення оцінок.

Unit-тестування це перевірки працездатності найменших ізольованих компонентів коду, таких як окремі функції або методи класу. Такий підхід дозволяє виявити логічні помилки на ранніх етапах розробки та гарантує коректність роботи кожного модуля до його інтеграції в основний програмний комплекс.

Клас FileCategorizerTest призначений для модульної верифікації сервісу FileCategorizer, який відповідає за автоматичну класифікацію файлів за їхнім розширенням. Основна мета цього тесту, забезпечити коректність роботи алгоритму визначення типу контенту в ізольованому середовищі. Лістинг коду методу наведено нижче. Метод test_it_categorizes_known_extensions, демонструє базовий сценарій перевірки. Він порівнює фактичний результат роботи методу getType() з очікуваним значенням для стандартних форматів (наприклад, для pdf очікується document).

```
public function test_it_categorizes_known_extensions(): void
    {
        $this->assertEquals('document', $this->categorizer->getType('pdf'));
        $this->assertEquals('image', $this->categorizer->getType('PNG'));
        $this->assertEquals('spreadsheet', $this->categorizer->getType('xlsx'));
    }
```

Інші методи тестування, зокрема test_it_handles_new_formats та test_it_returns_other_for_unknown_extensions, функціонують за аналогічним принципом. Вони перевіряють здатність системи класифікувати специфічні формати (архіви, електронні книги) та коректно обробляти невідомі типи файлів, повертаючи значення за замовчуванням.

Клас HomeworkStatusEngineTest призначений для модульної перевірки логіки визначення поточного статусу домашнього завдання. Його основна мета – гарантувати, що сервіс, який аналізує наявність зданої роботи, виставленої оцінки та термінів виконання, коректно призначає завданням статуси (graded, pending, overdue, unsubmitted). Метод test_it_returns_graded_when_grade_exists, перевіряє, що за наявності виставленої оцінки завдання отримує статус graded, незалежно від інших параметрів. Лістинг коду методу наведено нижче. Метод імітує створення об'єкта оцінки та через сервіс перевіряє відповідність результату очікуваному значенню, що підтверджує коректність побудованого алгоритму пріоритезації станів.

```
public function test_it_returns_graded_when_grade_exists(): void
    {
        $grade = new Grade();
        $this->assertSame('graded', $this->engine->getStatus(null, $grade, null));
    }
```

Інші методи тесту, test_it_returns_pending_when_submission_exists, test_it_returns_overdue_when_deadline_passed та test_it_returns_unsubmitted_when_deadline_is_future, функціонують за аналогічним принципом. Вони перевіряють решту можливих станів завдання, використовуючи різні комбінації даних, зокрема перевірку дати дедлайну за допомогою бібліотеки Carbon та наявність файлів зданої роботи. Такий підхід дозволяє повністю покрити тестами всі етапи життєвого циклу завдання, гарантуючи, що система відображає актуальний стан виконання робіт для будь-якої ситуації.

Підсумовуючи, впроваджена стратегія тестування забезпечує комплексний підхід до контролю якості програмного продукту. Поєднання модульних та функціональних тестів для контролю критичних сценаріїв дозволило створити надійну архітектуру, де кожна зміна коду є контрольованою та прогнозованою.

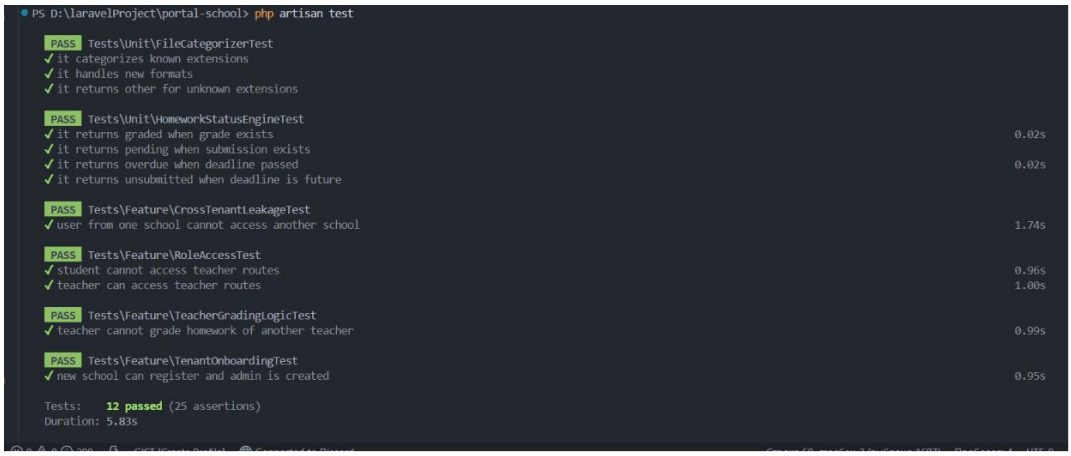

Рисунок 4.2 – Результат тестування

На рисунку 4.2 наведено звіт про виконання тестів підтверджує успішну верифікацію ключових функціональних вузлів системи. Сумарно було виконано 12 тестових сценаріїв, що охоплюють 25 перевірок (assertions), кожна з яких завершилася успішно.

4.6 Інструкція користувача

Програмний комплекс розроблений для автоматизації освітнього процесу в навчальних закладах. Система працює в режимі вебдодатку, тому для початку роботи не потрібно встановлювати додаткове програмне забезпечення, достатньо сучасного браузера (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) та підключення до мережі Інтернет.

Дії для зовнішніх користувачів описуються в таблиці 4.1, де описується початковий шлях користувача до взаємодії з платформою: від ознайомлення з можливостями порталу до реєстрації власного навчального закладу та авторизації.

Таблиця 4.1 – Послідовність дій зовнішнього користувача

Етап	**Дія користувача**	**Результат**	**Посилання**
Ознайомлення	Відкриття головної сторінки порталу	Відображення переліку можливостей та переваг платформи	Рис. Г.1
Реєстрація	Заповнення форми створення школи	Розгортання ізольованого середовища для нового закладу	Рис. Г.2
Перехід	Введення персонального домену школи	Відкриття публічної сторінки (цифрової візитки)	Рис. Г.3
Авторизація	Введення облікових даних у форму входу	Вхід до закритої частини та відкриття кабінету	Рис. Г.4

Робота адміністратора навчального закладу, з внутрішніми даними платформи після автентифікації, що дозволяє налаштувати структуру навчального року та забезпечити функціонування системи, описуються в таблиці 4.2

Таблиця 4.2 – Послідовність дій адміністратора в системі

Етап	**Дія користувача**	**Результат**	**Посилання**

Кінець таблиці 4.2

Вхід	Перша авторизація та ознайомлення з інтерфейсом	Візуальний моніторинг стану системи та доступ до ключових розділів	Рис. Г.5
Налаштування	Керування довідниками (додавання користувачів, класів, предметів, розкладу, новин, дзвінків та налаштувань закладу)	Формування цілісної структури навчального року	Рис. Г.6, Рис. Г.7, Рис. Г.8, Рис. Г.9, Рис. Г.10, Рис. Г.11, Рис. Г.12, Рис. Г.13,

Після авторизації в системі вчитель отримує доступ до інструментарію, необхідного для ведення навчальної документації. Роль вчителя передбачає активну роботу з академічними показниками учнів та наповнення контентом навчальних дисциплін, описуються в таблиці 4.3.

Таблиця 4.3 – Послідовність дій вчителя в системі

Етап	**Дія користувача**	**Результат**	**Посилання**
Доступ	Авторизація та перегляд персонального кабінету	Відображення актуального розкладу викладача	Рис. Г.14
Проведення уроку	Відкриття журналу поточного уроку, відмітка відсутніх	Фіксація присутності та початок роботи в класі	Рис. Г.15, Рис. Г. 16
Журнал	Робота з класом та оцінювання учнів	Фіксація навчальних досягнень у базі даних	Рис. Г.17, Рисс. Г. 18
ДЗ та перевірка	Публікація завдань та перевірка робіт	Забезпечення зворотного зв'язку з учнями	Рис. Г.19
Звітність	Створення та перегляд аналітичних звітів	Аналіз динаміки успішності та навчального процесу	Рис., Г.20

Після авторизації учень отримує доступ до власного інформаційного простору, який дозволяє відстежувати академічну успішність, планувати час

виконання завдань та комунікувати з викладачами через систему, описуються в таблиці 4.4.

Таблиця 4.4 – Послідовність дій учня в системі

Етап	**Дія користувача**	**Результат**	**Посилання**
Доступ	Перегляд особистого кабінету та розкладу	Відображення навчального навантаження на тиждень	Рис. Г.21
Навчання	Перегляд домашніх завдань та навчальних матеріалів	Отримання необхідної інформації для опрацювання тем	Рис. Г.22, Рис. Г.23, Рис. Г.24
Зворотний зв'язок	Надсилання виконаних завдань на перевірку	Надання вчителю результатів самостійної роботи	Рис. Г.25
Успішність	Аналіз оцінок та звітів про прогрес	Визначення рівня власних навчальних досягнень	Рис. Г.26

Батьки мають можливість здійснювати віддалений моніторинг навчального процесу своєї дитини, що забезпечує прозорість взаємодії між сім'єю та закладом освіти, описується в таблиці 4.5.

Таблиця 4.5 – Послідовність дій батьків в системі

Етап	**Дія користувача**	**Результат**	**Посилання**
Доступ	Авторизація та вибір дитини з профілю	Відображення навчального профілю учня	Рис. Г.27, Рис. Г.28 Рис. Г.29
Контроль	Перегляд поточних оцінок та відвідуваності	Розуміння поточної академічної успішності	Рис. Г.30 Рис. Г.31
Навантаження	Перегляд домашніх завдань та розкладу	Координація часу дитини на виконання завдань	Рис. Г.32

Наведений комплекс інструкцій забезпечує зручне та структуроване управління освітнім процесом для всіх учасників системи: від адміністратора до батьків. Завдяки чіткому розмежуванню прав доступу та логічному розподілу функціоналу, портал стає ефективним інструментом для цифровізації школи, мінімізуючи час на виконання адміністративних завдань та підвищуючи прозорість навчання.

Висновки до розділу 4

У межах четвертого розділу успішно реалізовано освітню SaaS-платформу. Використання стеку Laravel, Inertia.js та Vue.js дозволило створити високоефективний SPA-застосунок, а архітектура мультитенантності забезпечила надійну ізоляцію даних між навчальними закладами.

Реалізована фізична модель «Database-per-Tenant» із застосуванням СКБД Microsoft SQL Server гарантувала не лише найвищий рівень безпеки та конфіденційності, але й строгу транзакційну цілісність ACID під час паралельної роботи користувачів. Проєктування бізнес-логіки на основі об'єктно-реляційного відображення Eloquent ORM дозволило абстрагуватися від складних прямих SQL-запитів, що оптимізувало роботу з розгалуженими базами академічних сутностей та суттєво спростило подальший супровід системи.

