МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами

Кафедра програмного забезпечення автоматизованих систем

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

КВАЛІФІКАЦІЙНАРОБОТА

на здобуття ступеня вищої освіти «бакалавр»

на тему: Розробка програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Виконав: студент групи 4151

Заваденко Н. Н.

(підпис, ПІБ)

Керівник роботи:

доцент кафедри ПЗАС, к.т.н., доцент

(посада, науковий ступінь, вчене звання)

Макарова Л. М.

(підпис, ПІБ)

Миколаїв – 2026 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами Кафедра програмного забезпечення автоматизованих систем

Спеціальність 121 «Інженерія програмного забезпечення» Освітня програма «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____доц. Макарова Л.М.
(підпис)
«_07_» __04__ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «бакалавр»

Студенту Заваденко Назарій Назарович_____
(Прізвище, ім'я, по батькові)

1. Тема роботи: Розробка програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії_____
Керівник роботи Макарова Лідія Миколаївна_____
Затверджені наказом ректора №275-уч від 07.04.2026 р._____
2. Термін подання роботи: 01.06.2026 р_____
3. Вихідні дані по роботі_____
4. Перелік питань, що належать до розробки (найменування розділів)

_____

Титульний аркуш; Завдання на кваліфікаційну роботу; Анотація (українською та англійською мовами); Зміст; Перелік умовних позначень, символів, одиниць та термінів (при необхідності); Вступ; Аналіз практичної задачі інженерії програмного забезпечення; Проєкт програмного забезпечення; Результати розробки програмного забезпечення; Розділ з охорони праці; Висновки; Список використаних джерел; Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма та методика випробувань програмного забезпечення).

5. Перелік презентаційних матеріалів: Титульний слайд, Мета та завдання кваліфікаційної роботи, Аналіз вимог до програмного забезпечення, Архітектура програмного забезпечення, Діаграма класів програмного забезпечення, Діаграми послідовності програмного забезпечення, Логічна модель бази даних програмного забезпечення, Робочий проект програмного забезпечення, Результати розробки програмного забезпечення, Висновки

_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	Гурець Н.В., ст. викладач		

7. Дата видачі завдання 07.04.2026 р.

К А Л Е Н Д А Р Н И Й П Л А Н

Номер	Назва етапів роботи	Терміни виконання	Примітка
1.	Підготовка розділу ВСТУП	30.03.2026	ПП*
2.	Аналіз практичної задачі інженерії ПЗ	06.04.2026	ПП*
3.	Аналіз вимог до ПЗ	13.04.2026	ПП*
4.	Розробка архітектури ПЗ	20.04.2026	
5.	Розробка проекту ПЗ	04.05.2026	
6.	Реалізація та тестування ПЗ	11.05.2026	
7.	Підготовка розділу Результати розробки ПЗ	15.05.2026	
8.	Підготовка розділу з охорони праці	18.05.2026	ПП*
9.	Підготовка розділу ВИСНОВКИ	22.05.2026	
10.	Оформлення списку використаних джерел та додатків	25.05.2026	
11.	Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, у роздрукованому та електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версій роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020 р.)	01.06.2026	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку
12.	Підготовка презентації та доповіді	05.06.2026	
13.	Попередній захист роботи на засіданні кафедри ПЗАС	08.06.2026	
14.	Подання на кафедру ПЗАС електронних версій наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.); презентації доповіді, а також Авторського договору з додатком	15.06.2026	

* - за результатами переддипломної практики (ПП), яка була з 02.02.2026 до 22.03.2026 р.

Студент _____ Заваденко Н. Н.
(підпис) (ПІБ)

Керівник роботи _____ Макарова Л.М.
(підпис) (ПІБ)

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці вебзастосунку для автоматизації процесу вступу учнів молодших класів до гімназії, що відноситься до практичної задачі інженерії програмного забезпечення з комплексністю та невизначеністю умов.

Було проведено аналіз практичної задачі інженерії програмного забезпечення та розглянуто існуючі програмні продукти та методи розробки, що стосуються відповідної проблематики. Було проведено аналіз практичної задачі інженерії програмного забезпечення та розглянуто існуючі програмні продукти та методи розробки, що стосуються відповідної проблематики. Проведено постановку задачі на розробку вебзастосунку для автоматизації процесу вступу учнів молодших класів до гімназії. Виконано аналіз вимог до системи, спроєктовано архітектуру, розроблено ескізний, технічний та робочий проєкти програмного забезпечення, а також представлено результати розробки та розділ з охорони праці.

Робота викладена на 88 сторінках, містить 21 рисунок, 11 таблиць, список використаних джерел із 17 найменувань та 5 додатків.

ABSTRACT

The qualification work is devoted to the development of a web application to automate the process of admission of junior high school students to a gymnasium, which refers to a practical problem of software engineering with complexity and uncertainty of conditions.

An analysis of a practical problem of software engineering was conducted and existing software products and development methods related to the relevant issues were considered. An analysis of a practical problem of software engineering was conducted and existing software products and development methods related to the relevant issues were considered. A task was set for the development of a web application to automate the process of admission of junior high school students to a gymnasium. An analysis of system requirements was performed, the architecture was designed, draft, technical and working software projects were developed, and the development results and a section on labor protection were presented.

The work is presented on 88 pages, contains 21 figures, 11 tables, a list of sources used with 17 names, and 5 appendices.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій цифровізація освітніх процесів стала не просто перевагою, а необхідністю для ефективного функціонування навчальних закладів. Автоматизація вступу до гімназій дозволяє значно знизити ймовірність помилок, підвищити продуктивність роботи адміністрації та забезпечити прозорість взаємодії між навчальним закладом і батьками.

Освітня галузь України нині стикається зі значними викликами, зумовленими воєнним станом, що вимагає переходу багатьох процесів у дистанційний або гібридний формат. У таких умовах ручна обробка паперових заявок стає неефективною. Вебзастосунок для автоматизації процесу вступу учнів молодших класів дозволяє централізовано приймати заявки, відстежувати їхній статус, фіксувати результати випробувань та оперативно інформувати батьків, що є критично важливим для підтримки довіри до освітньої системи.

Мета роботи - розробка вебзастосунку для автоматизації процесу вступу учнів молодших класів до гімназії, що дозволить підвищити зручність і ефективність роботи адміністрації, забезпечити прозорість відбору та мінімізувати кількість помилок у процесі обліку даних.

Завдання роботи:

- Проаналізувати предметну область та сучасні тенденції автоматизації освітніх процесів.
- Провести порівняльний аналіз існуючих програмних рішень-аналогів та обґрунтувати доцільність розробки власного вебзастосунку.
- Сформулювати постановку задачі та розробити технічне завдання на створення програмного продукту.
- Провести аналіз функціональних та нефункціональних вимог до системи.
- Розробити архітектуру вебзастосунку.

- Розробити проєкт програмного забезпечення, що включає проєктування бази даних, архітектури та інтерфейсу користувача.
- Здійснити програмну реалізацію вебзастосунку з використанням сучасного стеку технологій.
- Провести тестування розробленого рішення та розробити супровідну програмну документацію.

Об'єктом кваліфікаційної роботи є процес розробки програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії.

1 АНАЛІЗ ПРАКТИЧНОЇ ЗАДАЧІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІНФОРМАЦІЙНОГО СУПРОВОДУ ВСТУПНОЇ КАМПАНІЇ ДО ГІМНАЗІЇ

1.1 Аналіз практичної задачі інженерії програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

В даній кваліфікаційній роботі в якості практичної задачі інженерії програмного забезпечення розглядається розробка вебзастосунку для автоматизації процесу вступу учнів молодших класів до гімназії.

Процес вступу охоплює широкий спектр організаційних та управлінських завдань, починаючи від прийому заяв від батьків і закінчуючи зарахуванням учнів за результатами вступних випробувань чи співбесід. Основна діяльність під час вступної кампанії гімназії включає забезпечення оптимального використання часу адміністрації та зручності для батьків майбутніх учнів. Крім того, правильна організація прийому є важливим елементом управління освітнім процесом.

Наразі, під час вступної кампанії, багато гімназій змушені виконувати значний обсяг ручної та паперової роботи. Адміністратори приймальної комісії витрачають багато часу на тривалі процедури обробки заявок, а вчителі виконують значний обсяг роботи під час перевірки іспитів та формування паперових висновків. Для батьків ручний метод створює суттєві незручності: відстежувати статус заявки доводиться вручну, через телефонні дзвінки або особисті візити до закладу.

Тому автоматизація цих дій дозволить значно зменшити всі перелічені недоліки. Запропоноване рішення дозволить скоротити час на обробку заявок, надати батькам зручний інструмент для відстеження статусу онлайн, а також спростити роботу вчителів завдяки можливості електронного внесення

результатів. Директор отримає зручний інструментарій для наочного контролю за вступною кампанією та аналізу статистики.

Для вирішення цієї задачі розробляється спеціалізоване програмне забезпечення. Для цього можуть використовуватись такі технології як React, TypeScript, Node.js та HTML. Також використовується легка реляційна база даних SQLite.

Для організації процесу вступу необхідно мати дані про користувачів (батьків, вчителів, адміністраторів, директора):

- прізвище, ім'я, по батькові;
- електронна пошта (щоб надсилати сповіщення та входити в систему);
- контактний номер телефону;
- пароль та визначена роль у системі.

Дані про вступну заявку містять наступні відомості:

- дані про дитину (кандидата);
- статус заявки;
- результати іспитів та співбесід;
- коментарі екзаменаторів.

1.2 Аналіз існуючих програмних рішень

При виконанні цієї роботи важливо розглянути вже існуючі програмні рішення для цифровізації процесу прийому учнів. Нижче наведено аналіз декількох таких програм.

Одним із найпоширеніших рішень в Україні є державна інформаційна система «Електронна реєстрація в заклади загальної середньої освіти» (ІСУО). Це загальнонаціональний портал, який дозволяє батькам створювати електронні кабінети, завантажувати сканкопії документів та подавати заяви до обраних навчальних закладів за територіальним принципом.

Перевагами системи ІСУО є наявність єдиної централізованої бази даних, стандартизований процес реєстрації та високий рівень безпеки персональних даних. Проте система має суттєві недоліки для гімназій: вона жорстко орієнтована на загальноосвітні школи та територіальні черги. Система абсолютно не адаптована для гімназій, де зарахування відбувається виключно на основі конкурсного відбору, оскільки у ній відсутній модуль для організації іспитів та немає спеціалізованої ролі «Вчитель/Екзаменатор» для виставлення оцінок [1].

Головна сторінка порталу «Електронна реєстрація в ЗЗСО» зображено на рисунку 1.1.

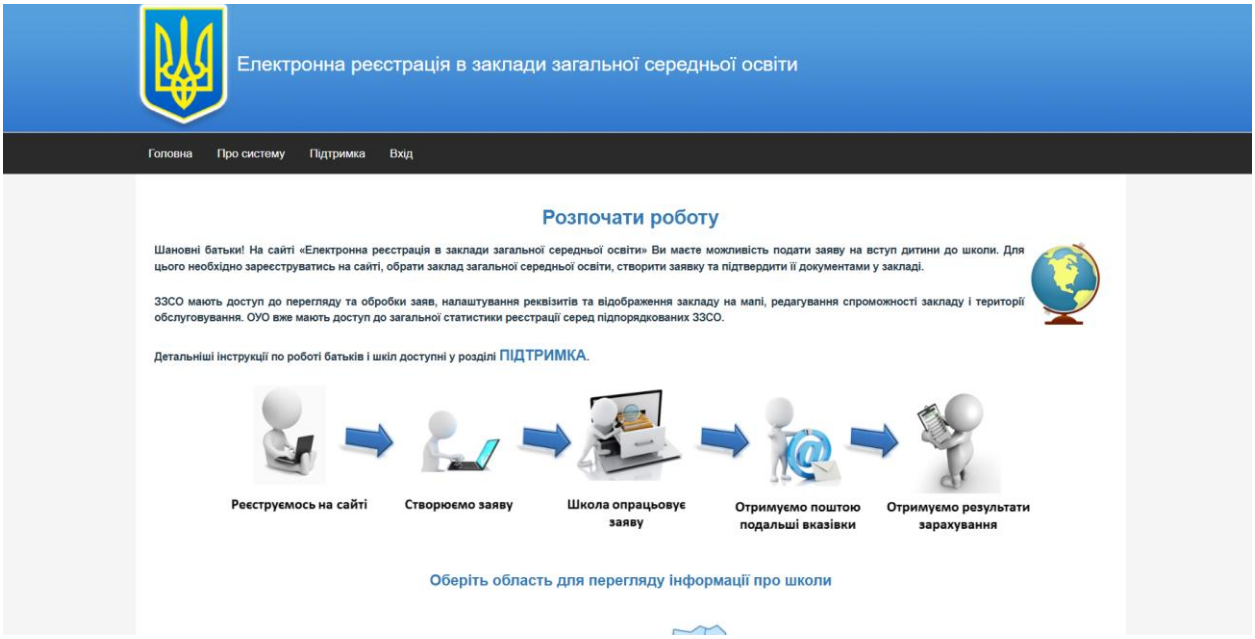

Рисунок 1.1 Головна сторінка порталу «Електронна реєстрація в ЗЗСО»

Ще одним поширеним підходом є використання адаптованих універсальних CRM-систем для навчальних закладів, таких як KeyCRM або KeepinCRM. Ці платформи є комерційними програмними продуктами, які орієнтовані на збір лідів та візуалізацію процесу за допомогою Kanban-дошок.

Перевагами CRM-систем є потужні інструменти для комунікації з батьками (SMS, месенджери) та гнучка архітектура бази даних, що дозволяє створювати додаткові поля в картці учня. Однак їх недоліком є надмірна

комерційна спрямованість: вони перевантажені модулями фінансового обліку, які не потрібні державній гімназії. Крім того, у CRM-системах відсутній спеціалізований особистий кабінет для батьків для перегляду оцінок, а також незручно реалізований процес виставлення оцінок для вчителів [2].