Ефективна підсистема маршрутизації та багаторівневе розмежування доступу чітко структурували інформаційні потоки платформи. Використання спеціалізованого Middleware для динамічного перемикання з'єднань на основі субдоменів дозволило коректно спрямовувати користувачів відповідно до їхніх ролей та приналежності до конкретного закладу освіти. Додаткова автоматизація інфраструктури за допомогою контейнеризації Docker та налаштування CI/CD процесів забезпечили надійне середовище для розгортання продукту.

Розроблена комплексна стратегія тестування, що поєднує функціональні та модульні перевірки, підтвердила високу надійність програмного продукту. Успішне проходження критичних тестових сценаріїв, від автоматичної реєстрації нового орендаря до перевірки політик доступу при виставленні оцінок, підтвердила механізми ізоляції даних та політики авторизації..

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи створено веборієнтовану інформаційну систему для автоматизації навчального процесу, що є сучасним рішенням для цифровізації закладів загальної середньої освіти. Використання архітектури Multi-tenancy за моделлю SaaS стало ефективним рішенням для подолання фрагментації даних, що дозволяє розгортати ізольовані освітні середовища швидко та безпечно.

Задля досягнення поставленої мети, виконано наступні завдання:

– проведено аналіз предметної області та існуючих аналогів систем управління навчанням;

– обрано технічний стек та архітектурний підхід;

– спроєктовано структуру бази даних, яка передбачає розділення на центральну базу та окремі бази даних для кожної школи;

– реалізовано механізм автоматичної реєстрації, що включає створення субдомену, генерацію бази даних та виконання міграцій таблиць;

– розроблено публічну частину для презентації платформи та публічного сайту для кожної окремої школи;

– розроблено закриту частину з функціоналом для ролей: адміністратор, вчитель, учень, батьки;

– забезпечено захист даних та запобігання несанкціонованому доступу між різними школами.

Практична реалізація системи на базі стека Laravel, Vue.js та Inertia.js успішно пройшла тестування, підтвердивши високу надійність та безпеку завдяки моделі Database-per-Tenant, яка гарантує абсолютну ізоляцію даних кожної школи та унеможливлює витік інформації. Наразі подальший розвиток передбачає створення нативних мобільних застосунків для iOS та Android із підтримкою push-сповіщень, а також інтеграцію інтелектуальних інструментів предиктивної аналітики успішності на базі алгоритмів машинного навчання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Google Classroom Wiki. SitesWiki. URL: https://classroom.siteswiki.org/wiki (Accessed: 29.04.2026).

2. Schoology Learning. PowerSchool. URL: https://www.powerschool.com/solutions/personalized-learning/schoology-learning/ (Accessed: 29.04.2026).

3. Prosvita – сучасний освітній простір. Prosvita. URL: https://prosvita.net/ (дата звернення: 29.04.2026).

4. Djeki E., Dégila J., Bondiombouy C., Alhassan M. H. Data protection in digital learning space: An overview. AIP Conference Proceedings. 2024. Vol. 3109, Iss. 1. URL: https://doi.org/10.1063/5.0204895 (Accessed: 29.04.2026).

5. Unlocking SaaS Scalability: Mastering Multi-Tenant Architecture. Outsourcify. URL: https://outsourcify.net/unlocking-saas-scalability-mastering-multi-tenant-architecture/ (Accessed: 29.04.2026).

6. Reasons to Choose Multi-Tenant SaaS Architecture for Your Applications. Persistent Systems. URL: https://www.persistent.com/blogs/reasons-to-choose-multi-tenant-saas-architecture-for-your-applications/ (Accessed: 29.04.2026).

7. Choosing the right SaaS architecture: Multi-Tenant vs. Single-Tenant. Clerk. URL: https://clerk.com/blog/multi-tenant-vs-single-tenant (Accessed: 29.04.2026).

8. Tak089. Multi-Tenant Architecture: A Complete Guide (Basic to Advanced). DEV Community. 2025. URL: https://dev.to/tak089/multi-tenant-architecture-a-complete-guide-basic-to-advanced-119o (Accessed: 29.04.2026).

9. Multi-Tenant Architecture. Future Processing. URL: https://www.future-processing.com/blog/multi-tenant-architecture/ (Accessed: 29.04.2026).

10. Multi-Tenant vs Single-Tenant Architectures Guide & Comparison. Acropolium. URL: https://acropolium.com/blog/multi-tenant-vs-single-tenant-architectures-guide-comparison/ (Accessed: 29.04.2026).

11. Multi-Tenant Architecture SaaS Guide. Ariel Softwares. URL: https://www.arielsoftwares.com/multi-tenant-architecture-saas-guide/ (Accessed: 29.04.2026).

12. Data Isolation and Sharding Architectures for Multi-Tenant Systems. Medium. URL: https://medium.com/@justhamade/data-isolation-and-sharding-architectures-for-multi-tenant-systems-20584ae2bc31 (Accessed: 29.04.2026).

13. Database-per-tenant Use Case. Atlas. URL: https://atlasgo.io/use-cases/database-per-tenant (Accessed: 29.04.2026).

14. Djeki E., Dégila J., Bondiombouy C., Alhassan M. H. Data protection in digital learning space: An overview. AIP Conference Proceedings. 2024. Vol. 3109, Iss. 1. URL: https://pubs.aip.org/aip/acp/article-abstract/3109/1/030007/3282169/Data-protection-in-digital-learning-space-An (Accessed: 29.04.2026).

15. Alier M., Casañ Guerrero M. J., Amo D., Severance C., Fonseca D. Privacy and E-Learning: A Pending Task. Sustainability. 2021. Vol. 13, No. 16. Article 9206. URL: https://doi.org/10.3390/su13169206 (Accessed: 29.04.2026).

16. Kilit J., Bobin Blychert J. Edge Computing and GDPR: A Technical Security and Legal Compliance Analysis. DiVA Portal [Student thesis, Jönköping University]. 2025. URL: https://www.diva-portal.org/smash/get/diva2:1982107/FULLTEXT01.pdf (Accessed: 29.04.2026).

17. Database Security. Palo Alto Networks. URL: https://www.paloaltonetworks.com/cyberpedia/database-security (Accessed: 29.04.2026).

18. Mirza M. A., Ajay G. P., Hareesh K., Brahmaiah B., Rohith J. TenantX: Enterprise-Grade Multi-Tenant SaaS Platform with Dynamic RBAC and Configurable Workflow Engine. International Journal of Engineering Research & Technology (IJERT). 2026. Vol. 15, Iss. 04. URL: https://doi.org/10.5281/zenodo.19788686 (Accessed: 29.04.2026).

19. GDPR Compliance. Vectra AI. URL: https://www.vectra.ai/topics/gdpr-compliance (Accessed: 29.04.2026).

20. Front-End Frameworks for Development of SPA and MPA Web Applications. ResearchGate. URL: https://doi.org/10.2139/ssrn.3987838 (Accessed: 29.04.2026).

21. Single Page Application vs Progressive Web Apps: A Comparison. Microverse. URL: https://www.microverse.org/blog/single-page-application-vs-progressive-web-apps-a-comparison (Accessed: 29.04.2026).

22. Laravel Inertia: Simplifying Front-end Development in Laravel. Cubet Tech. URL: https://cubettech.com/resources/blog/laravel-inertia-simplifying-front-end-development-in-laravel/ (Accessed: 29.04.2026).

23. Schmalbach V. Laravel + Inertia + Vue vs Node + React. Vincent Schmalbach. URL: https://www.vincentschmalbach.com/laravel-inertia-vue-vs-node-react/ (Accessed: 29.04.2026).

24. Putra F. P. E., Efendi R. W., Tamam A. B., Pramadi W. A. Trends and Best Practices in API-Based Web Development Using Laravel and React. ResearchGate. 2025. URL: https://doi.org/10.47709/brilliance.v5i1.5970 (Accessed: 29.04.2026).

25. Couriol B. Inertia.JS Lets Developers Write API-Free Monolithic React/Vue/Svelte Applications in PHP or Ruby. InfoQ. 2020. URL: https://www.infoq.com/news/2020/12/inertia-modern-monolith/ (Accessed: 29.04.2026).

26. What is Laravel. IGM Guru. URL: %20https:/www.igmguru.com/blog/what-is-laravel (Accessed: 29.04.2026).

27. Craft Your Startup MVP Using the Laravel Vue Stack. Emveep. URL: https://www.emveep.com/blog/craft-your-startup-mvp-using-the-laravel-vue-stack/ (Accessed: 29.04.2026).

28. Aghera D. Laravel Inertia: Developing Modern Single-Page Applications. Concetto Labs. 2025. URL: https://www.concettolabs.com/blog/laravel-inertia/ (Accessed: 29.04.2026).

ДОДАТОК А
Лістинг коду моделі Homework.php

```php
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\Relations\HasOne;
class Homework extends Model
{
    use HasFactory;
    protected $table = 'homework';
    protected $primaryKey = 'homework_id';
   protected $fillable = [
        'subject_id',
        'class_id',
        'schedule_id',
        'created_by',
        'description',
        'file_path',
        'file_name',
        'deadline',
        'created_at'
    ];
    protected function casts(): array
    {
        return [
            'deadline' => 'date',
        ];
    }
    public function submissions(): HasMany
    {
        return $this->hasMany(Submission::class, 'homework_id', 'homework_id');
    }
    public function schoolClass(): BelongsTo
    {
        return $this->belongsTo(SchoolClass::class, 'class_id', 'class_id');
    }
    public function subject(): BelongsTo
    {
        return $this->belongsTo(Subject::class, 'subject_id', 'subject_id');
    }
    public function creator(): BelongsTo
    {
         return $this->belongsTo(User::class, 'created_by', 'id');
    }
    public function schedule()
    {
        return $this->belongsTo(Schedule::class, 'schedule_id', 'schedule_id');
    }
    public function files(): HasMany
    {
        return $this->hasMany(HomeworkFile::class, 'homework_id', 'homework_id');
    }
    public function grades()
    {
        return $this->hasMany(Grade::class, 'homework_id', 'homework_id');
    }
}
```

ДОДАТОК Б
Лістинг коду тесту CrossTenantLeakageTest.php

```php
<?php
namespace Tests\Feature;

use Tests\TestCase;
use App\Models\Tenant;
use App\Models\User;
use Illuminate\Support\Str;
use Illuminate\Foundation\Testing\DatabaseMigrations;

class CrossTenantLeakageTest extends TestCase
{
    use DatabaseMigrations;
     protected function setUp(): void
    {
        parent::setUp();
        $this->withoutVite();
    }
    public function test_user_from_one_school_cannot_access_another_school(): void
    {
        $tenantAId = 'school-' . Str::random(8);
        $tenantBId = 'school-' . Str::random(8);
        $tenantA = Tenant::create([
            'id' => $tenantAId,
        ]);
        $tenantA->domains()->create([
            'domain' => "{$tenantAId}.localhost",
        ]);

        $tenantB = Tenant::create([
            'id' => $tenantBId,
        ]);

        $tenantB->domains()->create([
            'domain' => "{$tenantBId}.localhost",
        ]);

        $tenantB->run(function () {
            User::factory()->student()->create([
                'email' => 'spy@school-b.com',
                'password' => bcrypt('SecurePass123!'),
            ]);
        });

        $response = $this->post(
            "http://{$tenantAId}.localhost/login",
            [
                'email' => 'spy@school-b.com',
                'password' => 'SecurePass123!',
            ]
        );
        $response->assertSessionHasErrors('email');
        $this->assertGuest();
        $tenantA->run(function () {
            $this->assertDatabaseMissing('users', [
                'email' => 'spy@school-b.com',
            ]);
        });
    }
}
```