Інтерфейс Kanban-дошки управління заявками в CRM-системі зображено на рисунку 1.2

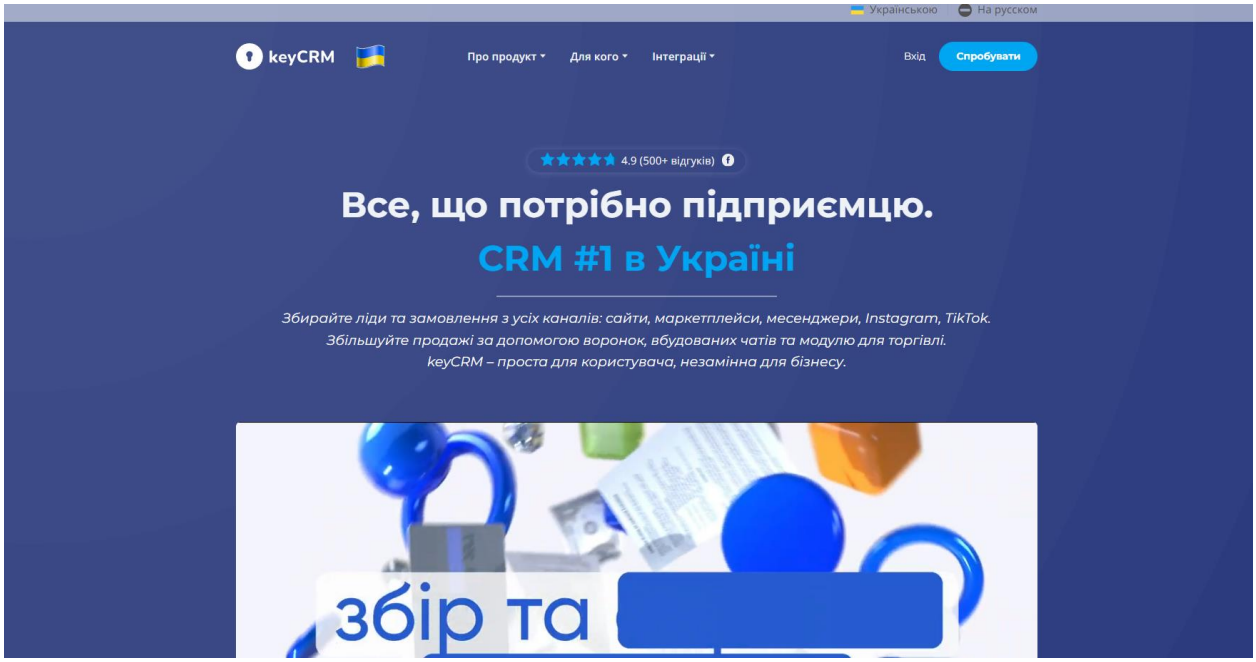

Рисунок 1.2 Інтерфейс Kanban-дошки управління заявками в CRM-системі

Незважаючи на наявні аналоги, все ж потрібно розробити власний вебзастосунок для автоматизації вступу до гімназії і на це є ряд причин:

– Адаптація: державні та комерційні програми не враховують індивідуальні вимоги конкурсного відбору та оцінювання. Власний вебзастосунок дозволить точно налаштувати функціональність під потреби закладу.

– Простіший інтерфейс: комерційні системи мають забагато бізнес-функцій. Власний вебзастосунок дозволить зробити інтерфейс простішим для вчителів та батьків, залишивши необхідний мінімум функцій.

– Економічність: відсутність необхідності сплачувати щомісячну абонентську плату за кожного вчителя або адміністратора.

1.3 Постановка задачі на розробку програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Виходячи з проведеного аналізу, сформулюємо постановку задачі на розробку програмного забезпечення для автоматизації процесу вступу до гімназії.

Потрібно створити вебзастосунок для реєстрації нових заявок, обліку кандидатів, внесення результатів випробувань та відстеження статусу заявки. Вебзастосунок повинен мати зрозумілий та гарний інтерфейс. У базі даних вебзастосунку повинна зберігатися наступна інформація:

Про заявку (кандидата):

- номер заявки;
- прізвище, ім'я, по батькові дитини;
- дата народження;
- адреса проживання;
- статус заявки.

Про користувачів (батьки, вчителі, адміністратор, директор):

- унікальний ідентифікатор;
- прізвище, ім'я, по батькові;
- електронна пошта;
- номер телефону;
- роль у системі.

Про результати іспитів:

- унікальний ідентифікатор запису;
- кандидат;
- назва іспиту/предмету;
- кількість отриманих балів;
- висновок/коментар екзаменатора.

З даним застосунком повинні мати можливість працювати такі групи користувачів:

- адміністратор (завуч);
- директор;
- вчителі (екзаменатори);
- батьки (заявники).

Вебзастосунок повинен забезпечити можливість виконання наступних функцій:

- Створення електронної заяви на вступ Вхідні дані: ПІБ дитини, дата народження, адреса, контакти батьків. Вихідні дані: Створення нового запису кандидата, присвоєння статусу «Нова».
- Редагування статусу заявки Вхідні дані: ID заявки, новий статус (наприклад: "Документи прийнято", "Відхилено", "Зараховано"). Вихідні дані: Підтвердження зміни статусу та збереження в базі.
- Внесення результатів іспитів та співбесід Вхідні дані: ID іспиту, ID кандидата, кількість отриманих балів, коментар екзаменатора. Вихідні дані: Підтвердження додавання оцінки та збереження результатів у базі даних.
- Затвердження списку зарахованих Вхідні дані: Натиснути на кнопку затвердження для вибраних кандидатів (накладання електронної резолюції). Вихідні дані: Масова зміна статусів обраних кандидатів на «Рекомендовано до зарахування».

При роботі з вебзастосунком адміністратор повинен мати можливість вирішувати такі завдання:

- Переглянути інформацію про всі подані заявки;
- Редагувати статуси заяв та інформацію про кандидатів;
- Формувати екзаменаційні групи та розклад випробувань;
- Управляти профілями користувачів у системі.

При роботі з вебзастосунком директор повинен мати можливість вирішувати такі завдання:

– Переглядати загальну статистику та відфільтровані рейтингові списки;

– Приймати остаточне рішення щодо зарахування (змінювати статуси масово).

При роботі з вебзастосунком вчитель повинен мати можливість вирішувати такі завдання:

– Переглянути інформацію про кандидатів у закріпленій за ним групі;

– Внести оцінки, бали та коментарі до профілю кандидата після перевірки робіт.

При роботі з вебзастосунком батьки (заявники) повинні мати можливість вирішувати такі завдання:

– Створити нову заявку на вступ дитини;

– Переглянути поточний статус власної заявки, дати іспитів та результати оцінювання.

Для забезпечення коректної роботи розробленого вебзастосунку висуваються наступні мінімальні вимоги до технічних та програмних засобів[3].

Вимоги до технічних засобів (апаратного забезпечення):

– процесор: 2-ядерний.

– обсяг оперативної пам'яті: 4 ГБ.

Вимоги до програмних засобів:

– операційна систем Windows 10.

– середовище виконання Node.js[4];

– сучасний веббраузер для доступу до користувацького інтерфейсу системи (рекомендовано Google Chrome або Microsoft Edge).

2 РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ІНФОРМАЦІЙНОГО СУПРОВОДУ ВСТУПНОЇ КАМПАНІЇ ДО ГІМНАЗІЇ

2.1 Аналіз вимог до вебзастосунку для інформаційного супроводу вступної кампанії до гімназії

В роботі застосовується структурна методологія аналізу та проектування розробки. Моделювання функціональних вимог виконано за допомогою варіантів використання.

Розробка варіантів використання до вебзастосунку для інформаційного супроводу вступної кампанії до гімназії. Короткий опис акторів представленно в таблиці 2.1.

Таблиця 2.1 – Короткий опис акторів

Користувач	Короткий опис
Батьки учня	Особи віком 30-45 років, мають середній рівень володіння комп'ютером, зацікавлені в успішному вступі дитини до гімназії. Вимоги: доступ до актуальної інформації про вступ, умови та терміни подачі документів. Основна потреба — отримання інформації про процес вступу та критерії відбору.
Адміністратор приймальної комісії гімназії (Завуч)	Особи віком 23-60 років, мають високий рівень володіння комп'ютером, відповідають за управління процесом вступу, організацію навчального процесу та координацію вступу Вимоги: доступ до повної інформації про заявників, можливість створення звітів, контролю термінів.
Директор гімназії	Особа віком 30-65 років, має високий рівень володіння комп'ютером та значний досвід управління освітнім закладом. Вимоги: доступ до повної інформації про заявників, контроль процесу вступу на кожному етапі, можливість затверджувати остаточні результати відбору. Основна потреба — інструменти для моніторингу процесу вступу та забезпечення прозорості відбору.
Вчителі	Особи віком 25-60 років, мають високий рівень володіння комп’ютером та спеціалізовані знання з перевірки учнівських робіт. Відповідають за оцінювання результатів вступних іспитів, участь у проведенні співбесід та формування рекомендацій для зарахування. Вимоги: доступ до списків кандидатів, результатів їх тестувань, інструментів для внесення оцінок та коментарів. Основна потреба — забезпечення об’єктивної оцінки знань кандидатів із можливістю швидкого обміну інформацією між учасниками комісії.

Виявлені варіанти використання представленні у таблиці 2.2.

Таблиця 2.2 – Виявлення варіантів використання

Код	Основний актор	Назва	Формулювання
1	2	3	4
A1	Батьки школяра	Авторизація	Цей варіант використання дозволяє авторизуватися у профіль «вступника»
A1	Батьки школяра	Додати заяву на вступ	Цей варіант використання дозволяє додати заяву на вступ
A1	Батьки школяра	Корегувати заяву на вступ	Цей варіант використання дозволяє коригувати додану заяву
A1	Батьки школяра	Видалити заяву на вступ	Цей варіант використання дозволяє видалити раніше додану заяву
A1	Батьки школяра	Додати інформацію про школяра	Цей варіант використання дозволяє додати особисту інформацію про школяра
A1	Батьки школяра	Змінити інформацію про школяра	Цей варіант використання дозволяє змінити раніше додану інформацію
A1	Батьки школяра	Видалити інформацію про школяра	Цей варіант використання дозволяє видалити раніше додану інформацію
A1	Батьки школяра	Перегляд результатів іспитів	Цей варіант використання дозволяє переглянути результати іспитів
P1	Адміністратор вступної кампанії(Завуч)	Авторизація	Цей варіант використання дозволяє авторизуватися у профіль адміністратора вступної кампанії
P1	Адміністратор вступної кампанії(Завуч)	Додати оцінку по іспиту	Цей варіант використання дозволяє додати оцінку по іспиту
P1	Адміністратор вступної кампанії(Завуч)	Редагувати результат іспиту	Цей варіант використання дозволяє редагувати раніше введений результат іспиту
P1	Адміністратор вступної кампанії(Завуч)	Видалити результат іспиту	Цей варіант використання дозволяє видалити раніше введений результат іспиту
P1	Адміністратор вступної кампанії(Завуч)	Редагувати статус заяви	Цей варіант використання дозволяє змінити статус заяви (відхилено / прийнято)

Кінець таблиці 2.2

1	2	3	4
B1	Директор гімназії	Моніторингу процесу вступу та забезпечення прозорості відбору	Директор повинен мати доступ до результатів відбору та можливість остаточного затвердження списків прийнятих учнів після завершення всіх етапів перевірки.
G1	Вчителі	Внесення оцінок та коментарів	Вчителі повинні отримувати сповіщення про статус перевірених і неперевірених робіт для кращого управління часом.

Діаграма варіантів використання представлена на рисунку 2.1.

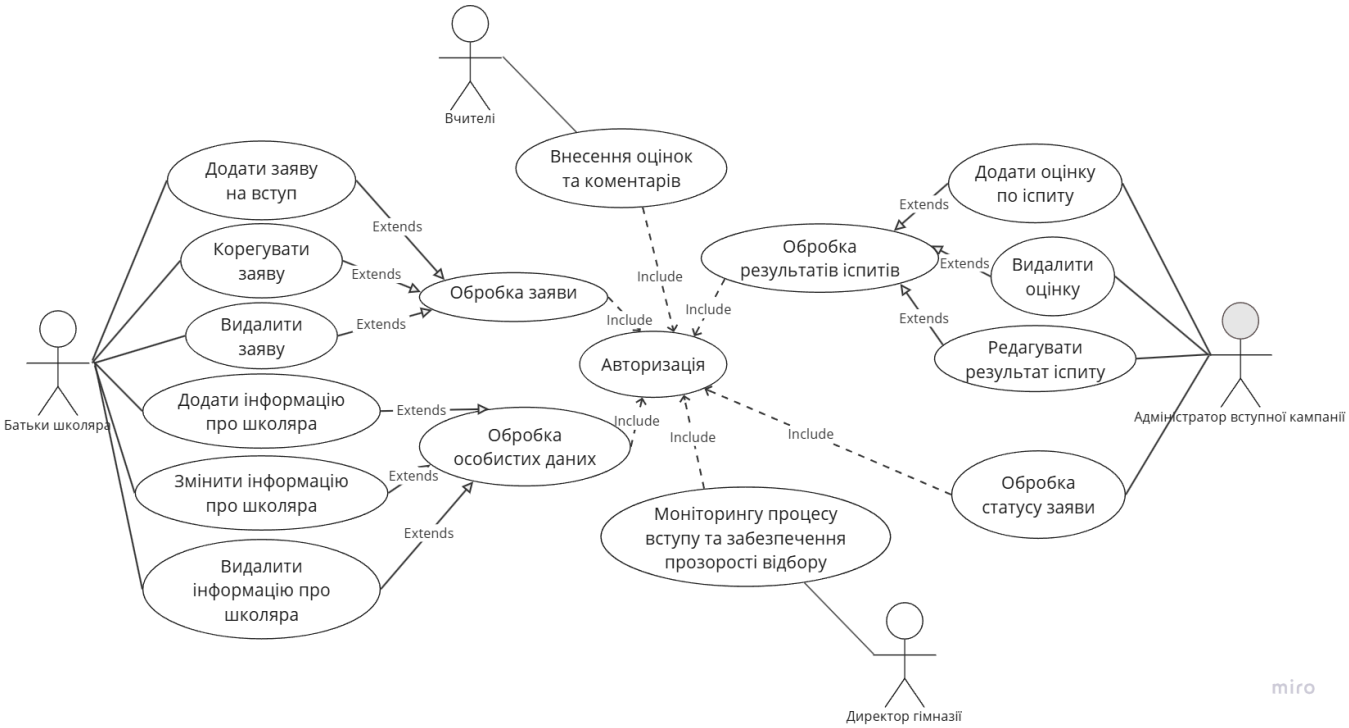

Рисунок 2.1 – Діаграма варіантів використання

Схема охоплює ключові бізнес-процеси, зокрема авторизацію, управління заявами, обробку особистих даних та результатів іспитів, що дозволяє забезпечити прозорість і ефективність проведення вступної кампанії

2.2 Розробка архітектури вебзастосунку для інформаційного супроводу вступної кампанії до гімназії

Проєктування архітектури є критично важливим етапом розробки, оскільки вона визначає структуру системи, взаємодію між її компонентами та забезпечує виконання нефункціональних вимог, таких як масштабованість, безпека та швидкодія. Для вебзастосунку обрано багаторівневу клієнт-серверну архітектуру.

Архітектура клієнтської частини (Frontend) побудована за модульним принципом з використанням компонентно-орієнтованого підходу. Основні складові фронтенд-частини включають:

- Рівень представлення (UI Components): сукупність функціональних компонентів React, що відповідають за візуалізацію даних та взаємодію з користувачем .
- Рівень управління станом (State Management): використовує React Context для збереження глобальних даних (авторизація, мовні налаштування) та TanStack Query для синхронізації стану з сервером [5].
- Рівень маршрутизації (Routing): забезпечує навігацію між сторінками застосунку (кабінет батьків, панель адміністратора тощо) з перевіркою прав доступу на рівні клієнта [6].

Архітектура серверної частини (Backend) базується на середовищі Node.js та фреймворку Express [7]. Логіка сервера організована за наступною схемою:

- Рівень маршрутизації (API Routes): приймає вхідні HTTP-запити від клієнта через префікс /api, проводить первинну валідацію за допомогою схем Zod та спрямовує їх до відповідних контролерів [8].
- Рівень бізнес-логіки (Controllers): реалізує основні алгоритми системи — обробку заявок, розрахунок рейтингів, управління правами користувачів.
- Рівень доступу до даних (Data Access Layer): використовує Drizzle ORM для взаємодії з базою даних SQLite. Це забезпечує безпечне виконання

запитів та автоматичне відображення реляційних даних у об’єкти мови TypeScript [9].

Важливою особливістю архітектури є наявність спільного шару (Shared Schema), що містить визначення типів даних та правил валідації. Це гарантує цілісність даних на всіх етапах: від введення користувачем у форму на фронтенді до збереження в базі даних на бекенді.