ДОДАТОК В
Лістинг коду тесту TeacherGradingLogicTest.php

```php
<?php

namespace Tests\Feature;

use Tests\TestCase;
use App\Models\Tenant;
use App\Models\User;
use App\Models\Homework;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Support\Str;
use Illuminate\Support\Facades\DB;

class TeacherGradingLogicTest extends TestCase
{
    use DatabaseMigrations;
    protected function setUp(): void
    {
        parent::setUp();
        $this->withoutVite();
    }

    public function test_teacher_cannot_grade_homework_of_another_teacher(): void
    {
        $tenantId = 'school-' . Str::random(5);
        $domain = $tenantId . '.localhost';

        $tenant = Tenant::create(['id' => $tenantId]);
        $tenant->domains()->create(['domain' => $domain]);

        $tenant->run(function () use ($domain) {

            $student = User::factory()->student()->create();
            $mathTeacher = User::factory()->teacher()->create();
            $literatureTeacher = User::factory()->teacher()->create();

            DB::table('students')->insert(['student_id' => $student->id]);
            DB::table('teachers')->insert(['teacher_id' => $mathTeacher->id]);
            DB::table('teachers')->insert(['teacher_id' => $literatureTeacher-
>id]);

            $classId = DB::table('classes')->insertGetId([
                'name' => '10-A',
                'grade_level' => 10,
                'created_at' => now(),
                'updated_at' => now()
            ]);

            $subjectId = DB::table('subjects')->insertGetId([
                'name' => 'Математика',
                'created_at' => now(),
                'updated_at' => now()
            ]);

            $homework = Homework::factory()->create([
                'created_by' => $mathTeacher->id,
                'subject_id' => $subjectId,
                'class_id' => $classId,
            ]);

           $responseAttacker = $this->actingAs($literatureTeacher)
```

```
                ->postJson("http://{$domain}/portal/teacher/homeworks/{$homework-
>homework_id}/grade/{$student->id}", [
                    'grade_value' => 12,
                    'comment' => 'Written off!'
                ]);

            if ($responseAttacker->status() !== 403) {
                dd($responseAttacker->getContent());
            }
            $responseAttacker->assertStatus(403);

            $responseOwner = $this->actingAs($mathTeacher)
                ->post("http://{$domain}/portal/teacher/homeworks/{$homework-
>homework_id}/grade/{$student->id}", [
                    'grade_value' => 10,
                    'comment' => 'Good work'
                ]);

            $responseOwner->assertStatus(302);

            $this->assertDatabaseHas('grades', [
                'student_id' => $student->id,
                'homework_id' => $homework->homework_id,
                'subject_id' => $subjectId,
                'grade_value' => 10
            ]);
        });

        $tenant->delete();
    }
}
```

ДОДАТОК Г
Графічні матеріали та інтерфейс користувача

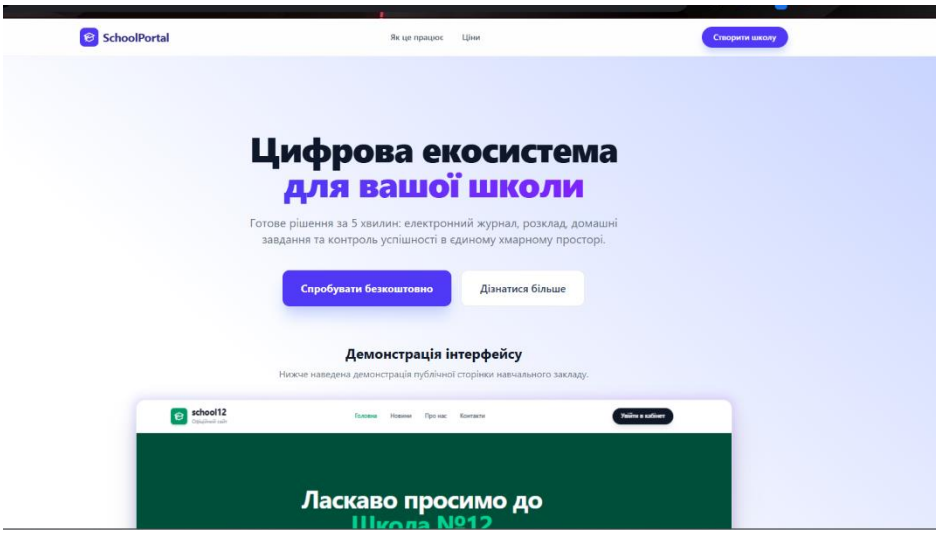

Рисунок Г.1 – Landing page

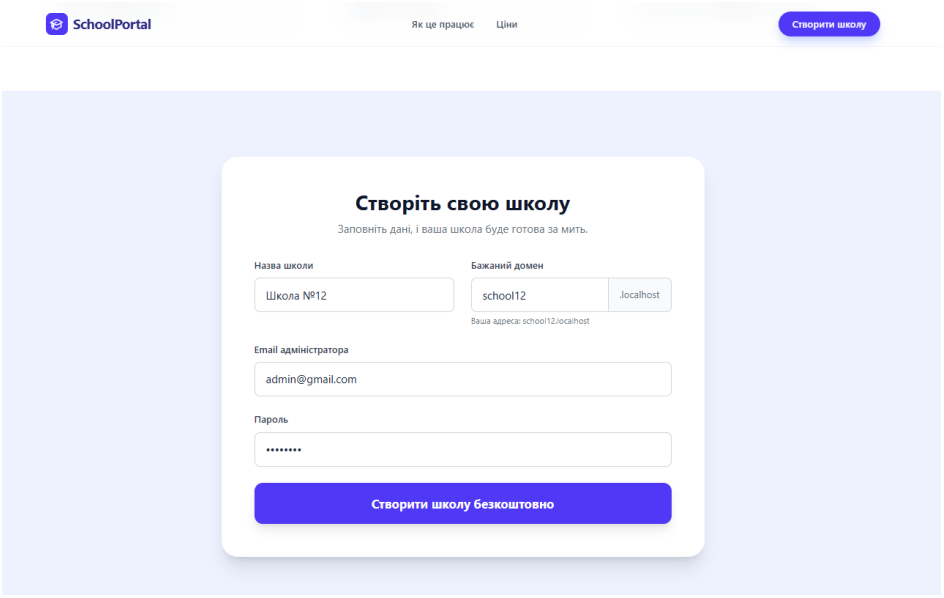

Рисунок Г.2 – Форма реєстрації нового навчального закладу

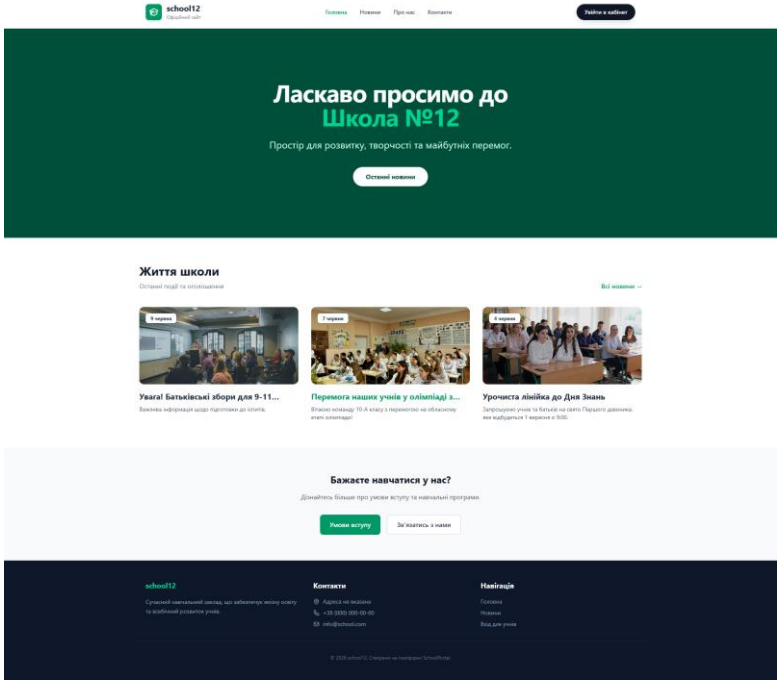

Рисунок Г.3 – Публічна сторінка навчального закладу

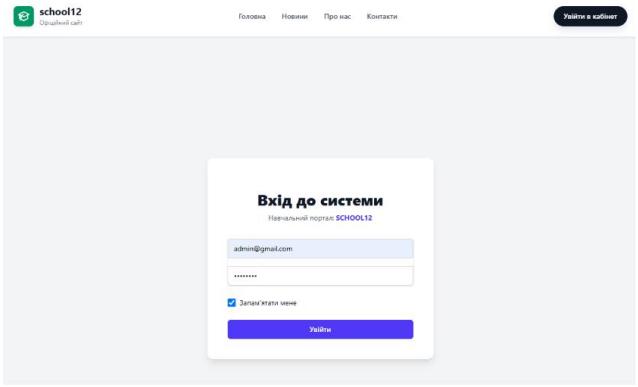

Рисунок Г.4 – Сторінка авторизації користувача

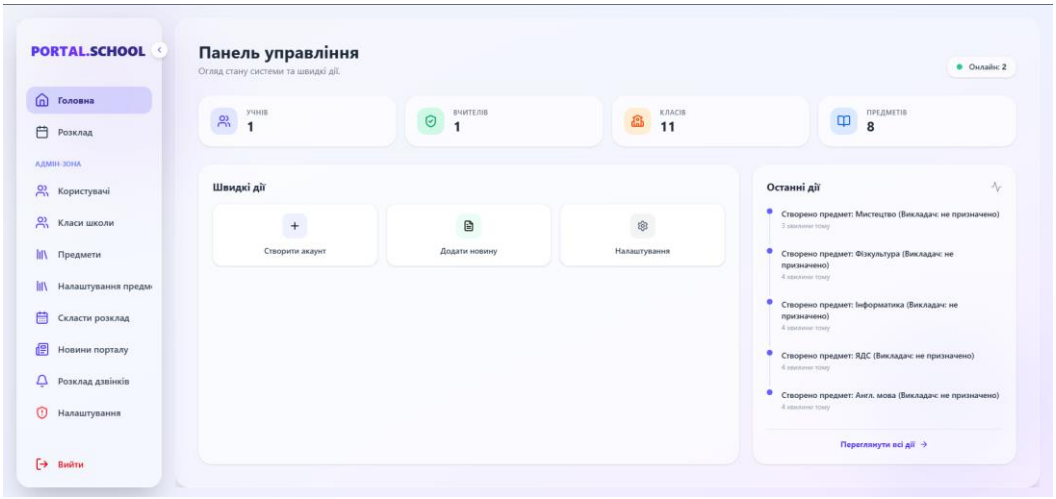

Рисунок Г.5 – Панель управління адміністратора

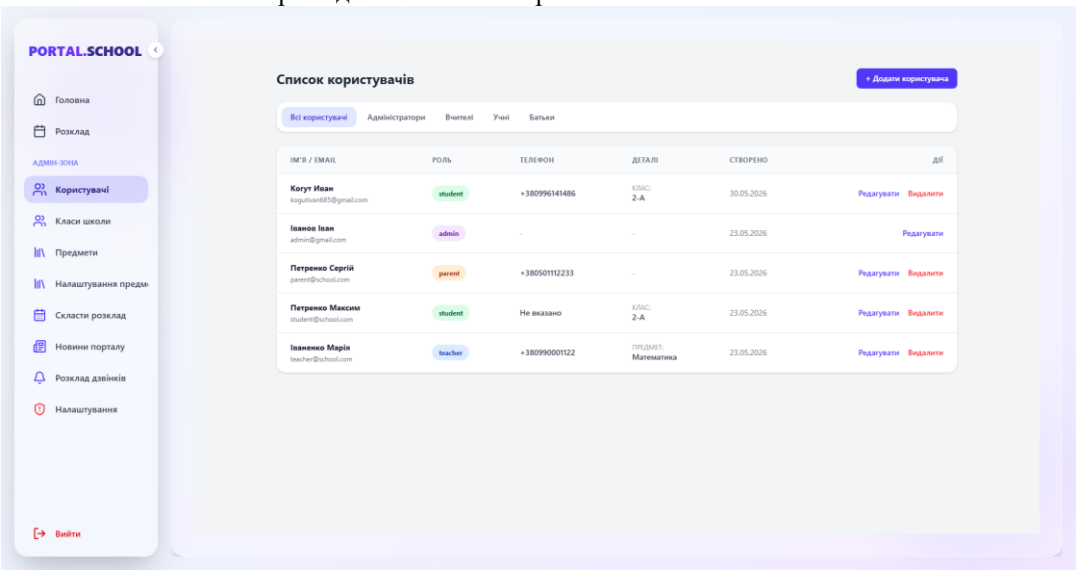