Взаємодія між компонентами системи та її фізичне розгортання відображені за допомогою UML-діаграм, опис яких наведено нижче.

Діаграма компонентів показує основні модулі вебзастосунку та їх взаємодію представлено на рисунку 2.2. Діаграма розміщення представлено на рисунку 2.3.

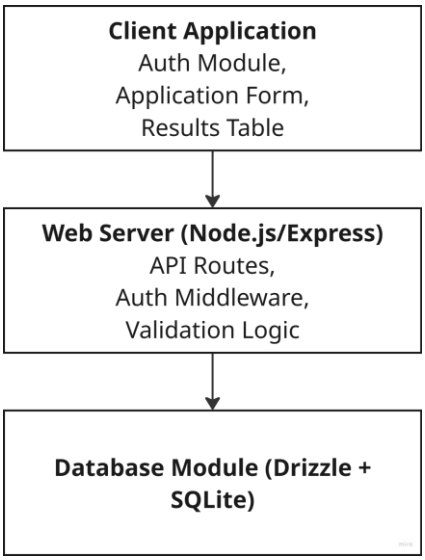

Рисунок 2.2 – Діаграма компонентів програмної системи

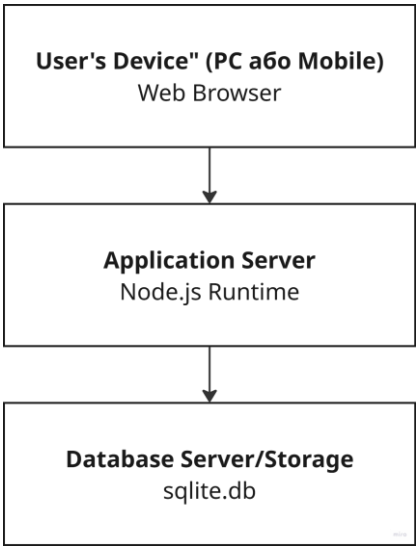

Рисунок 2.3 – Діаграма розміщення програмної системи

Обрана архітектурна модель базується на розділенні зовсім різних за функціональним призначенням частин: інтерфейсу користувача (Frontend) та логіки обробки даних (Backend). Такий підхід дозволяє розробляти та тестувати кожну частину незалежно, що підвищує загальну надійність програмного продукту.

2.3 Розробка проєкту програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

2.3.1 Ескізний проект програмного забезпечення

Розробка концептуальної моделі бази даних до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Сутності та їх атрибути концептуальної моделі:

Інформація про школяра:

Програмне забезпечення повинно зберігати такі дані про кожного школяра:

- ID школяра – унікальний ідентифікатор.
- Ім’я та прізвище школяра.
- Дата народження.
- Стать.
- Адреса проживання.
- Контактний номер телефону школяра.
- Контактний номер телефону батька/матері.
- Середній бал – оцінка за результатами навчання.
- Заклад освіти (School) – школа, де навчався школяр.

Інформація про заклад освіти:

Для кожного навчального закладу, з якого батьки школяра подають документи на вступ, зберігається така інформація:

- Ідентифікатор закладу.
- Назва закладу.
- Адреса закладу.
- Контактна інформація (телефон, електронна пошта).

Інформація про вступні іспити:

Для школярів можуть проводитись вступні випробування або тести. Для таких іспитів зберігається наступна інформація:

- ID іспиту – унікальний номер іспиту.
- Назва іспиту.
- Дата проведення іспиту.

Результати іспитів:

Кожен школяр, який складав вступні іспити, матиме персональні результати, для яких фіксується:

- Номер результату.
- ID школяра.
- ID іспиту.
- Оцінка.

Інформація про подані заяви:

Батьки школярів повинні подати заяви для вступу до гімназії.

Програмне забезпечення повинно зберігати таку інформацію про кожну заяву:

- Ідентифікатор заяви.
- ID школяра.
- Дата подання заяви.
- Статус заяви (подана, розглядається, прийнята, відхилена).

Зв’язки між сутностями представлені в табл. 2.3

Таблиця 2.3 – Зв’язки між сутностями

Назва сутності	Назва сутності	Зв’язок	Пояснення
1	2	3	4
Школярі	Заяви на вступ	1:1	Один школяр може подати одну заяву

Назва сутності	Назва сутності	Зв’язок	Пояснення
1	2	3	4
Іспити	Результати іспитів	1:M	Один іспит може мати декілька результатів іспитів

Кінець таблиці 2.3

1	2	3	4
Заява на вступ	Результати іспитів	1:M	Одна заява на вступ може мати декілька результатів іспитів
Школярі	Навчальні заклади	M:1	Один заклад вказує багато школярів

Побудова діаграми "сутність-зв'язок" до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Концептуальна модель, у вигляді діаграми сутність-зв’язок, представлена на рис. 2.4.

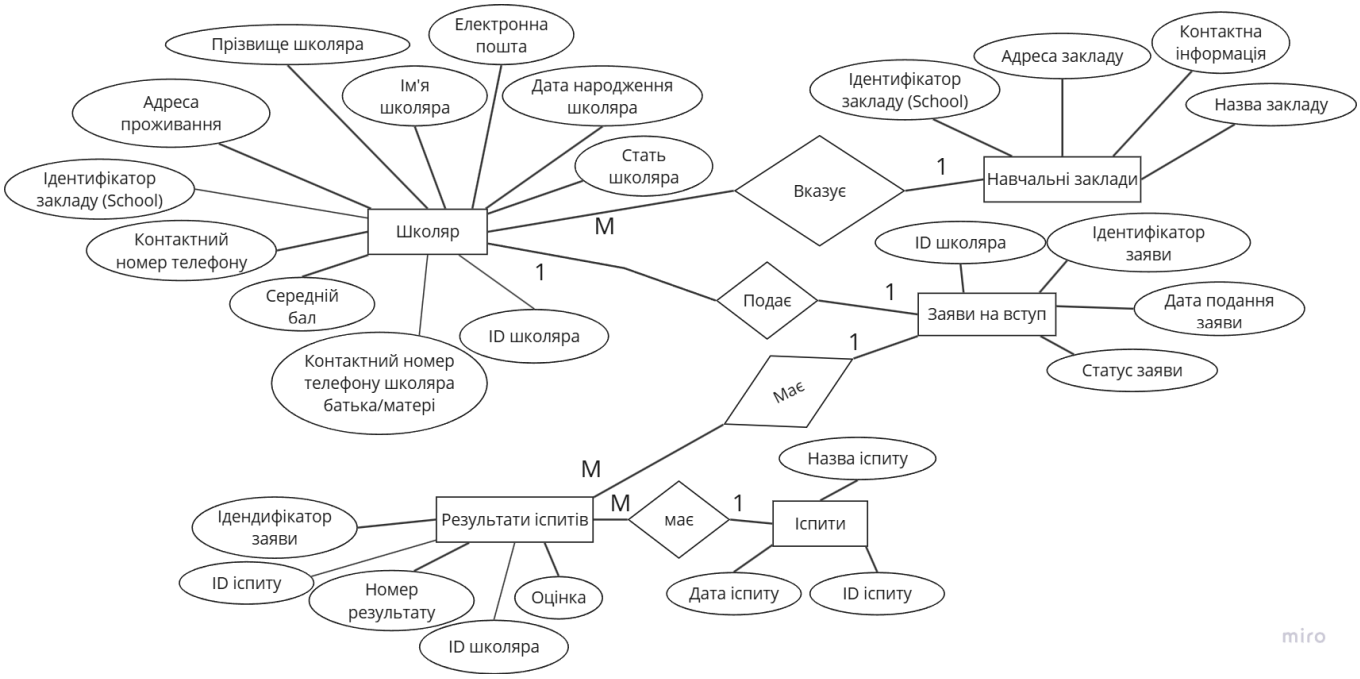

Рисунок 2.4 – Концептуальна модель програмного забезпечення

Розробка графічного інтерфейсу до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Проробивши аналіз предметної галузі, сформувавши профілі і сценарії дій користувачів, визначивши функціональність програми можна скласти схему навігаційної системи для «Батьків вступника», яка зображена на рисунку 2.5.

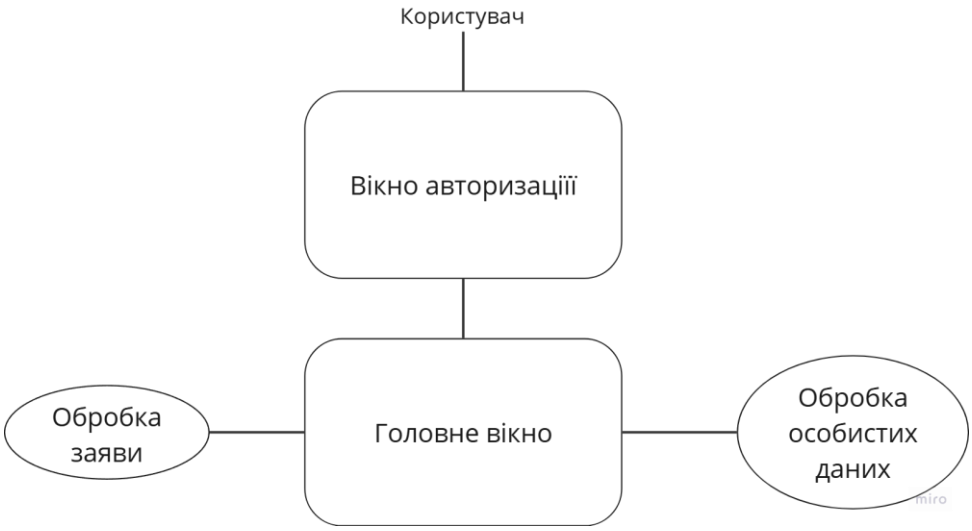

Рисунок 2.5 – Схема навігаційної системи для «Батьків вступника»

Схема навігаційної системи для «Адміністратора вступної кампанії» зображена на рисунку 2.6

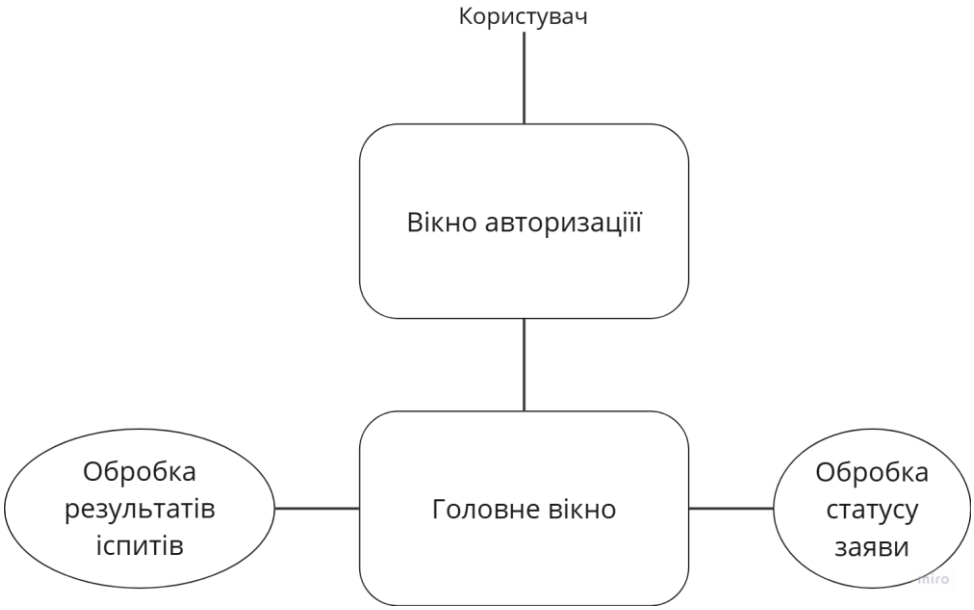

Рисунок 2.6 – Схема навігаційної системи для «Адміністратора вступної кампанії»

На паперовому етапі прототип може пережити багато виправлень і, відповідно, багато версій . Прототипи вікон програми приведено на рисунках 2.7 – 2.8.

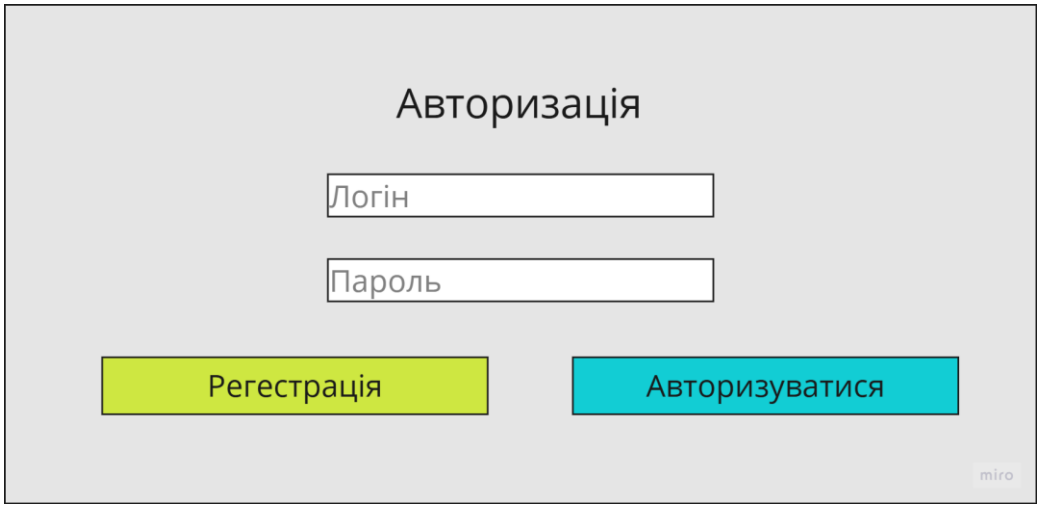

Рисунок 2.7 – Прототип вікна авторизації

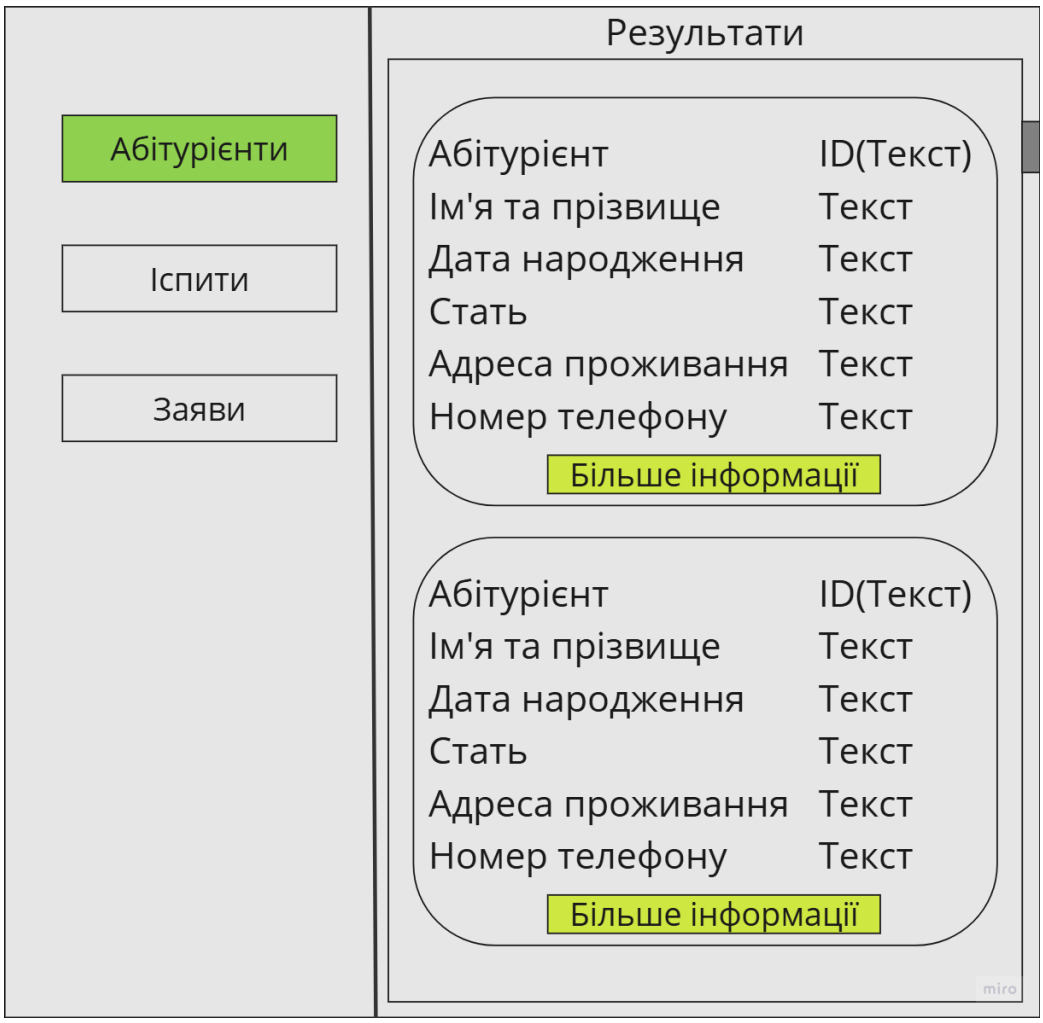

Рисунок 2.8 – Прототип вікна програми (Абітурієнти), якщо користувач є «Адміністратор вступної кампанії

Користувацький інтерфейс є важливою частиною програмного забезпечення. Від нього залежить ефективність роботи працівника й точність управління машиною з сторони людини.