Рисунок Г.6 – Список користувачів

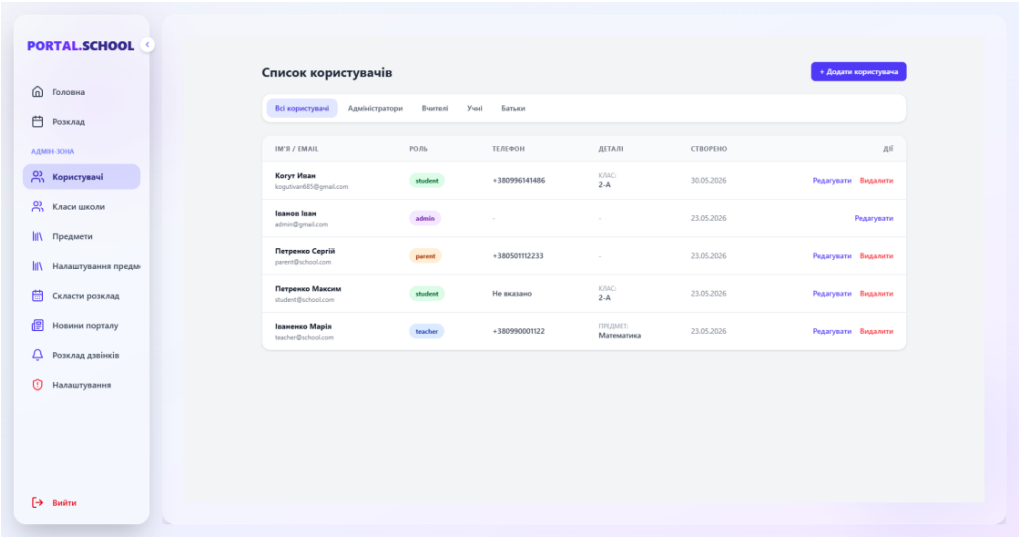

Рисунок Г.7 – Додавання користувачів

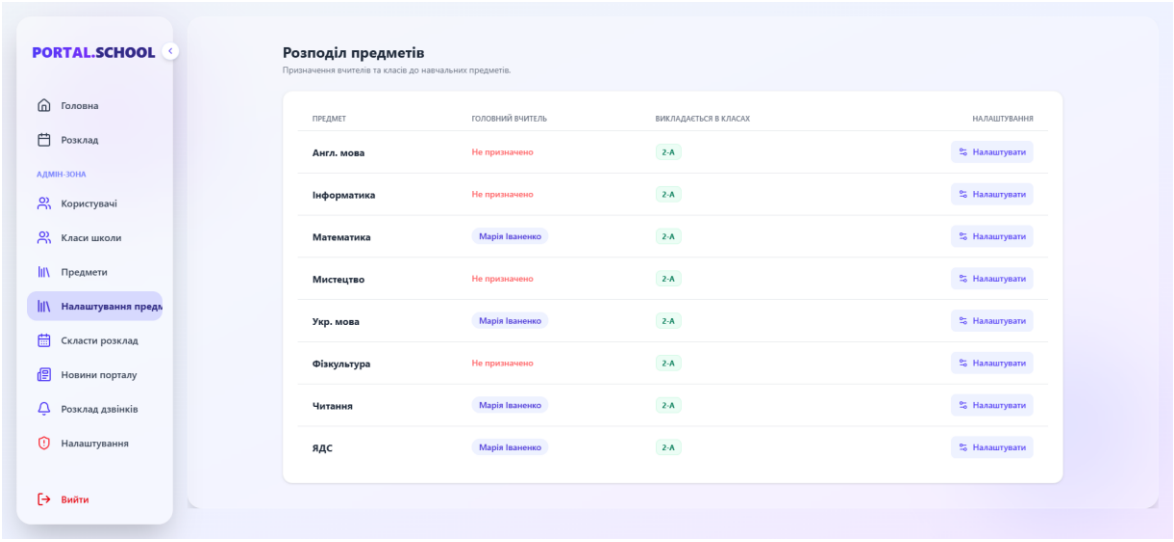

Рисунок Г.8 – Налаштування предмету

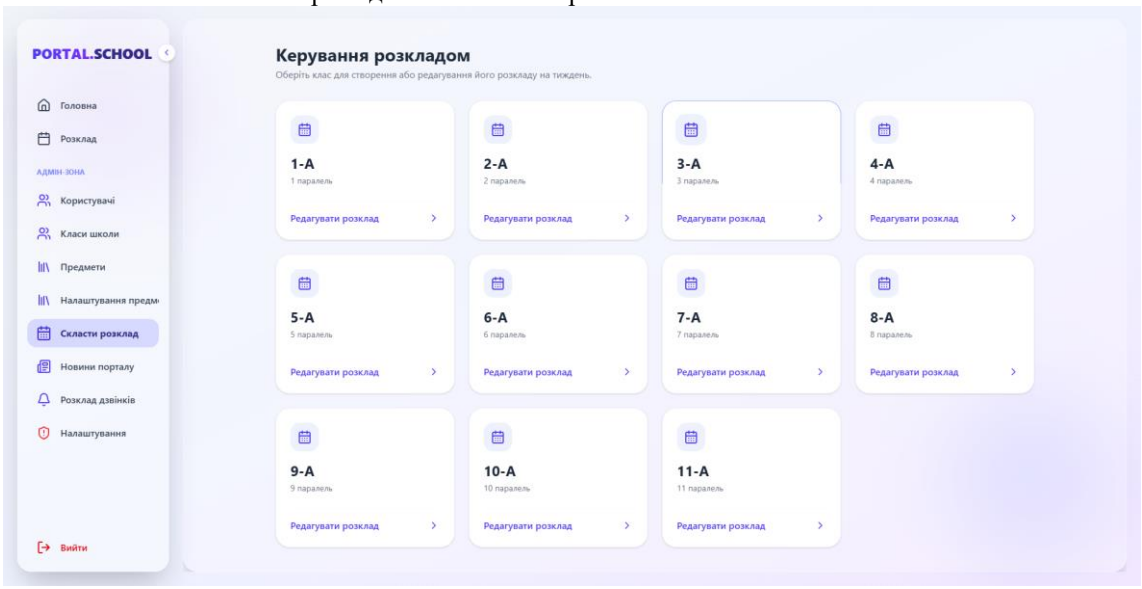

Рисунок Г.9 – Керуванням розкладом

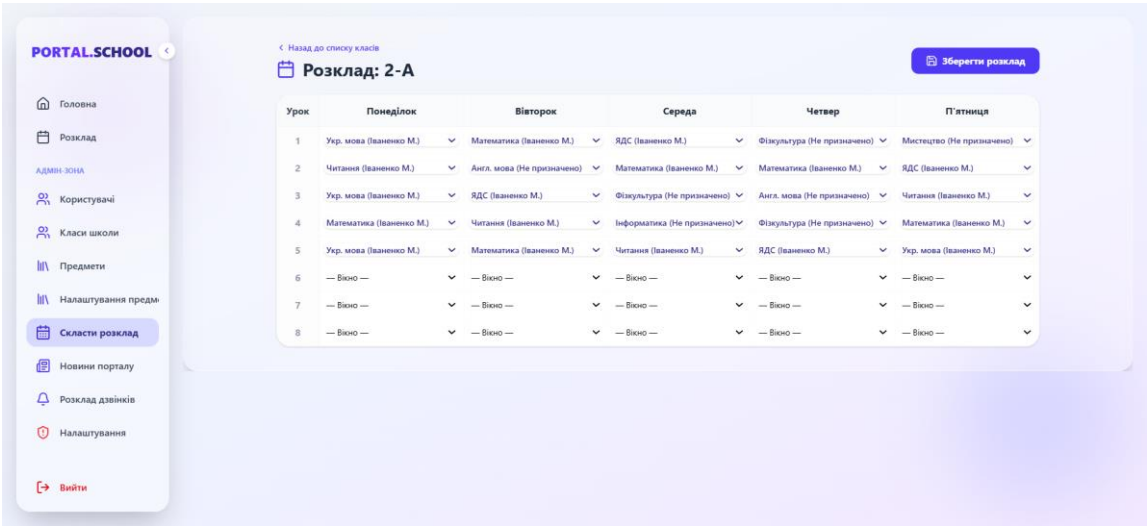

Рисунок Г.10 – Налаштування розкладом

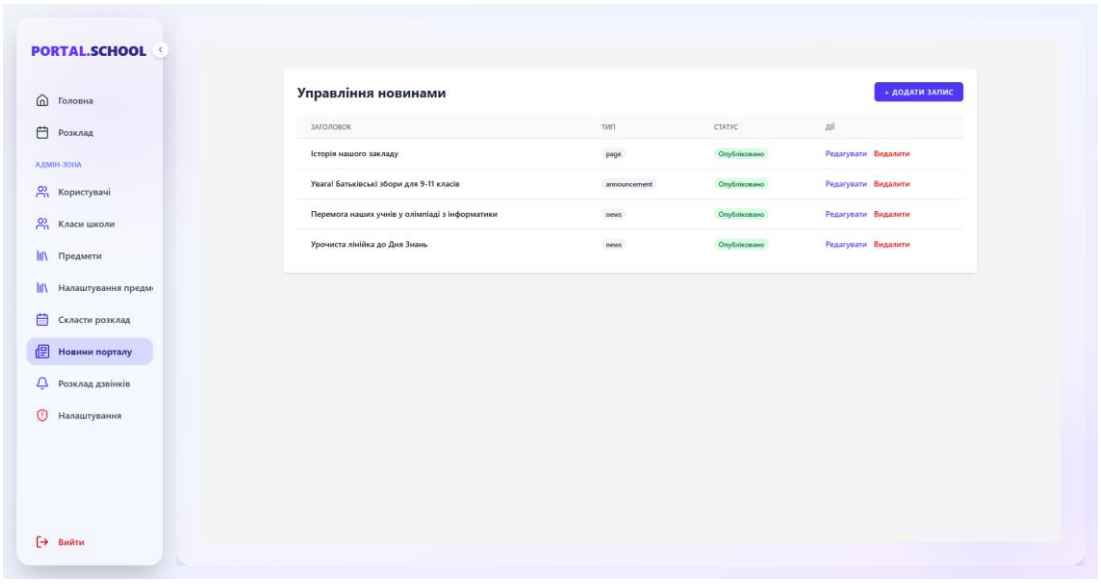

Рисунок Г.11 – Управління новинами

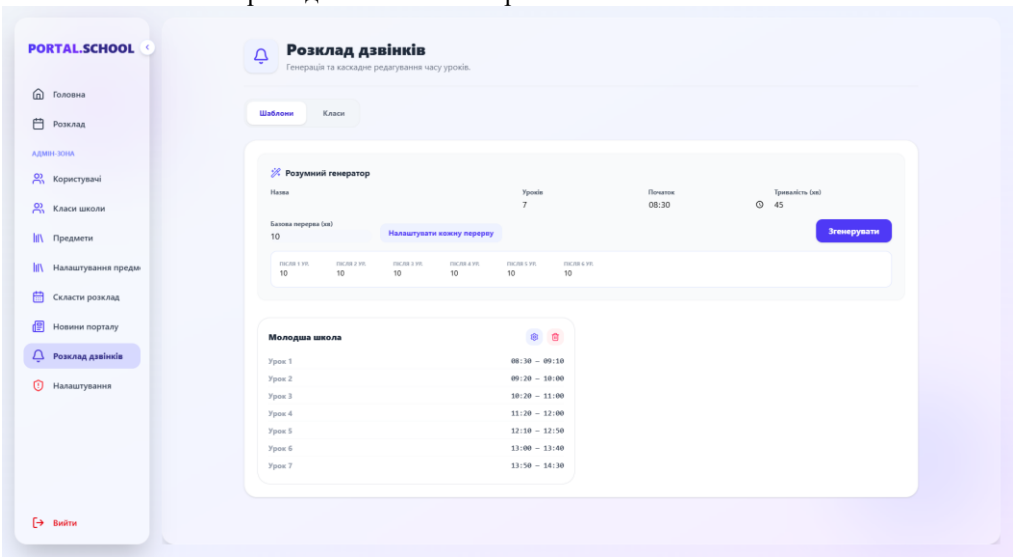

Рисунок Г.12 – Розклад дзвінків

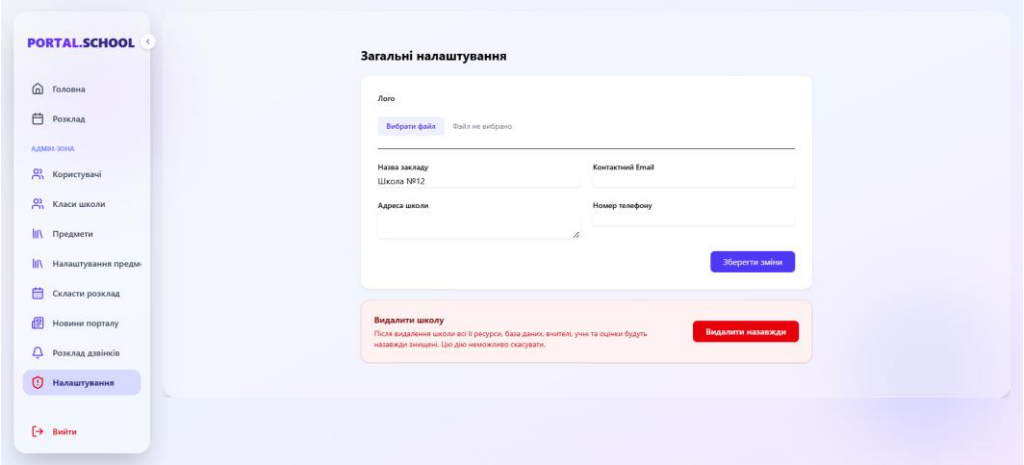