2.3.2 Технічний проєкт до програмного забезпечення

Розробка діаграми класів до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії.

У разі застосування об’єктно-орієнтованої методології, в цьому підрозділі наводяться статичні та динамічні моделі реалізацій варіантів використання.

Статична модель представляється діаграмами класів і специфікаціями класів. Специфікація класу включає:

- ім’я;
- відповідальність класу;
- атрибути;
- операції зі специфікаціями нетривіальних специфікацій;
- діаграму станів.

Динамічна модель представляється кооперaціями, що є реалізаціями варіантів використання й подається діаграмами взаємодії.

Діаграму класів застосунку, представлено на рис. 2.9.

Специфікація основних класів програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Класс: «Authorization»

Атрибути:

- username – ім’я користувача;
- password – пароль користувача;
- role – роль користувача (батьки, вчителі, завуч, директор).

Методи:

- login() – перевірка даних для входу в систему;
- logout() – вихід із системи;
- authorize() – отримання доступу відповідно до ролі.

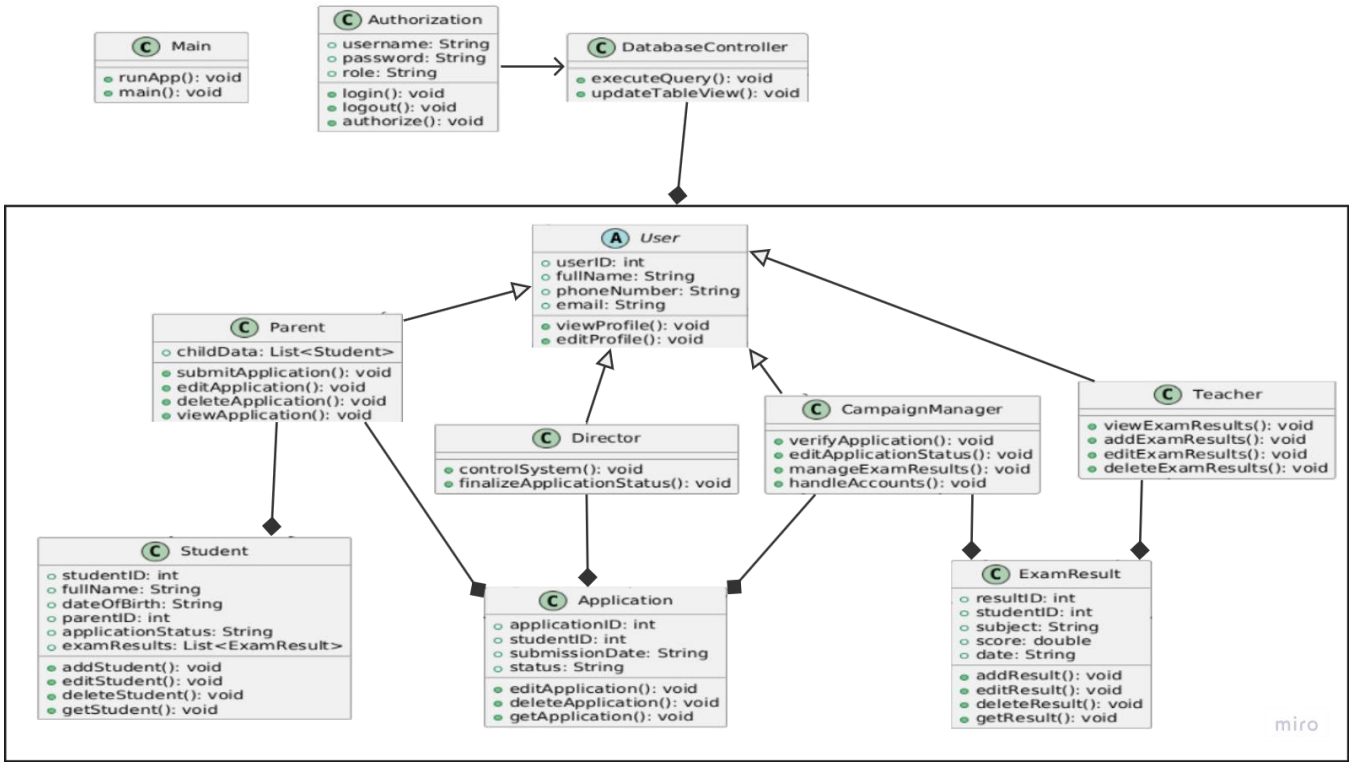

Рисунок 2.9 – Діаграма классів до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Класс: «User (Абстрактний клас)»

Атрибути:

- userID – унікальний ідентифікатор користувача;
- fullName – повне ім’я користувача;
- phoneNumber – номер телефону користувача;
- email – електронна пошта користувача.

Методи:

- viewProfile() – перегляд профілю;
- editProfile() – редагування профілю.

Класс: «Parent (extends User)»

Атрибути:

- childData – список даних про учнів (дітей).
- Методи:
- submitApplication() – подання заяви на вступ;

- editApplication() – редагування поданої заяви;
- deleteApplication() – видалення поданої заяви;
- viewApplication() – перегляд поданих заявок.

Класс: «Teacher (extends User)»

Методи:

- viewExamResults() – перегляд результатів іспитів;
- addExamResults() – додавання результатів іспитів;
- editExamResults() – редагування результатів іспитів;
- deleteExamResults() – видалення результатів іспитів.

Класс: «CampaignManager (extends User)»

Методи:

- verifyApplication() – перевірка поданих заявок;
- editApplicationStatus() – редагування статусу заявок («Прийнято», «Відхилено»);
- manageExamResults() – ведення результатів іспитів;
- handleAccounts() – робота з обліковими даними користувачів.

Класс: «Director (extends User)»

Методи:

- controlSystem() – загальний контроль системи;
- finalizeApplicationStatus() – остаточне затвердження статусу зарахування.

Класс: «Student»

Атрибути:

- studentID – унікальний ідентифікатор студента;
- fullName – повне ім’я студента;
- dateOfBirth – дата народження студента;

- parentID – ідентифікатор батьків;
- applicationStatus – статус заяви студента («На розгляді», «Прийнято», «Відхилено»);
- examResults – список результатів іспитів.

Методи:

- addStudent() – додавання нового студента;
- editStudent() – редагування інформації про студента;
- deleteStudent() – видалення інформації про студента;
- getStudent() – отримання інформації про студента.

Класс: «Application»

Атрибути:

- applicationID – унікальний ідентифікатор заявки;
- studentID – ідентифікатор студента;
- submissionDate – дата подання заявки;
- status – статус заявки.

Методи:

- editApplication() – редагування заявки;
- deleteApplication() – видалення заявки;
- getApplication() – отримання інформації про заявку.

Класс: «ExamResult»

Атрибути:

- resultID – унікальний ідентифікатор результату;
- studentID – ідентифікатор студента;
- subject – назва предмета;
- score – оцінка за іспит;
- date – дата складання іспиту.

Методи:

- addResult() – додавання результату іспиту;

- editResult() – редагування результату іспиту;
- deleteResult() – видалення результату іспиту;
- getResult() – отримання результату іспиту.

Класс: «DatabaseController»

Методи:

- executeQuery() – виконання SQL-запиту для взаємодії з базою даних;
- updateTableView() – оновлення даних у таблицях інтерфейсу.

Побудова логічної моделі даних до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Логічна модель даних, наведена на рис. 2.10.

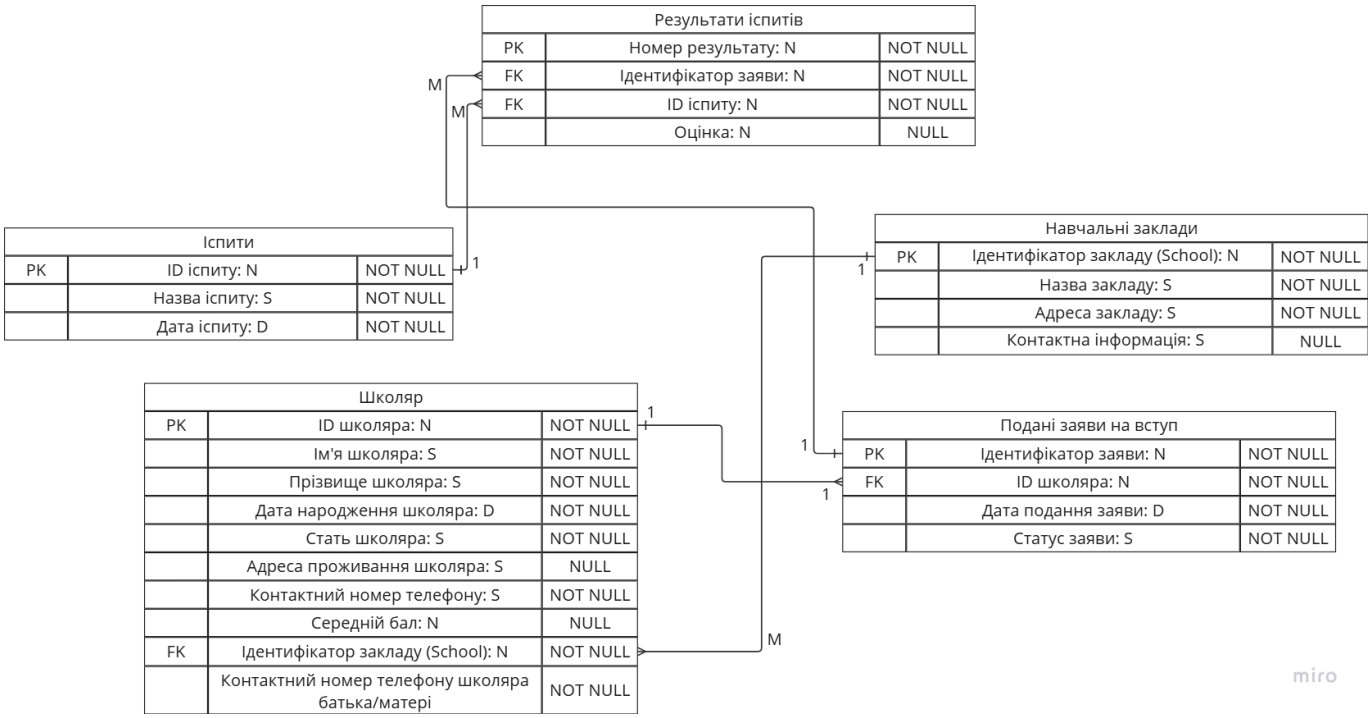

Рисунок 2.10 – Логічна модель даних

Таблиця логічної моделі даних елементу «Школярі», наведена в табл. 2.4

Таблиця 2.4 – Логічна модель даних елементу «Школярі»

Атрибут	Тип даних	Ключ	Обов’язковість заповнення
1	2	3	4
ID школяра	N	PK	NOT NULL
Ім'я школяра	S		NOT NULL
Прізвище школяра	S		NOT NULL
Дата народження школяра	D		NOT NULL
Стать школяра	S		NOT NULL
Адреса проживання школяра	S		NULL
Контактний номер телефону	S		NOT NULL
Середній бал	N		NULL
Ідентифікатор закладу	N	FK	NOT NULL
Контактний номер телефону батька/матері	S		NOT NULL

Таблиця логічної моделі даних елементу «Навчальні заклади», наведена в табл. 2.5

Таблиця 2.5 – Логічна модель даних елементу «Навчальні заклади»

Атрибут	Тип даних	Ключ	Обов’язковість заповнення
Ідентифікатор закладу	N	PK	NOT NULL
Назва закладу	S		NOT NULL
Адреса закладу	S		NOT NULL
Контактна інформація	S		NULL

Таблиця логічної моделі даних елементу «Вступні іспити», наведена в табл. 2.6

Таблиця 2.6 – Логічна модель даних елементу «Вступні іспити»

Атрибут	Тип даних	Ключ	Обов’язковість заповнення
ID іспиту	N	PK	NOT NULL
Назва іспиту	S		NOT NULL
Дата іспиту	D		NOT NULL

Таблиця логічної моделі даних елементу «Результати іспитів», наведена в табл. 2.7

Таблиця 2.7 – Логічна модель даних елементу «Результати іспитів»

Атрибут	Тип даних	Ключ	Обов’язковість заповнення
Номер результату	N	PK	NOT NULL
Ідентифікатор заяви	N	FK	NOT NULL

Атрибут	Тип даних	Ключ	Обов’язковість заповнення
ID іспиту	N	FK	NOT NULL
Оцінка	N		NULL

Таблиця логічної моделі даних елементу «Подані заяви», наведена в табл. 2.8

Таблиця 2.8 – Логічна модель даних елементу «Подані заяви»

Атрибут	Тип даних	Ключ	Обов’язковість заповнення
Ідентифікатор	N	PK	NOT NULL
ID школяра	N	FK	NOT NULL
Дата подання	D		NOT NULL
Статус заяви	S		NOT NULL

Представлені таблиці деталізують структуру всіх необхідних сутностей системи та визначають правила їх взаємодії на рівні реляційних зв'язків.

2.3.3 Робочий проєкт програмного забезпечення

Використані СКБД та інші програмні засоби до програмного забезпечення

Для успішної реалізації, тестування та розгортання програмного забезпечення, було використано комплекс сучасних програмних засобів, інструментів розробки та систему управління базами даних, що відповідають вимогам надійності, швидкодії та кросплатформенності.