Рисунок Г.13 – Загальні налаштування

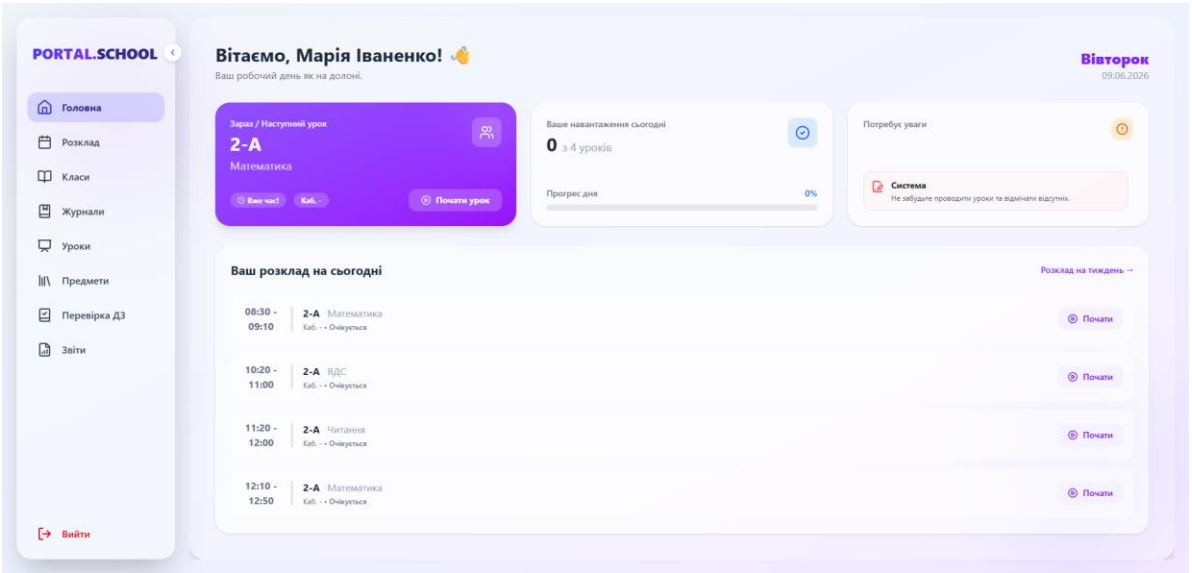

Рисунок Г.14 – Панель управління учителем

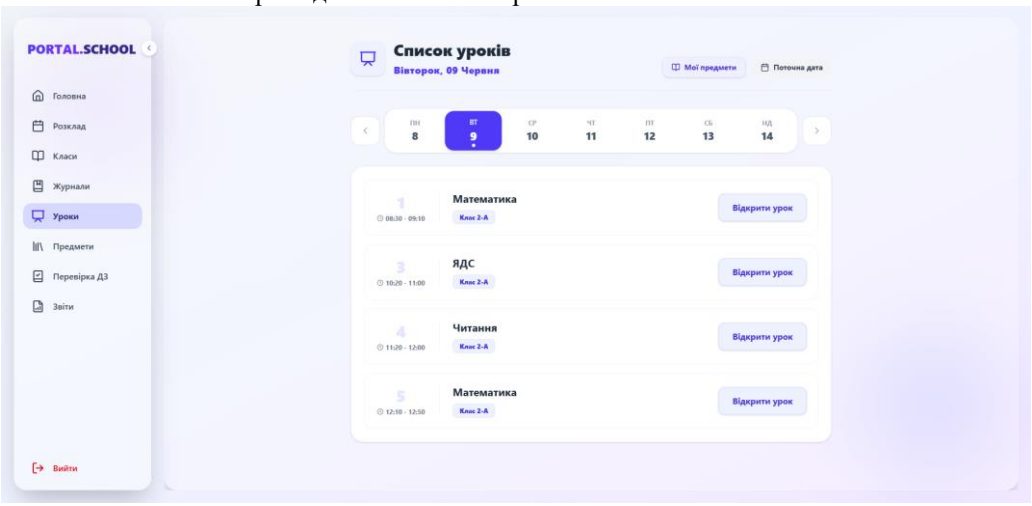

Рисунок Г.15 – Список поточних уроків

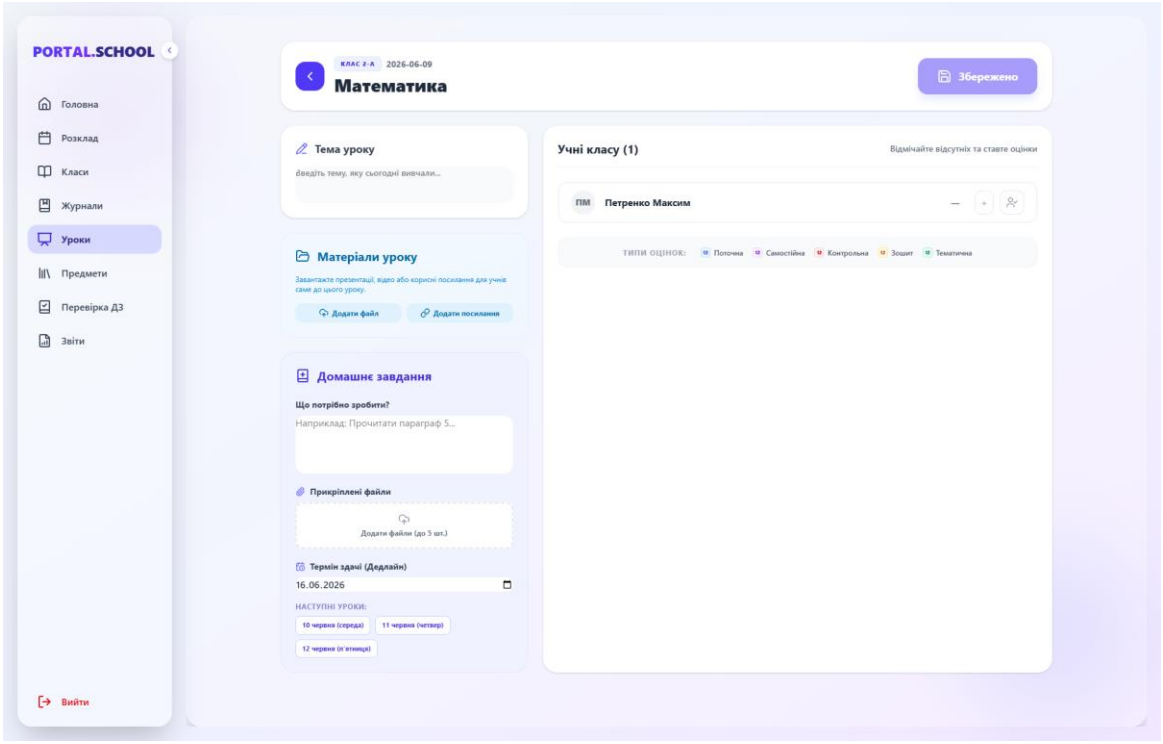

Рисунок Г.16 – Перехід до уроку

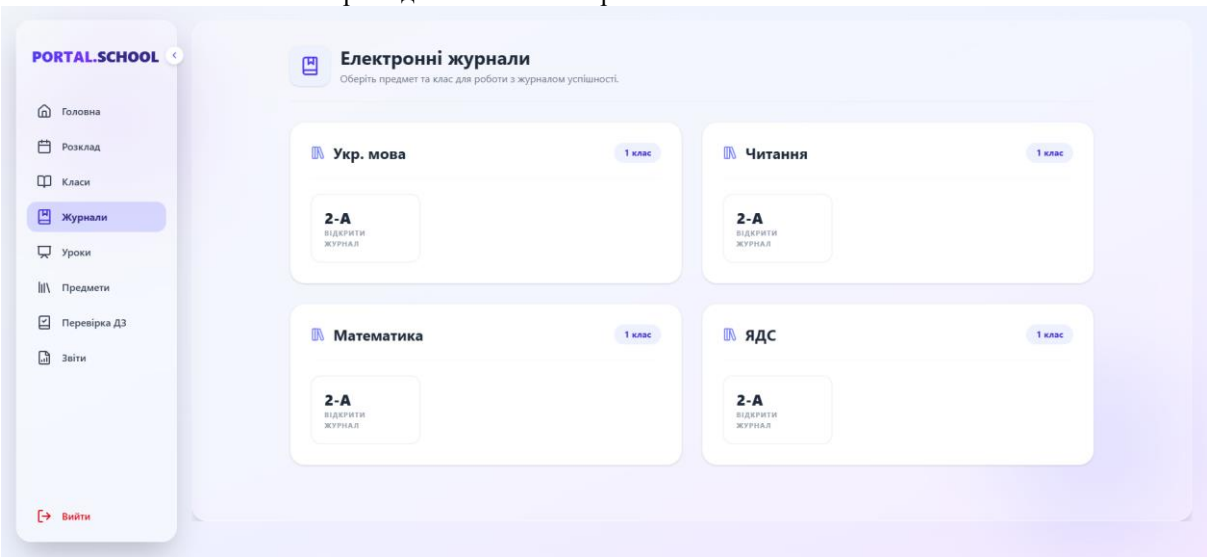

Рисунок Г.17 – Список журналів

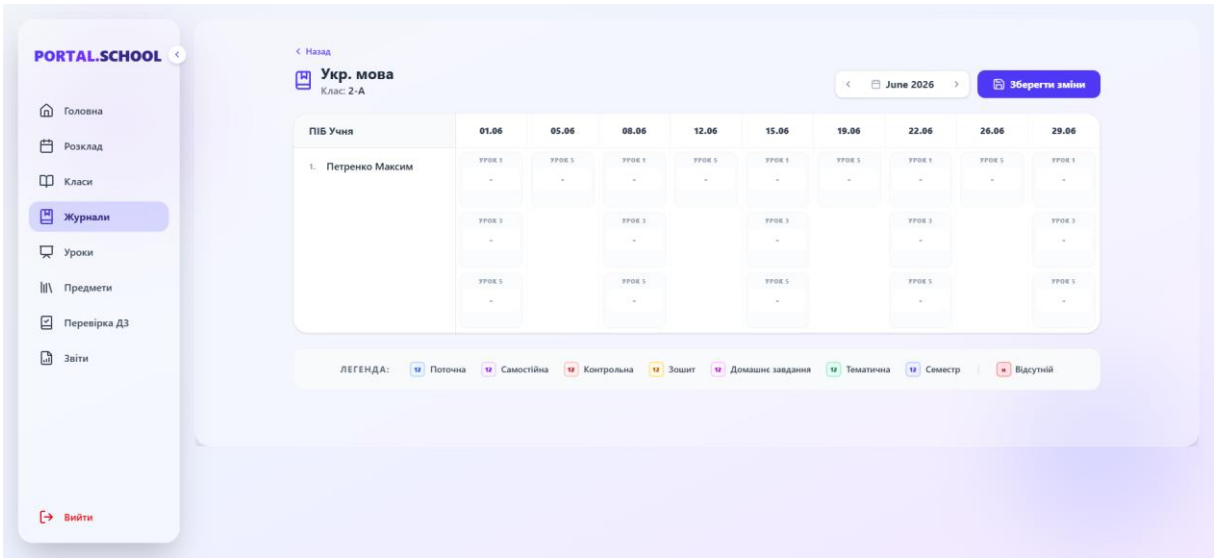

Рисунок Г.18 – Перегляд журналів

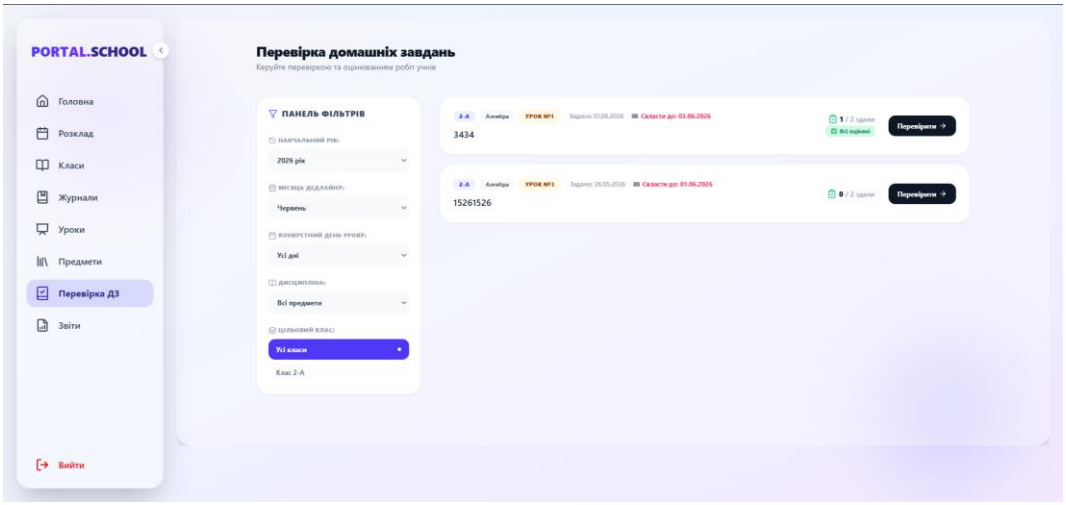

Рисунок Г.19 – Перегляд ДЗ

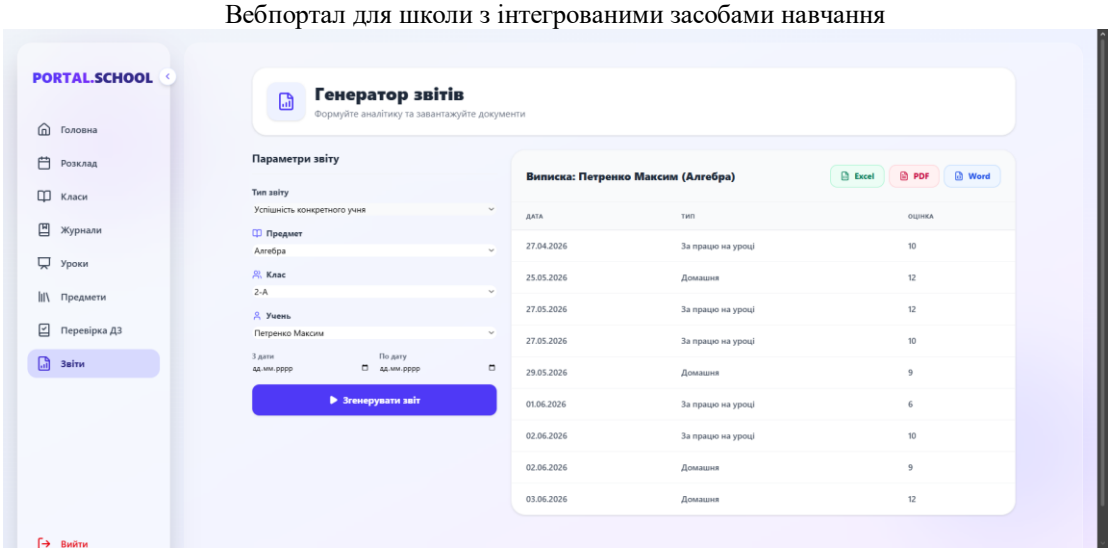

Рисунок Г.20 – Перегляд генератору звіту

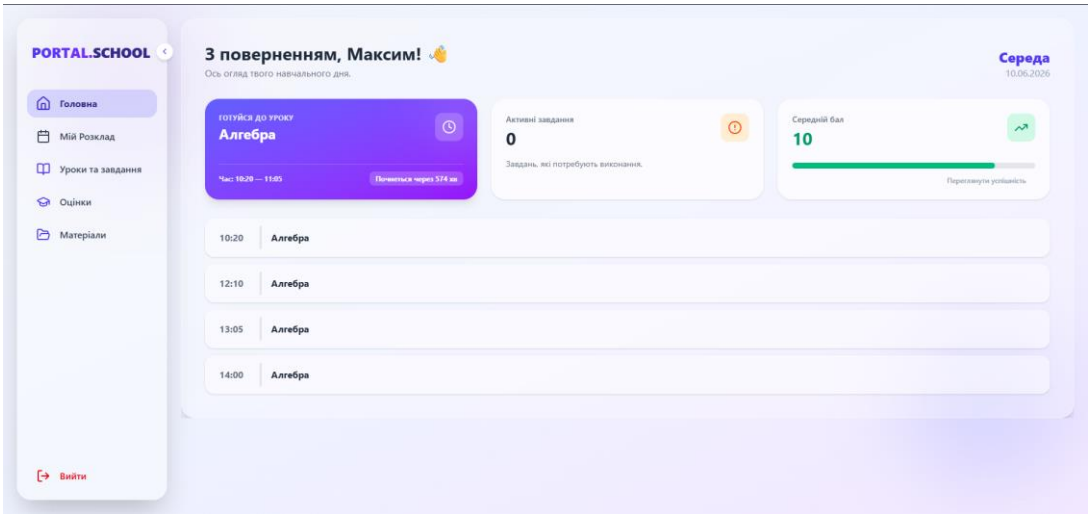

Рисунок Г.21 – Панель управління студента

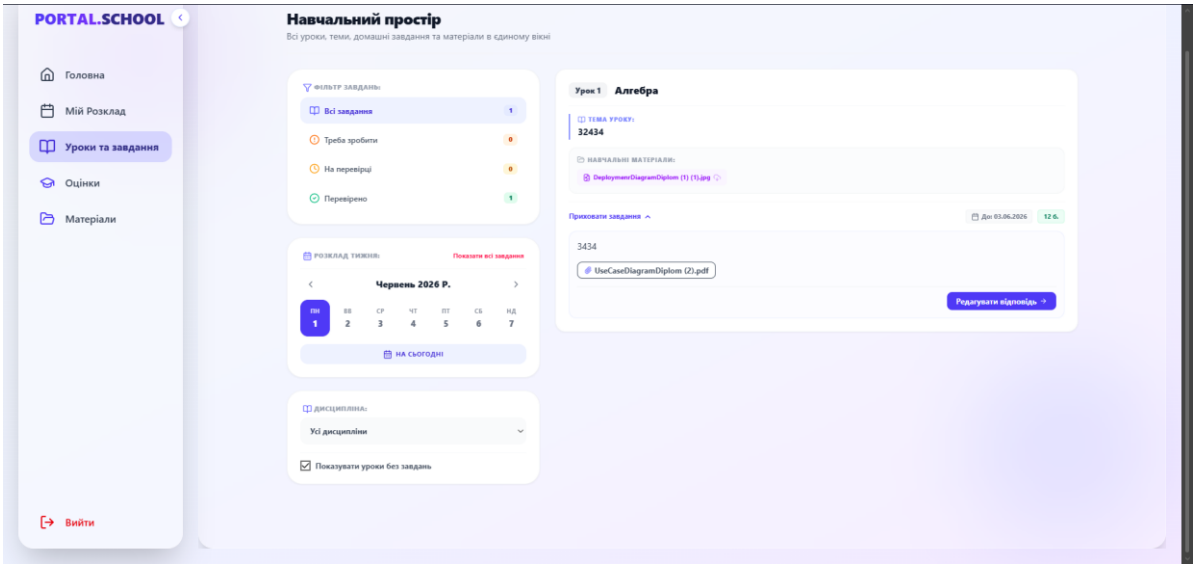

Рисунок Г.22 – Перегляд завдань та матеріалу з уроку

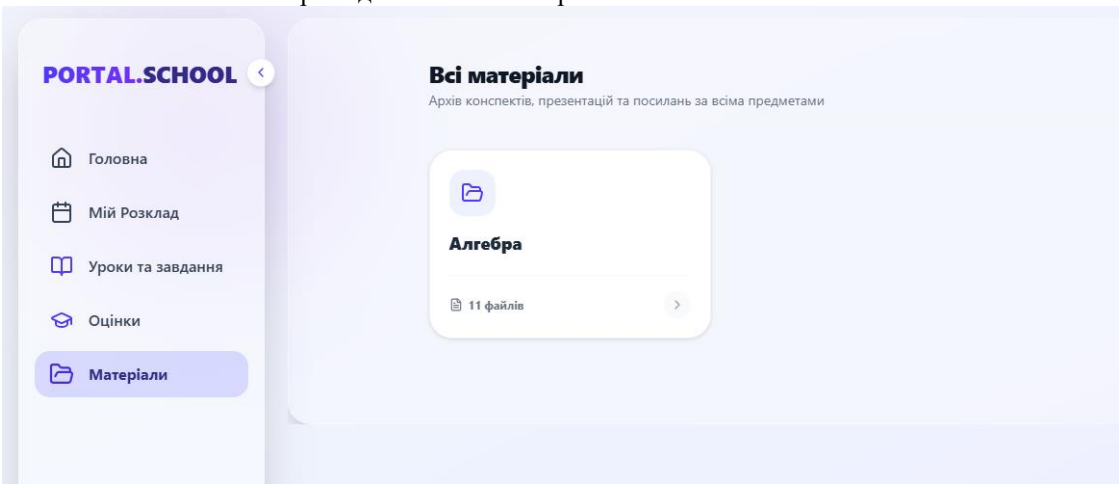

Рисунок Г.23 – Вибір предмету для перегляду всіх матеріалів

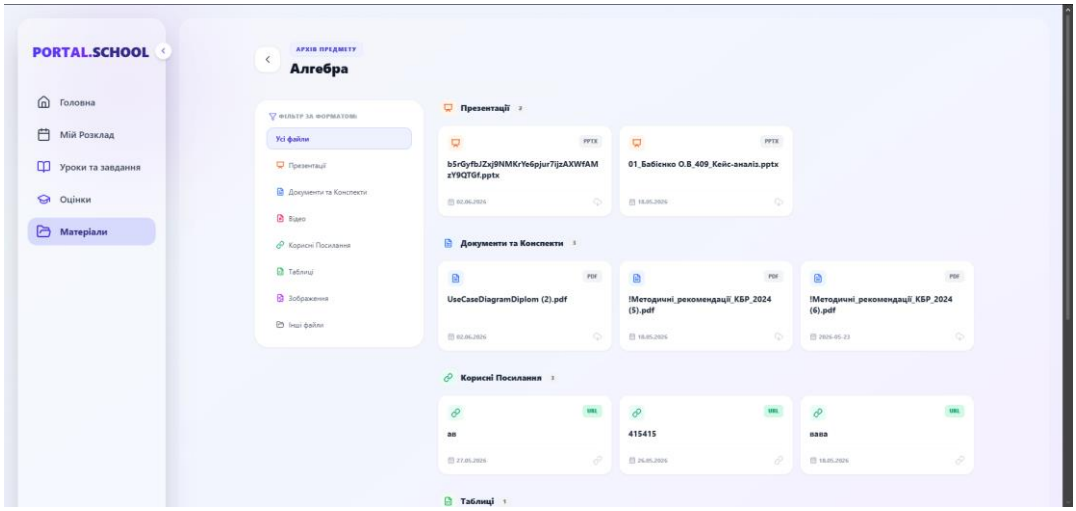

Рисунок Г.24 – Перегляд всіх матеріалів по предмету

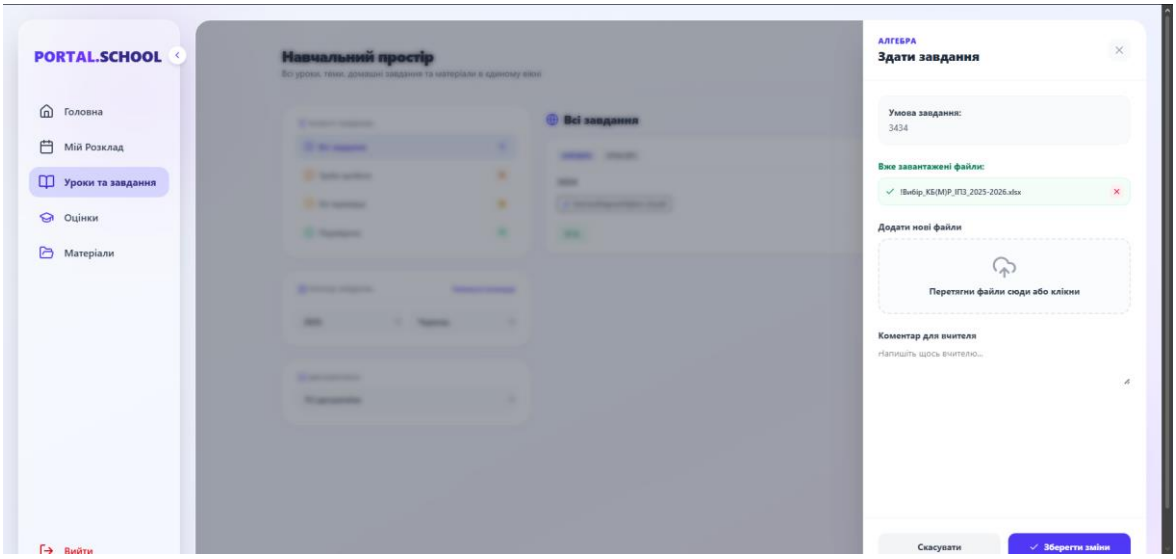

Рисунок Г.25 – Завантаження домашнього завдання по уроку

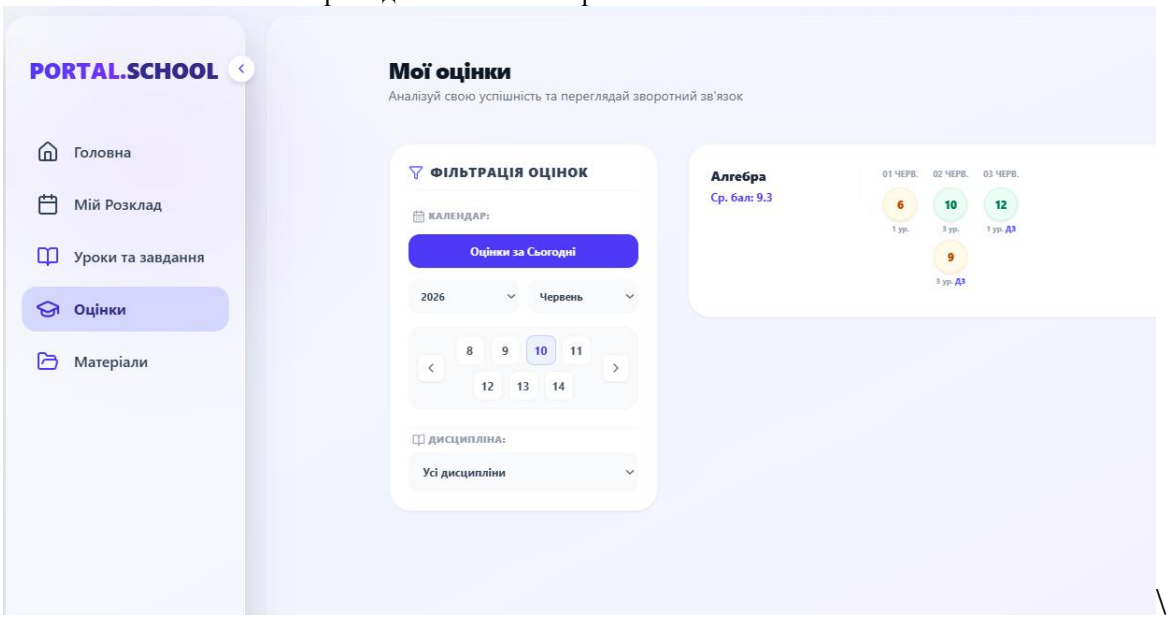

Рисунок Г.26 – Перегляд оцінок студента