В якості системи управління базами даних (СКБД) обрано реляційну СКБД SQLite. Головною особливістю SQLite є її безсерверна архітектура (serverless) — база даних зберігається в одному локальному файлі («sqlite.db»), що мінімізує витрати на адміністрування та значно спрощує процес розгортання і резервного копіювання системи у межах локальної інфраструктури навчального закладу. Для забезпечення високої продуктивності виконання операцій взаємодія з базою даних на рівні платформи Node.js реалізована за допомогою бібліотеки better-sqlite3.

Для об’єктно-реляційного відображення та управління міграціями структури даних застосовано інструмент Drizzle ORM[10]. Вибір цього засобу зумовлений суворою типізацією TypeScript та можливістю використання спільної схеми даних як для клієнтської, так і для серверної частин застосунку. Валідація моделей даних та перевірка вхідних параметрів на відповідність заданим типам забезпечується інтегрованою бібліотекою Zod.

Написання програмного коду, проєктування архітектури компонентів та налагодження вебзастосунку здійснювалося у середовищі розробки Visual Studio Code (VS Code). Вибір цієї IDE зумовлений її високою швидкодією, низьким споживанням системних ресурсів та наявністю широкого спектра розширень для роботи з екосистемою JavaScript і TypeScript. Під час виконання роботи активно використовувалися вбудовані інструменти керування версіями (Git), інтегрований термінал для керування залежностями і запуску локального сервера розробки, а також спеціалізовані плагіни для автоматичного форматування коду (Prettier) та статичного аналізу (ESLint).

Серверна частина системи функціонує в середовищі виконання Node.js, яке дозволяє створювати масштабовані мережеві застосунки завдяки асинхронній моделі керування подіями (Event-driven I/O). Безпосередня обробка HTTP-запитів від клієнта, реалізація захищених маршрутів (Protected Routes) та побудова архітектурного стилю REST API реалізовані на базі вебфреймворку Express.

Для організації процесу розробки фронтенд-частини та швидкого збирання модулів проєкту використано інструмент Vite [11]. Завдяки інтеграції з компілятором SWC (Speedy Web Compiler) забезпечується швидка компіляція та миттєве оновлення модулів у браузері (Hot Module Replacement) без втрати поточного стану застосунку. Клієнтська візуальна частина побудована на базі бібліотеки React 18 та фреймворку Tailwind CSS [12]. Для утилітарної стилізації інтерфейсу та готових примітивів бібліотеки компонентів shadcn/ui [13] .

Фрагменти коду до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії.

Приведено інтерфейс та база даних.

Найбільш показові фрагменти кодів:

1) Оновлення існуючого запису (Update)

```
async updateApplication(id: number, application:
Partial<InsertApplication & { status?: string }>):
Promise<Application | undefined> {
  // Оновлюємо лише ті поля заяки, які були передані в об'єкті
application
  const [updated] = await db.update(applications)
    .set(application as any)
    .where(eq(applications.id, id))
    .returning();
  return updated;
  }
```

2) Каскадне видалення записів (Delete)

```
async deleteUser(id: number): Promise<boolean> {
  // 1. Отримуємо всіх дітей цього користувача
  const userChildren = await
db.select().from(children).where(eq(children.parentId, id));
  // 2. Видаляємо всі пов'язані дані для кожної дитини (оцінки,
результати іспитів)
  for (const child of userChildren) {
    await db.delete(examResults).where(eq(examResults.childId,
child.id));
    await db.delete(grades).where(eq(grades.childId, child.id));
    await db.delete(studentGrades).where(eq(studentGrades.childId,
child.id));
  }
  // 3. Видаляємо заявки та профілі дітей
  await db.delete(applications).where(eq(applications.parentId,
id));
  await db.delete(children).where(eq(children.parentId, id));
  // 4. Тільки після цього безпечно видаляємо самого користувача
  await db.delete(users).where(eq(users.id, id));
  return true;
}
```

Цей фрагмент демонструє сучасний підхід до проєктування бази даних за допомогою ORM (Object-Relational Mapping). Код ілюструє строгу типізацію на рівні бази даних, використання зв'язків (foreign keys), а також автоматичну

генерацію схем валідації (zod), що забезпечує безпеку даних між фронтендом та бекендом.

3) Складний компонент інтерфейсу користувача з управлінням станами (React)

```
import { useState, useMemo, useEffect } from 'react';
import { updateApplicationStatus, updateApplication } from
'@/utils/dataService';
import { useLanguage } from '@/context/LanguageContext';

const ApplicationTable = ({ applications, users,
onApplicationUpdate }: ApplicationTableProps) => {
  const { t, currentLanguage } = useLanguage();
  const [searchQuery, setSearchQuery] = useState('');
  const [statusFilter, setStatusFilter] = useState<string>('all');
  const [selectedApp, setSelectedApp] = useState<Application |
null>(null);

  // Оптимізація продуктивності через useMemo для пошуку та
фільтрації
  const filteredApplications = useMemo(() => {
    return applications.filter(application => {
      const nameMatch =
application.fullName.toLowerCase().includes(searchQuery.toLowerCas
e());
      const statusMatch = statusFilter === 'all' ||
application.status === statusFilter;
      return nameMatch && statusMatch;
    });
  }, [applications, searchQuery, statusFilter]);

  // Обробник прийняття рішення директором
  const handleAction = async (status: ApplicationStatus) => {
    if (!selectedApp) return;
    try {
      if (status !== selectedApp.status) {
        await updateApplicationStatus(selectedApp.id, status,
selectedApp.childId);
      }
      if (onApplicationUpdate) {
        await onApplicationUpdate(); // Оновлення даних таблиці
      }
      setSelectedApp(null);
    } catch (e) {
      console.error("Error updating status:", e);
    }
  };

  return (
```

```
    <div className="rounded-lg border bg-white shadow-md overflow-
hidden">
      {/* Панель фільтрів */}
      <div className="p-6 flex items-center gap-4 border-b">
        <Input
          placeholder={currentLanguage === 'ua' ? "Пошук за
ім'ям..." : "Search by name..."}
          value={searchQuery}
          onChange={(e) => setSearchQuery(e.target.value)}
        />
        <select value={statusFilter} onChange={(e) =>
setStatusFilter(e.target.value)}>
          <option value="all">{t('all.statuses')}</option>
          <option value="PENDING">Pending</option>
          <option value="APPROVED">Approved</option>
        </select>
      </div>

      {/* Таблиця рендеріться на основі відфільтрованих даних...
*/}
    </div>
  );
};
```

Тестування програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії.

Обґрунтування вибору тестування до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Тестування є невід’ємним етапом життєвого циклу розробки програмного забезпечення, що дозволяє виявити та усунути помилки до введення системи в експлуатацію. Для перевірки працездатності вебзастосунку було обрано метод функціонального тестування (тестування за принципом «чорної скриньки»). Цей підхід передбачає перевірку відповідності реальної поведінки системи до заданих функціональних вимог без втручання у внутрішній вихідний код.

Тестування проводилося у локальному середовищі з використанням сучасних веббраузерів (Google Chrome, Mozilla Firefox) на операційній системі Windows 10. У процесі тестування перевірялася коректність роботи інтерфейсу користувача, валідація вхідних даних, розмежування прав доступу та обробка інформації на сервері.

Тестування чорного ящика до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Тестування модуля авторизації та маршрутизації Першим етапом було перевірено систему розмежування прав доступу. При спробі неавторизованого доступу до захищених сторінок (наприклад, панелі адміністратора або сторінки виставлення оцінок) Програмне забезпечення коректно перенаправляє неавторизованого користувача на сторінку входу. Після успішної авторизації підтверджено, що інтерфейс та навігаційне меню адаптуються відповідно до ролі користувача (Батьки, Вчитель, Адміністратор, Директор).

Форма авторизації користувача в системі зображена на рисунку 2.11.

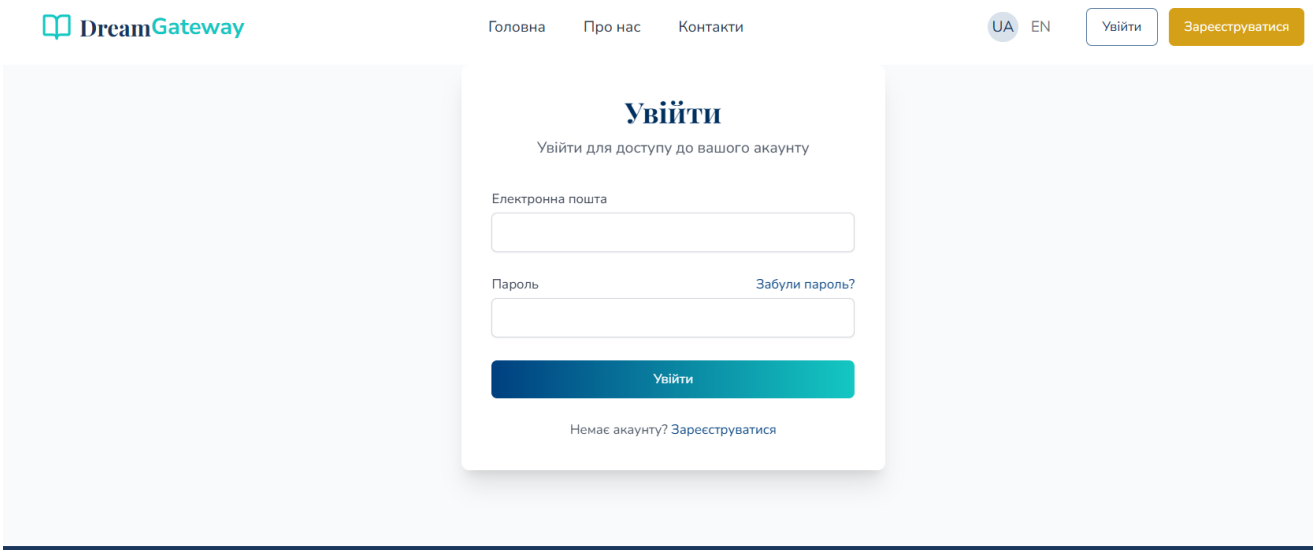

Рисунок 2.11 – Форма авторизації користувача в системі

Тестування функціоналу подачі заявки (роль «Батьки») Під час тестування процесу створення нової заяви на вступ перевірялася робота механізмів валідації форм на стороні клієнта. При введенні некоректних даних (наприклад, недійсний формат електронної пошти, введення тексту замість цифр у номері телефону або пропущені обов'язкові поля) Програмне забезпечення миттєво виводить відповідні текстові попередження. Цей механізм успішно реалізовано за допомогою інтеграції бібліотеки React Hook Form та валідатора Zod [14]. При

введенні валідних даних заявка успішно відправляється на сервер, зберігається в базі даних SQLite, а її статус за замовчуванням встановлюється як «Нова»[15] .

Для забезпечення коректності введених даних користувачем у системі реалізовано механізм валідації на стороні клієнта. Якщо користувач вводить дані, що не відповідають заданим критеріям (наприклад, невірний формат номера телефону), Програмне забезпечення блокує можливість збереження форми та виводить інформаційне повідомлення про помилку (рис. 2.12).

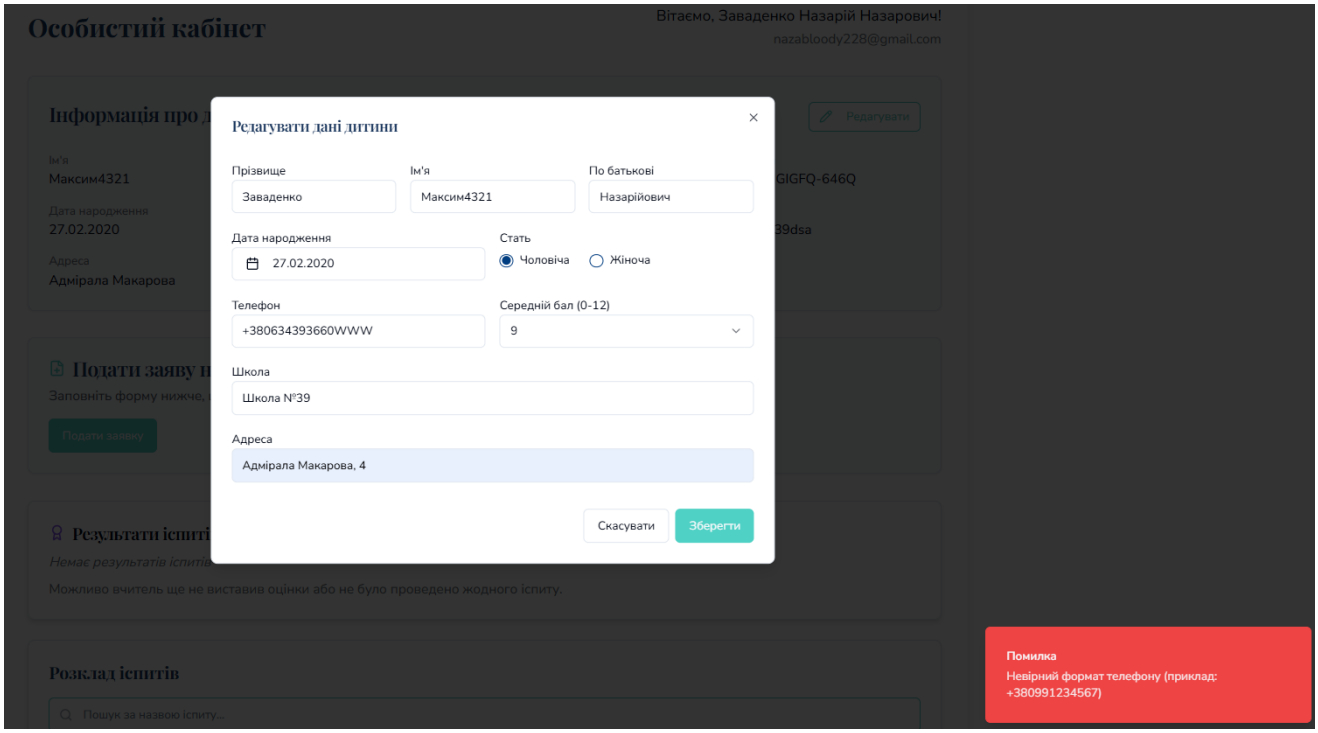

Рисунок 2.12 – Процес заповнення форми подачі заявки з відображенням помилок валідації

Після успішного заповнення всіх обов'язкових полів та проходження перевірки на валідацію, дані зберігаються у базі даних. Новий запис автоматично з'являється у списку зі статусом "В очікуванні". Відображення успішно поданої заявки в особистому кабінеті заявника наведено на рисунку 2.13

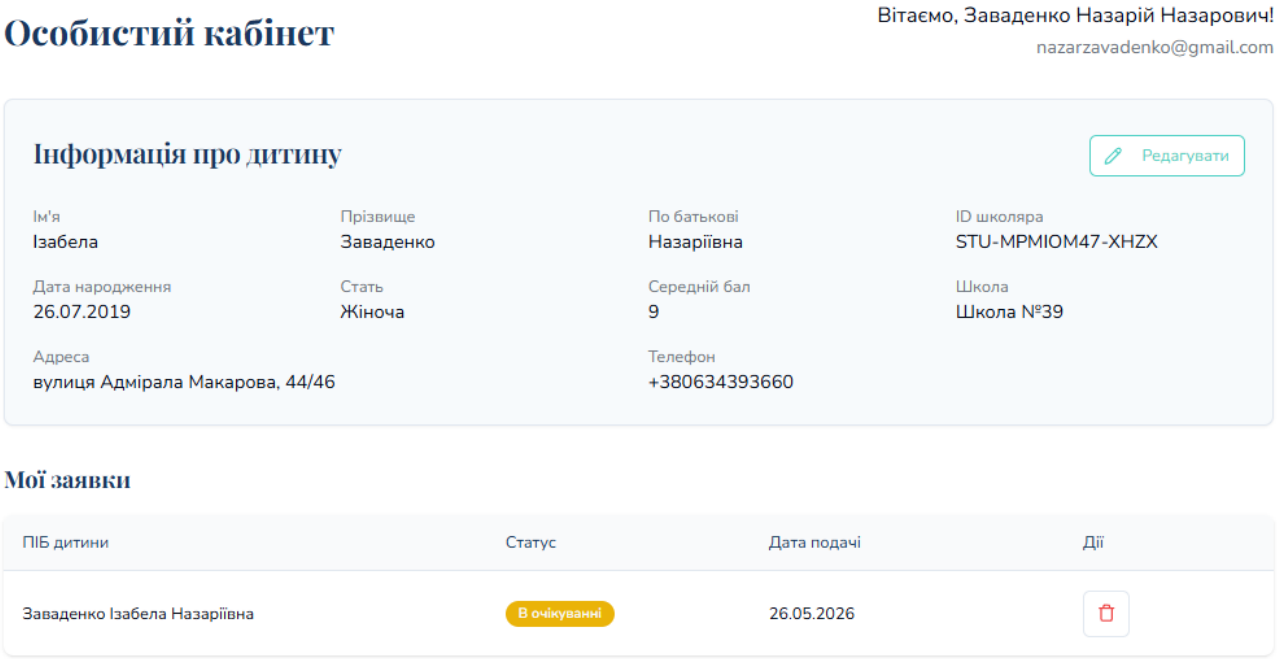

Рисунок 2.13 – Відображення успішно поданої заявки в особистому кабінеті заявника

Тестування особистого кабінету адміністратора приймальної комісії (Завуча)

Кабінет завуча є основним центром управління вступною кампанією, що вимагає перевірки функцій моніторингу, пошуку, фільтрації та конфігурування даних. Тестування цього модуля було розділено на три послідовні кроки:

1. Перевірка головного інтерфейсу та системи фільтрації заявок. Під час перевірки було здійснено вхід під обліковим записом завуча. Програмне забезпечення успішно відобразила загальну таблицю даних, що містить повний перелік поданих заяв. Перевірено роботу пошукового рядка за прізвищем кандидата, а також випадаючих списків фільтрації за поточним статусом заяви. Дані оновлюються миттєво без перезавантаження сторінки завдяки асинхронній інтеграції з сервером.

Головна сторінка кабінету завуча з переліком заяв зображено на рисунку 2.14.

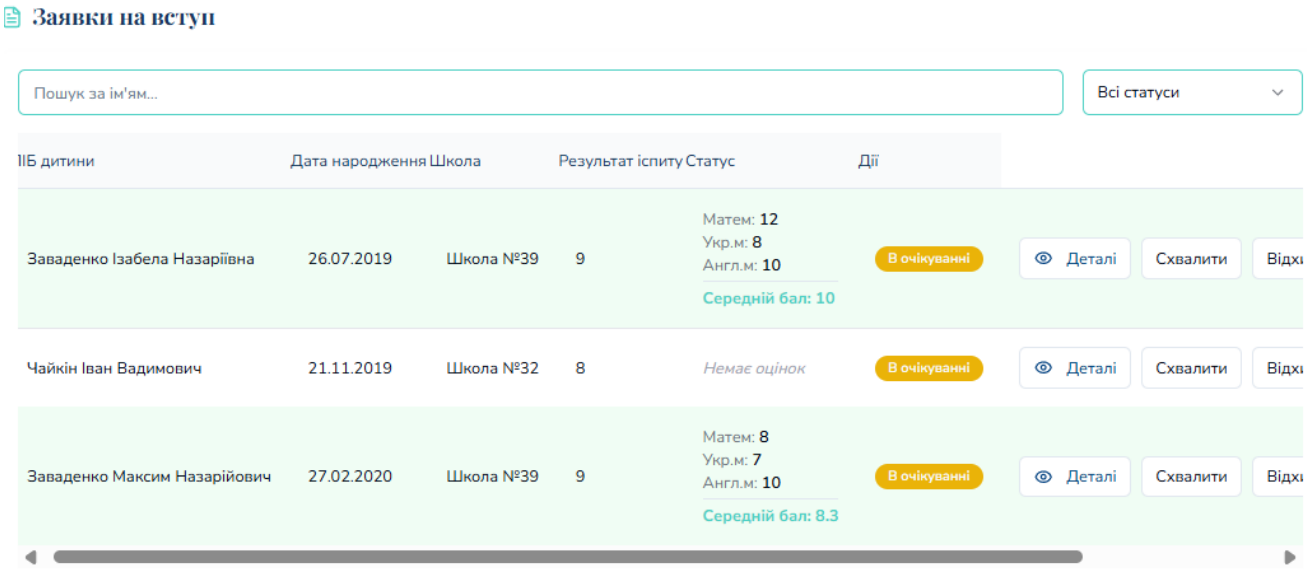

Рисунок 2.14 – Головна сторінка кабінету завуча з переліком заяв

2. Тестування механізму детального перегляду профілю кандидата та модифікації статусів. При виборі конкретного запису із загального списку відкривається детальна картка абітурієнта. Перевірено коректність функціонування інтерактивного випадаючого меню зміни статусів. При зміні значення (наприклад, переведення заяви зі статусу «Нова» у статус «Документи прийнято») Програмне забезпечення надсилає REST-запит до серверної частини на базі Express, оновлює відповідне поле у базі даних SQLite та повертає підтвердження, яке візуалізується у вигляді спливаючого сповіщення (toast notification).

Інтерфейс редагування картки кандидата та зміни статусу заяви зображено на рисунку 2.15.

3. Перевірка модуля адміністрування користувачів та формування екзаменаційних груп. Було протестовано функціонал створення облікових записів для викладачів, призначення модераторів та розподілу кандидатів за розкладом вступних випробувань. Програмне забезпечення продемонструвало стабільність під час обробки зв'язаних реляційних моделей даних.

Розділ конфігурування користувачів та екзаменаційних списків у кабінеті завуча зображено на рисунку 2.16.

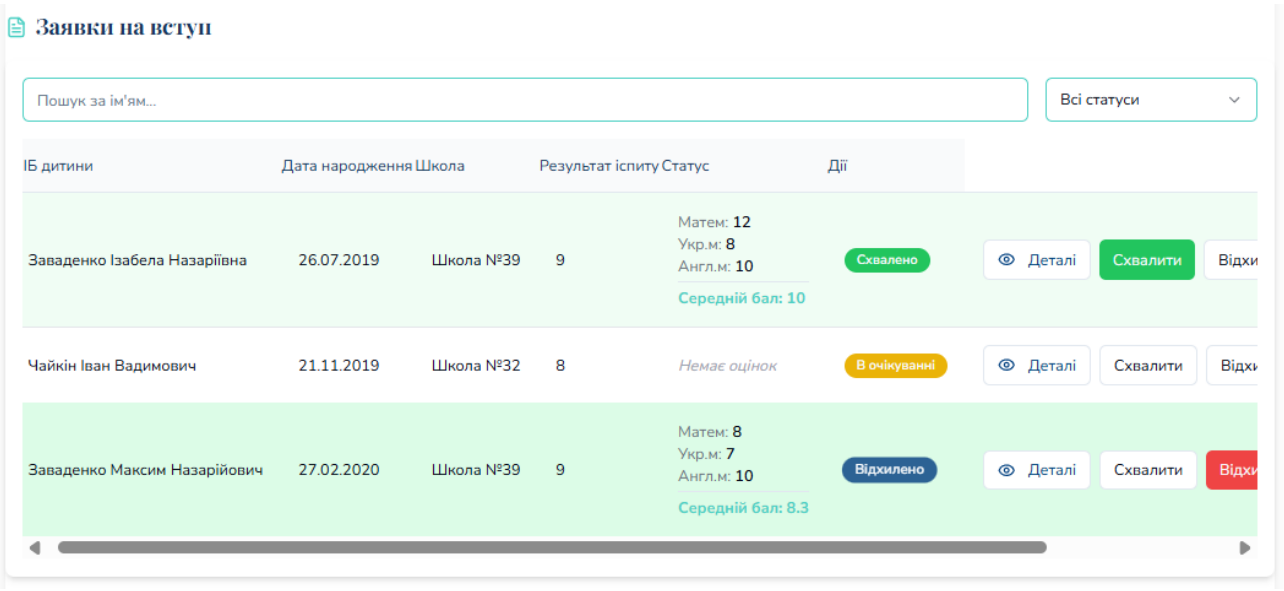

Рисунок 2.15 – Інтерфейс редагування картки кандидата та зміни статусу заяви

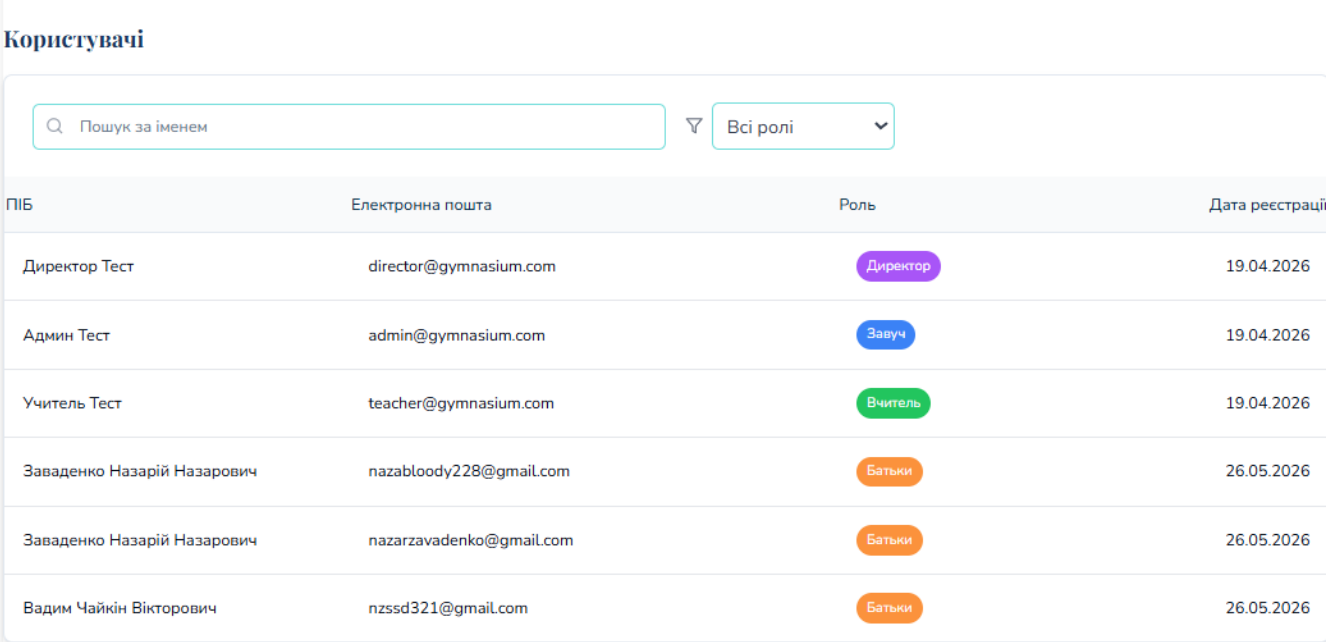

Рисунок 2.16 – Розділ конфігурування користувачів та екзаменаційних списків у кабінеті завуча.

Тестування особистого кабінету директора гімназії

Кабінет керівника освітнього закладу орієнтований на високорівневий моніторинг показників вступної кампанії та винесення фінальних рішень щодо зарахування учнів. Тестування охоплювало два ключові етапи:

1. Перевірка аналітичного дашборду. При автентифікації користувача з роллю «Директор» Програмне забезпечення завантажує екран статистичного аналізу. Було перевірено точність відображення числових метрик (загальна кількість поданих заяв, відсоток схвалених документів, поточний конкурс на одне місце) та графічних компонентів, що візуалізують динаміку реєстрації. Всі розрахунки на стороні сервера виконуються коректно та відповідають реальному стану бази даних.

Аналітична панель кабінету директора з відображенням статистики зображено на рисунку 2.17.

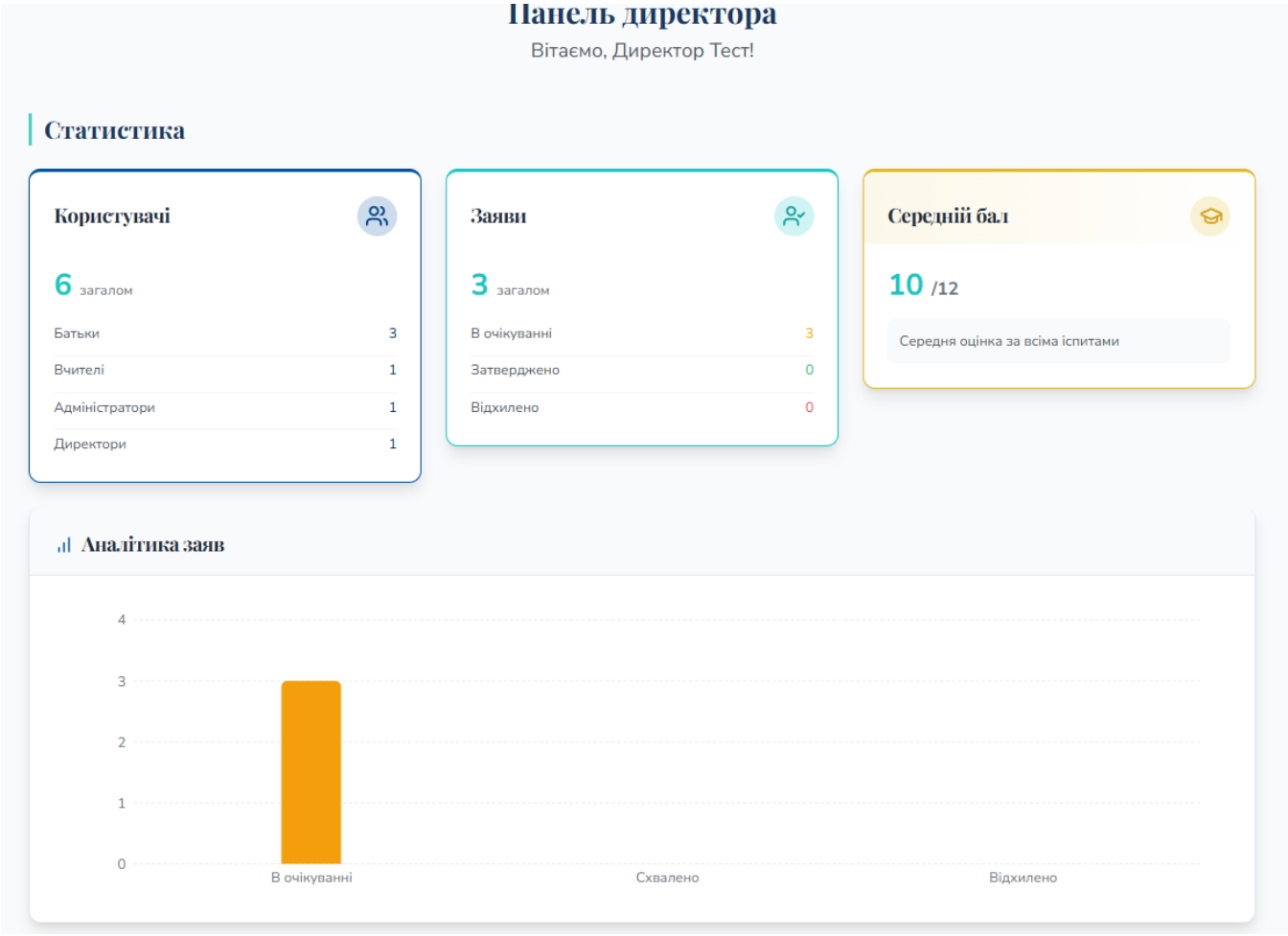

Рисунок 2.17 – Аналітична панель кабінету директора з відображенням статистики

2. Тестування функції остаточного затвердження списків. Директору доступний фінальний рейтинговий список абітурієнтів, сформований на основі

отриманих балів. Перевірено працездатність модуля накладання електронної резолюції. При масовому виділенні рекомендованих до зарахування учнів та активації функції затвердження, Програмне забезпечення успішно переводить їхні заяви у статус «Зараховано», закриваючи можливість подальшого редагування іншими користувачами.