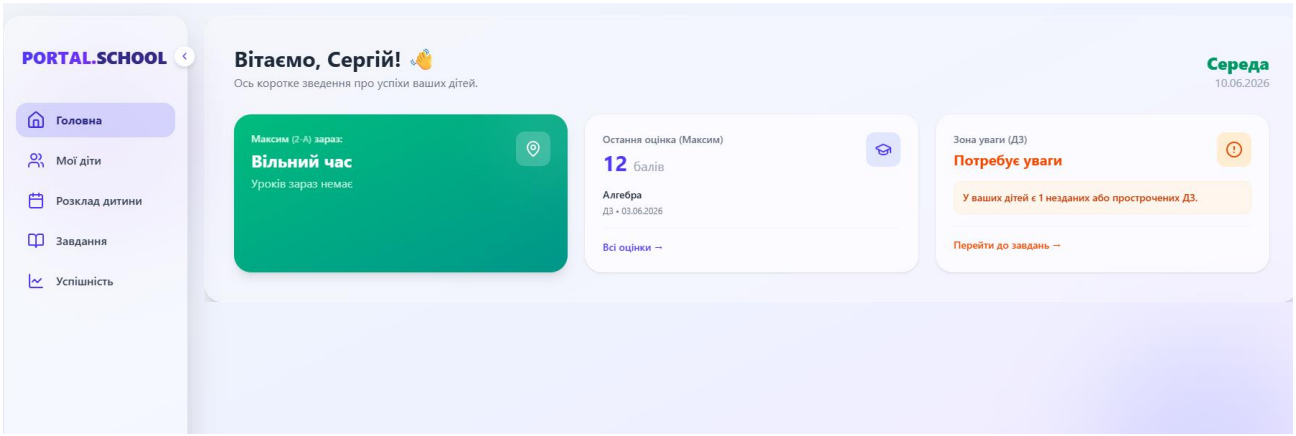

Рисунок Г.27 – Панель управління батьків

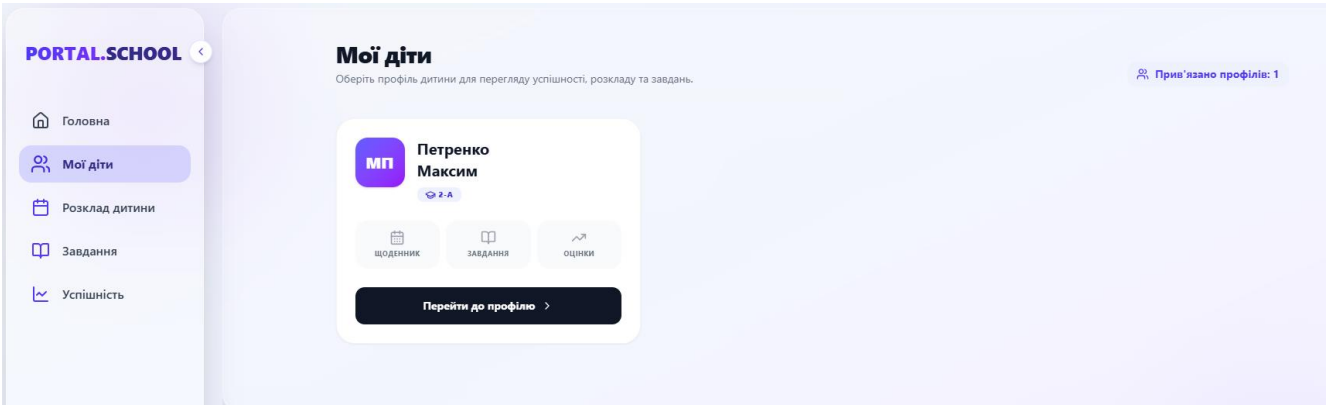

Рисунок Г.28 – Вибір для перегляду профілю учню

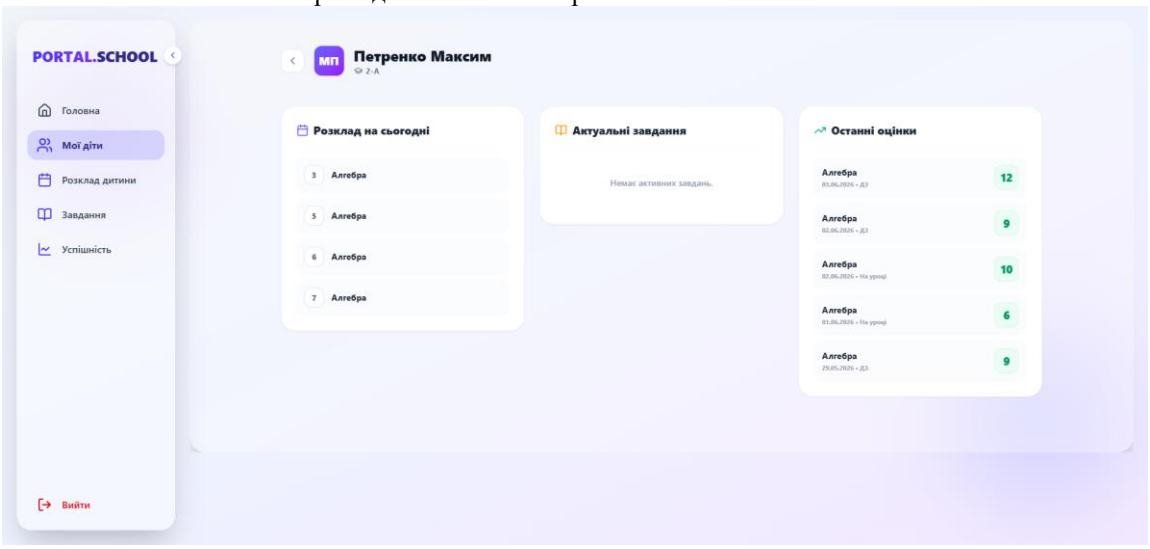

Рисунок Г.29 – Перегляд профілю дитину

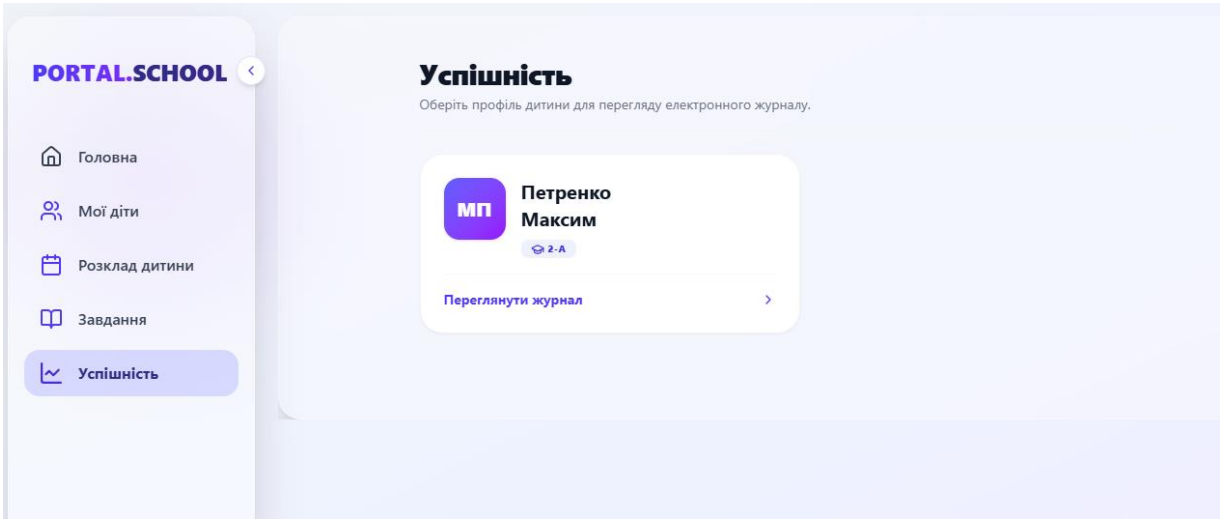

Рисунок Г.30 – Вибір профілю учню для перегляду успішність

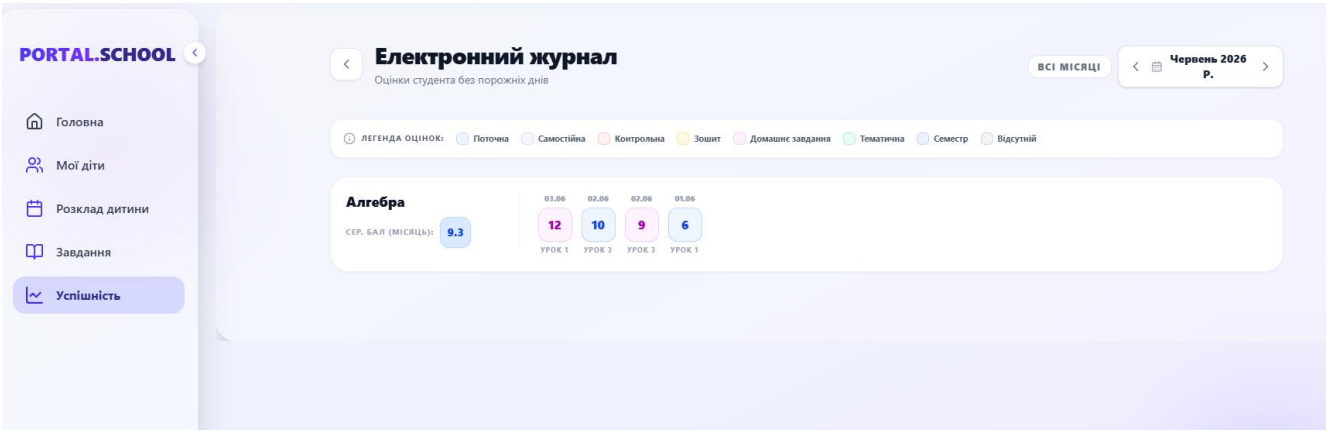

Рисунок Г.31 – Перегляд успішності обраного учня

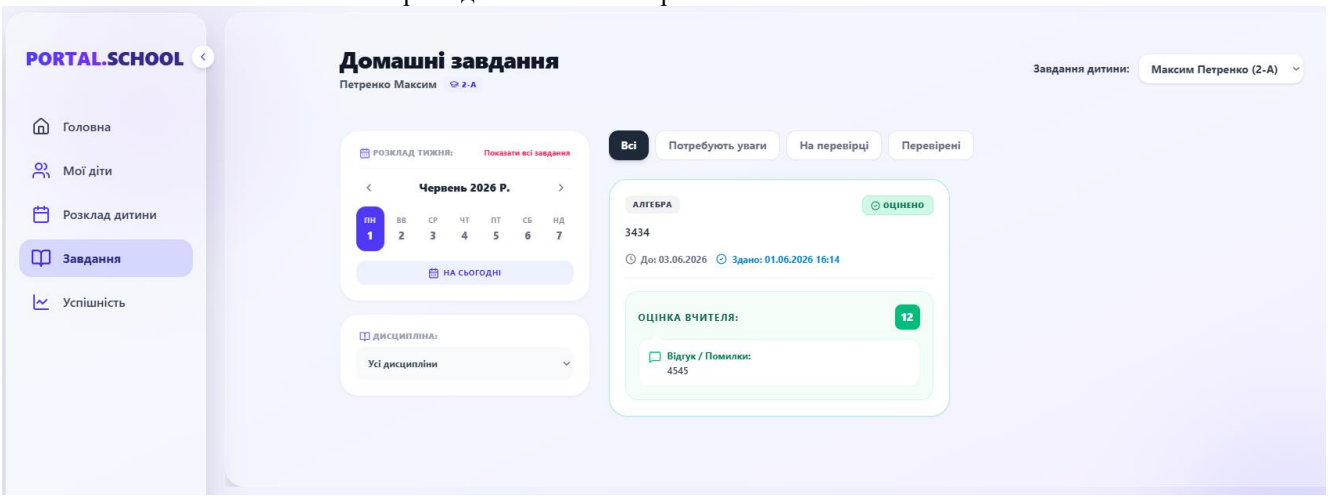

Рисунок Г.322 – Перегляд домашніх завдань учня