Інтерфейс затвердження рейтингових списків та винесення фінального рішення зображено на рисунку 2.18

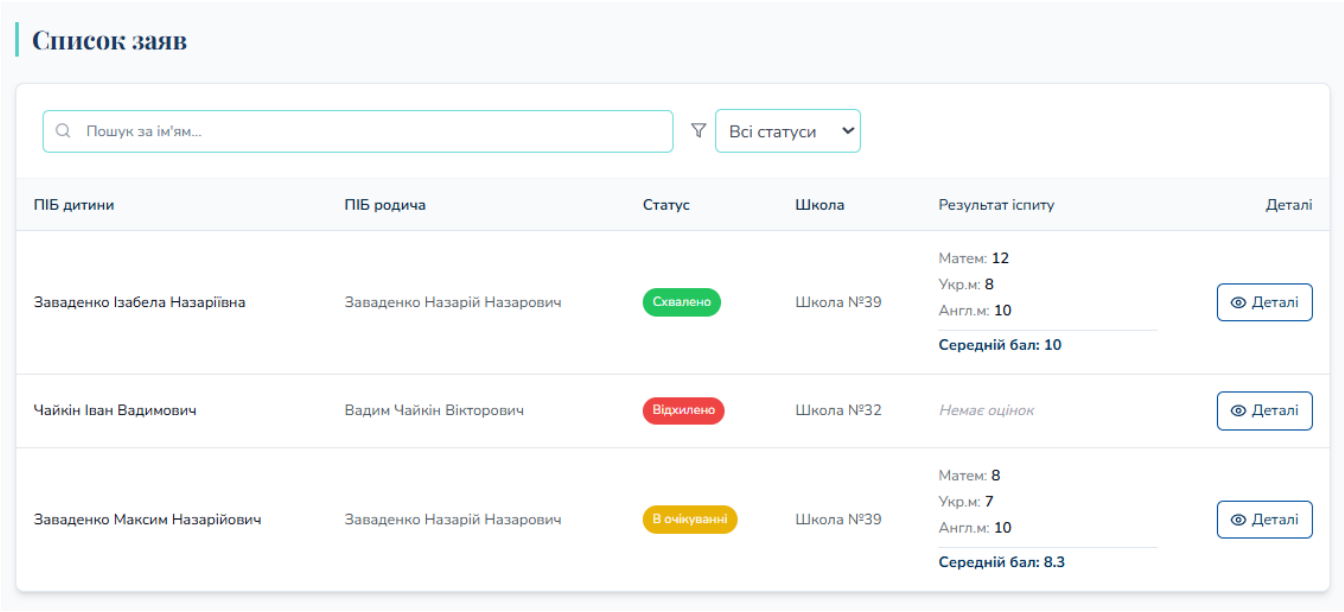

Рисунок 2.18 – Інтерфейс затвердження рейтингових списків та винесення фінального рішення

Тестування особистого кабінету вчителя (екзаменатора)

Робоче місце вчителя має чітко обмежені права доступу і призначене виключно для фіксації результатів конкурсного відбору.

Під час тестування кабінету вчителя було перевірено модуль виставлення оцінок. Після авторизації викладач отримує доступ лише до тих екзаменаційних груп та предметів, які були закріплені за ним адміністратором. При виборі групи відображається відомість, де навпроти кожного ПІБ дитини доступні поля для введення балів та текстового поля для нотаток.

Проведено стрес-тестування механізму валідації вхідних даних (Zod-schema validation): при спробі внести від'ємне значення, бал, що перевищує встановлений максимум, або порожній коментар, інтерфейс блокує кнопку відправки форми та виводить повідомлення про помилку. При введенні коректних параметрів дані зберігаються в системі, а статус оцінювання змінюється на «Оцінено».

Екран відомості виставлення балів та коментарів екзаменатора у кабінеті вчителя зображено на рисунку 2.19.

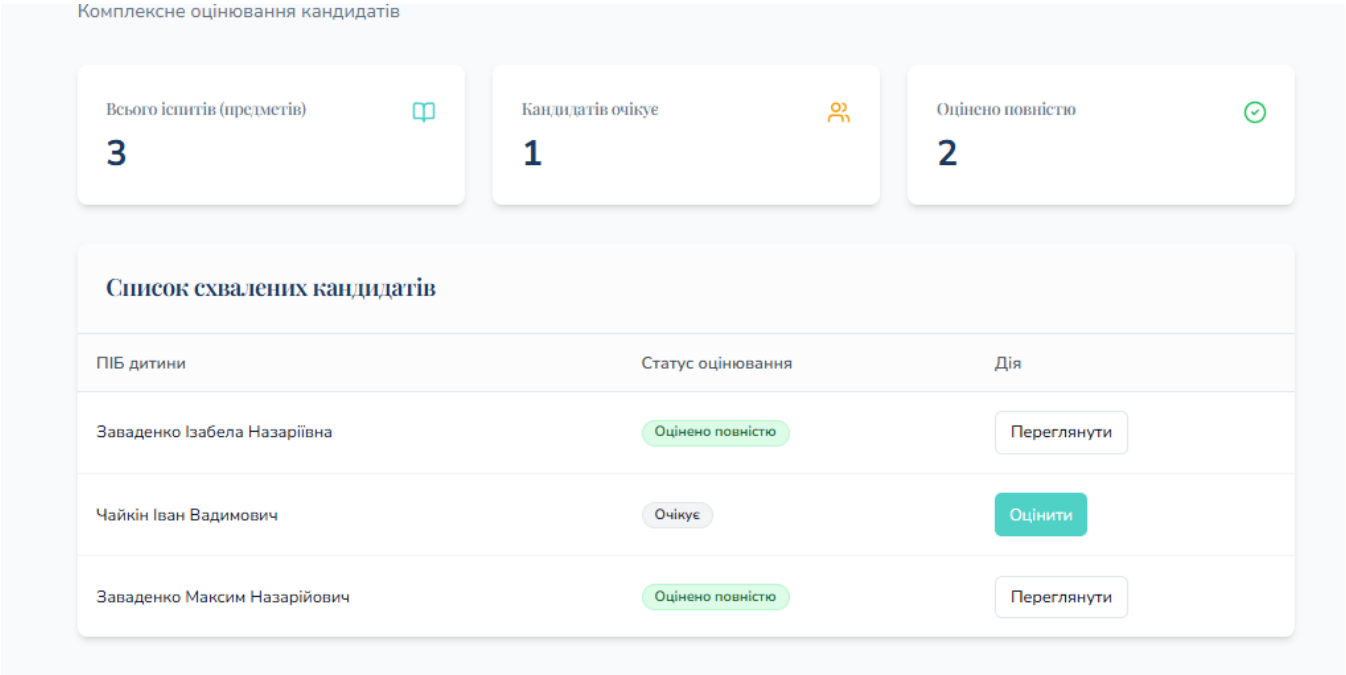

Рисунок 2.19 – Екран відомості виставлення балів та коментарів екзаменатора у кабінеті вчителя

Поданий список схвалених кандидатів дозволяє екзаменатору швидко відстежувати поточний статус кожної дитини та за допомогою відповідних кнопок дій оперативно переходити до процесу виставлення балів або перегляду залишених раніше результатів.

3 РЕЗУЛЬТАТИ РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ ІНФОРМАЦІЙНОГО СУПРОВОДУ ВСТУПНОЇ КАМПАНІЇ ДО ГІМНАЗІЇ

Загальний опис результатів розробки програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

У рамках кваліфікаційної роботи був розроблений вебзастосунок для автоматизації процесу вступу школярів до гімназії з функціоналом створення заяв, моніторингу статусу документів, виставлення екзаменаційних оцінок та затвердження списків на зарахування. Було реалізовано 4 основних модулі, які відповідають за ключовий функціонал системи. Модулі та їх функціонал показано в таблиці 3.1.

Таблиця 3.1 – Модулі та функціонал

Модулі	Функціонал
AuthModule (Модуль авторизації)	Цей модуль відповідає за автентифікацію користувачів, ведення сесій та чітке розмежування прав доступу між ролями: Батьки, Вчитель, Адміністратор, Директор.
ApplicationModule (Модуль заявок)	Цей модуль дозволяє батькам створювати нові електронні заяви на вступ, а адміністраторам (завучам) переглядати їх, фільтрувати та змінювати їхні статуси.
ExamModule (Модуль оцінювання)	Цей модуль відповідає за роботу вчителів: перегляд закріплених за ними кандидатів, виставлення балів та внесення текстових коментарів за результатами випробувань.
DashboardModule (Модуль аналітики)	Цей модуль збирає загальну статистику для директора, дозволяє переглядати рейтингові списки та масово затверджувати кандидатів на зарахування.

Відповідність результатів розробки вебзастосунку до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Порівняння, що вимагалося в технічному завданні та, що було реалізовано наведено у таблиці 3.2.

Таблиця 3.2 – Відповідність технічного завдання та реалізації

Вимога з ТЗ	Реалізовано
Розмежування прав доступу (Батьки, Вчитель, Адміністратор, Директор)	Так
Створення електронної заяви на вступ	Так
Можливість редагування статусу заявки адміністратором	Так
Внесення результатів іспитів та співбесід вчителем	Так
Формування аналітичних звітів для директора	Так
Затвердження списку зарахованих	Так

Кількісна та якісна характеристика до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Загальний обсяг розробленого проєкту налічує 251 файл. Загальна кількість вихідного коду становить понад 11200 рядків чистого коду. Основну частину проєкту складає клієнтський інтерфейс, тоді як серверна логіка на Node.js, налаштування бази даних SQLite та спільні інтерфейси займають близько 4000 рядків мовою TypeScript. Розмір проєкту (без урахування завантажених сторонніх бібліотек у директорії node_modules) становить близько 15 МБ.

Робота вебзастосунку перевірялась на персональному комп'ютері з операційною системою Windows 11 та характеристиками: 8 ГБ оперативної пам'яті, процесор на 4 ядра. Запуск локального сервера та збирання клієнтської частини інструментом Vite відбувається до 3 секунд. Завдяки оптимізації архітектури, використанню легкої реляційної СКБД SQLite та Drizzle ORM, швидкість обробки більшості HTTP-запитів (зміна статусу заяви, додавання оцінки) спрацьовує менше ніж за 0,1 секунди. На виконання всього циклу розробки, проєктування бази даних та налаштування архітектури було витрачено близько 120 годин.

В даний час вебзастосунок працює без збоїв та не допускає збереження некоректних даних. Для цього реалізовано надійну дворівневу перевірку вхідних даних за допомогою схем валідації Zod як на стороні клієнта, так і на сервері. Також передбачена висока масштабованість: за необхідності поточна модульна

структура дозволяє легко розширити функціонал (наприклад, додати модуль для завантаження сканкопій документів чи інтеграцію з платіжними системами).

Інтерфейс було створено максимально мінімалістичним і строгим, з використанням компонентів shadcn/ui. Він успішно пройшов внутрішнє тестування на реальних сценаріях використання без попередньої підготовки користувачів: всі учасники змогли інтуїтивно виконати поставлені завдання (реєстрація заяви, пошук кандидата, виставлення оцінки).

Результати тестування до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

В рамках кваліфікаційної роботи було проведено тестування методом «чорного ящика» для ключових сценаріїв використання за кожною з ролей. Перевірялися такі процеси: коректність реєстрації заяви батьками, безпомилкова зміна статусів у кабінеті адміністратора, валідація балів у кабінеті вчителя та успішне затвердження рейтингу директором. Всі сценарії пройшли успішно, вебзастосунок стабільно обробляє дані та цілком готовий до використання в умовах реальної вступної кампанії.

4 ОХОРОНА ПРАЦІ

Охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності[16].

У сфері інформаційних технологій, зокрема при розробці програмного забезпечення, робота зазвичай не є фізично важкою, але має високе інтелектуальне та психоемоційне навантаження, що потребує належної організації робочого середовища, режиму роботи та відпочинку. З огляду на те, що програмний продукт розробляється для подальшого впровадження та використання в освітній установі (гімназії), питання охорони праці розглядаються як з позиції розробника, так і з позиції кінцевого користувача системи (адміністратора, оператора).

4.1 Основні законодавчі і нормативні документи, які регламентують охорону праці в Україні

Охорона праці в Україні є важливою складовою державної політики, спрямованої на забезпечення безпечних та здорових умов праці для всіх працівників. Законодавча та нормативна база включає низку документів, які визначають права та обов'язки роботодавців і працівників, а також стандарти безпеки на робочих місцях[17]. Основними документами є:

- Конституція України: Стаття 43 гарантує кожному право на належні, безпечні та здорові умови праці, а також на соціальний захист у разі втрати працездатності.

- Кодекс законів про працю України (КЗпП): Розділ XI присвячений охороні праці, визначає обов'язки роботодавців щодо забезпечення безпеки праці та права працівників на охорону праці.
- Закон України «Про охорону праці»: Визначає правові, економічні та соціальні засади охорони праці в Україні, зобов'язує роботодавців створювати безпечні умови праці, проводити навчання та інструктажі.
- Закон України «Про загальнообов'язкове державне соціальне страхування»: Визначає порядок страхування працівників від нещасних випадків на виробництві та професійних захворювань.
- Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПіН 3.3.2.007-98): Регламентують гігієнічні вимоги до організації робочих місць користувачів комп'ютерів.
- Державні будівельні норми (ДБН В.2.5-28:2018 «Природне і штучне освітлення»): Встановлюють вимоги до освітленості робочих місць.
- Стандарти безпеки праці (ДСТУ): Національні стандарти, що регулюють безпеку праці, зокрема ДСТУ ISO 45001:2019 (Системи управління охороною здоров’я та безпекою праці).

4.2 Структура управління питаннями охорони праці в навчальному закладі

Оскільки розроблений вебзастосунок призначений для використання в освітній установі, управління питаннями охорони праці в гімназії є важливою складовою загальної системи управління. Структура управління включає кілька ключових елементів та рівнів відповідальності:

- Директор навчального закладу: несе загальну відповідальність за стан охорони праці, забезпечує фінансування заходів з охорони праці, приймає

рішення про призначення відповідальних осіб та контролює виконання законодавчих вимог.

- Служба охорони праці (або відповідальний інженер): відповідає за розробку, впровадження та контроль виконання заходів з охорони праці, проводить регулярні перевірки умов праці, організовує навчання та забезпечує ведення відповідної документації.
- Керівники структурних підрозділів (завідувачі кабінетів інформатики): безпосередньо контролюють дотримання норм безпеки на робочих місцях підлеглих та учнів.
- Співробітники та оператори ПЗ: зобов'язані дотримуватися вимог охорони праці, виконувати інструкції під час роботи та повідомляти керівництво про будь-які небезпечні умови праці або збої в роботі обладнання.

Важливою складовою є проведення інструктажів з охорони праці. Відповідно до Типового положення, розрізняють такі види інструктажів:

- Вступний – проводиться з усіма працівниками, які щойно приймаються на роботу, незалежно від їхньої освіти та стажу.
- Первинний – проводиться на робочому місці до початку роботи.
- Повторний – проводиться на робочому місці не рідше одного разу на півріччя для користувачів ПК.
- Позаплановий – проводиться при зміні технологічного процесу, заміні обладнання або після нещасних випадків.
- Цільовий – при виконанні разових робіт, не передбачених трудовою угодою.

У закладі обов'язково ведеться облік всіх заходів з охорони праці (журнали реєстрації інструктажів, акти перевірок) та проводяться регулярні навчання й тренування з евакуації.

4.3 Аналіз небезпечних і шкідливих виробничих факторів

При розробці програмного забезпечення та його подальшій експлуатації найбільш характерними шкідливими та небезпечними виробничими факторами є тривала робота за комп'ютером, підвищені вимоги до зору, гіподинамія та електричні ризики.

В таблиці 4.1 перераховані основні фактори згідно з ГОСТ 12.0.003-74.

Таблиця 4.1 – Шкідливі та небезпечні виробничі фактори

Найменування фактору	Можливі джерела його виникнення	Характер дії
Теплові небезпеки	Наявність речей, які можуть загорітися (електрична апаратура, кабелі)	Небезпечний та шкідливий
Небезпека випромінювання	Наявність дисплеїв комп'ютерів, які є джерелом електромагнітного випромінювання	Шкідливий
Небезпека ураження електричним струмом	Мережа живлення комп'ютерної та периферійної техніки, пошкоджена ізоляція	Небезпечний
Психофізіологічні небезпеки	Монотонність праці, перенапруженість зорових аналізаторів, розумова напруженість, тривала статичність пози (гіподинамія)	Шкідливий
Несприятлива освітленість	Недостатнє штучне і природне освітлення робочого місця, відблиски на екрані	Шкідливий
Порушення мікроклімату	Відсутність вентиляції, робота систем охолодження ПК (підвищення температури та зниження вологості)	Шкідливий

Для забезпечення безпечних і комфортних умов праці, а також збереження здоров'я користувача, необхідно передбачити комплекс організаційних, санітарно-гігієнічних та інженерно-технічних заходів. Ці рішення мають бути спрямовані на мінімізацію або повне усунення негативного впливу ідентифікованих факторів і розглядаються на наступних етапах проектування робочого місця.

4.4 Розробка заходів по зменшенню дії небезпечних і шкідливих факторів при розробці програмного забезпечення

Для створення безпечних і комфортних умов праці та мінімізації впливу зазначених факторів необхідно впроваджувати комплекс організаційних, санітарно-гігієнічних та технічних заходів.

Вимоги до організації робочого місця користувача ПК

Робоче місце розробника або оператора системи повинно відповідати ергономічним вимогам. Конструкція робочого столу має забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання. Висота робочої поверхні столу повинна регулюватися в межах 680-800 мм, а ширина і глибина становити не менше 800 мм.

Робоче крісло має бути підйомно-поворотним, з регулюванням висоти сидіння та кута нахилу спинки для забезпечення підтримки попереку. Екран монітора повинен знаходитися на відстані 600-700 мм від очей користувача, при цьому верхній край екрану слід розташовувати на рівні очей або трохи нижче.

Забезпечення нормативних параметрів мікроклімату

Мікроклімат виробничих приміщень суттєво впливає на працездатність та здоров'я. Пристрої обчислювальної техніки виділяють значну кількість тепла, що може призвести до зниження вологості та підвищення температури повітря. Згідно з санітарними нормами (ДСН 3.3.6.042-99), для категорії робіт 1а (легка фізична робота), оптимальні параметри мікроклімату становлять:

- у холодний період року: температура 22-24 °С, відносна вологість 40-60 %, швидкість руху повітря не більше 0,1 м/с;
- у теплий період року: температура 23-25 °С, відносна вологість 40-60 %, швидкість руху повітря не більше 0,1 м/с.
- Для підтримки цих параметрів приміщення обладнуються системами припливно-витяжної вентиляції та кондиціонування повітря. Також

обов'язковим є щоденне вологе прибирання та регулярне провітрювання приміщення.

Оптимізація виробничого освітлення

Правильно спроєктоване освітлення знижує втомлюваність очей. У приміщеннях для роботи з ПК використовується природне та штучне освітлення. Природне освітлення має здійснюватися через світлові отвори (вікна), орієнтовані переважно на північ або північний схід, щоб уникнути прямих сонячних променів.

Штучне освітлення проєктується як загальне рівномірне. Згідно з ДБН В.2.5-28:2018, рівень освітленості на робочому столі повинен становити не менше 300-500 лк. Для штучного освітлення рекомендується використовувати сучасні LED-світильники, які не створюють ефекту мерехтіння. Важливою умовою є відсутність відблисків на екрані монітора, для чого використовуються антиблікові покриття та жалюзі на вікнах.

4.5 Забезпечення електро- та пожежної безпеки на робочому місці

Сучасне робоче місце фахівця нерозривно пов'язане з використанням значної кількості електронно-обчислювальної техніки (ЕОТ), периферійних пристроїв та комунікаційного обладнання. Таке насичення робочого простору електричними приладами створює специфічні умови праці, що вимагають комплексного підходу до забезпечення електробезпеки та пожежної безпеки відповідно до чинних нормативно-правових актів з охорони праці.

Технічні та організаційні заходи з електробезпеки

Робота з комп'ютерною технікою пов'язана з потенційним ризиком ураження персоналу електричним струмом. За ступенем небезпеки ураження

електричним струмом офісні приміщення та комп'ютерні класи, як правило, відносяться до приміщень без підвищеної небезпеки. Це зумовлено наявністю сухого середовища, нормального температурного режиму, струмонепровідних підлог (дерево, лінолеум) та відсутністю струмопровідного пилу. Проте, щільне розміщення обладнання та розгалужена мережа живлення вимагають суворого дотримання низки технічних та організаційних правил.

Для створення безпечних умов експлуатації електроустановок на робочому місці передбачено наступні заходи:

- Система живлення та заземлення: Живлення комп'ютерної техніки повинно здійснюватися від однофазної мережі змінного струму напругою 220 В. Критично важливою вимогою є наявність надійного захисного заземлення. Підключення обладнання здійснюється виключно через триполюсні розетки європейського стандарту (з заземлюючим контактом), які підключені до контуру заземлення будівлі. Використання перехідників або розеток без заземлення категорично заборонено, оскільки це блокує скидання статичної електрики та струмів витоку.
- Захист від перепадів напруги: Сучасна обчислювальна техніка є чутливою до якості електроживлення. Тому підключення обладнання має здійснюватися через мережеві фільтри або джерела безперебійного живлення (ДБЖ / UPS). Вони не лише захищають апаратну частину від імпульсних стрибків напруги, але й дозволяють коректно завершити роботу та зберегти дані у разі раптового знеструмлення.
- Автоматичний захист мережі: Електричні щитки, що обслуговують робочі місця, повинні бути оснащені автоматичними вимикачами для захисту від струмів короткого замикання та перевантаження. Додатково рекомендується встановлення пристроїв захисного відключення (ПЗВ), які миттєво знеструмлюють лінію у разі виявлення струму витоку на корпус або тіло людини.
- Вимоги до експлуатації: Забороняється працювати з обладнанням, що має видимі механічні пошкодження корпусів, порушену ізоляцію кабелів живлення, оплавлені вилки або розетки. Кабелі живлення повинні бути

прокладені так, щоб виключити їх механічне пошкодження (бажано у спеціальних кабель-каналах або коробах) та унеможливити скручування.

– Організаційні заходи: Усі ремонтні роботи, модернізація чи технічне обслуговування ПК, пов'язане зі зняттям захисних кришок, повинні виконуватися лише після повного фізичного відключення обладнання від електромережі. Ці роботи мають право проводити виключно фахівці інженерно-технічного персоналу, які мають відповідну групу допуску з електробезпеки (не нижче III групи). Звичайні користувачі ПК проходять інструктаж і отримують I групу з електробезпеки.

Забезпечення пожежної безпеки приміщення

Специфіка приміщень, де розташована комп'ютерна техніка, полягає у високій концентрації горючих матеріалів: пластикові корпуси обладнання, ізоляція кабелів, паперові носії інформації, елементи меблів. Одночасно з цим у приміщенні присутні потенційні джерела запалювання: іскріння в розетках, коротке замикання, перегрів електронних компонентів або блоків живлення через накопичення пилу. Відповідно до норм вибухопожежної небезпеки, такі приміщення класифікуються як категорія «В» (пожежонебезпечні).

З метою попередження виникнення пожеж та мінімізації їх наслідків реалізується наступний комплекс протипожежних заходів:

– Системи виявлення пожежі: Приміщення обов'язково обладнуються автоматичними системами пожежної сигналізації. Найбільш ефективними для таких умов є димові та теплові сповіщувачі, оскільки на початкових стадіях загоряння електроніки та пластику (тління) виділяється значна кількість диму ще до появи відкритого полум'я.

– Засоби первинного пожежогасіння: Оскільки комп'ютерна техніка перебуває під напругою, гасіння пожежі водою або пінними вогнегасниками є категорично забороненим через ризик ураження струмом та незворотного пошкодження обладнання. Приміщення комплектуються вуглекислотними

вогнегасниками (типу ВВК). Вуглекислота не проводить електричний струм, безпечна для мікросхем і не залишає слідів після використання. Допускається використання порошкових вогнегасників (ВП), однак вогнегасний порошок може вивести електроніку з ладу. Норма оснащення складає не менше одного вогнегасника на 50 кв.м. площі приміщення, але не менше двох на окреме приміщення.

– Евакуаційні заходи: Шляхи евакуації з приміщення (коридори, сходи, двері) повинні утримуватися повністю вільними; їх захаращення меблями чи коробками є неприпустимим. У добре помітних місцях обов'язково розміщуються затверджені плани евакуації, а над виходами встановлюються світлові табло «Вихід», підключені до системи аварійного освітлення.

– Протипожежний режим: Важливим організаційним заходом є дотримання режиму робочого часу. Після закінчення робочого дня працівники зобов'язані повністю знеструмити все обладнання на своїх робочих місцях. Виняток становить лише обладнання, технологічний процес якого вимагає цілодобового функціонування (серверне обладнання, мережеві комутатори, системи безпеки) — воно повинно бути підключене до виділених ліній живлення з багаторівневим автоматичним захистом.

Комплексне та неухильне виконання зазначених вимог електро- та пожежної безпеки дозволяє звести до мінімуму ризики виникнення аварійних ситуацій, зберегти матеріальні цінності підприємства та, що найважливіше, гарантувати безпеку життя і здоров'я працівників.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було спроєктовано та розроблено повноцінний веб-застосунок для інформаційного супроводу вступної кампанії до гімназії. Створений програмний продукт дозволяє автоматизувати процес прийому заяв та зарахування учнів молодших класів до гімназії, що суттєво підвищує ефективність роботи приймальної комісії.

На початковому етапі був проведений ґрунтовний аналіз практичної задачі інженерії програмного забезпечення, що стосується управління вступною кампанією навчального закладу. Було проаналізовано існуючі програмні аналоги, зокрема державний портал ІСУО та комерційні CRM-системи, виявлено їхні недоліки у контексті конкурсного відбору та обґрунтовано необхідність створення спеціалізованого власного рішення. На основі цього дослідження була сформульована чітка постановка задачі та розроблено технічне завдання.

У процесі розробки вебзастосунку було виконано комплекс завдань: проведено збір та аналіз вимог, визначено чотири ключові групи користувачів із розмежованими правами доступу (батьки, вчитель, адміністратор, директор). Було обґрунтовано вибір багаторівневої клієнт-серверної архітектури та сучасного стеку технологій, що включає React, TypeScript, Node.js, Express, а також реляційну базу даних SQLite з використанням Drizzle ORM. На основі цих рішень розроблено проєкт бази даних, спроєктовано інтуїтивно зрозумілий інтерфейс та успішно реалізовано сам вебзастосунок.

У підсумку сформовано результати розробки, які підтверджують, що створений вебзастосунок повністю відповідає вимогам технічного завдання. Впровадження цього програмного продукту дозволить гімназії відмовитися від ручної паперової роботи, забезпечити прозорість конкурсного відбору та значно підвищити рівень довіри і комфорту для батьків майбутніх учнів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна реєстрація в заклади загальної середньої освіти [Електронний ресурс]. – Режим доступу: https://school.isuo.org/ (дата звернення: 12.03.2026).

2. CRM #1 в Україні [Електронний ресурс] // KeyCRM. – URL: https://ua.keycrm.app/ (дата звернення: 15.03.2026).

3. Соммервілл І. Інженерія програмного забезпечення. 10-те вид. Київ : Діалектика, 2019. – 736 с.

4. Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine [Електронний ресурс] // Офіційна документація Node.js. – URL: https://nodejs.org/ (дата звернення: 07.04.2026).

5. TanStack Query: Powerful asynchronous state management for TS/JS [Електронний ресурс] // Офіційна документація TanStack. – URL: https://tanstack.com/query/latest/ (дата звернення: 25.05.2026).

6. React Router: Declarative routing for React [Електронний ресурс] // Офіційна документація React Router. – URL: https://reactrouter.com/ (дата звернення: 26.05.2026).

7. Express: Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс] // Офіційна документація Express. – URL: https://expressjs.com/ (дата звернення: 27.05.2026).

8. Zod: TypeScript-first schema validation with static type inference [Електронний ресурс] // Офіційна документація Zod. – URL: https://zod.dev/ (дата звернення: 28.05.2026).

9. TypeScript: Typed JavaScript at Any Scale [Електронний ресурс] // Офіційна документація TypeScript. – URL: https://www.typescriptlang.org/ (дата звернення: 01.06.2026).

10. Drizzle ORM: Next generation TypeScript ORM [Електронний ресурс] // Офіційна документація Drizzle ORM. – URL: https://orm.drizzle.team/ (дата звернення: 30.05.2026).

11. Vite: Next Generation Frontend Tooling [Електронний ресурс] // Офіційна документація Vite. – URL: https://vitejs.dev/ (дата звернення: 02.04.2026).

12. Shadcn/ui: Beautifully designed components built with Radix UI and Tailwind CSS [Електронний ресурс]. – Режим доступу: https://ui.shadcn.com/ (дата звернення: 29.05.2026).

13. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML [Електронний ресурс] // Офіційна документація Tailwind CSS. – URL: https://tailwindcss.com/ (дата звернення: 31.05.2026).

14. React: The library for web and native user interfaces [Електронний ресурс] // Офіційна документація React. – URL: https://react.dev/ (дата звернення: 06.05.2026).

15. SQLite: A self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine [Електронний ресурс] // Офіційна документація SQLite. – URL: https://www.sqlite.org/ (дата звернення: 08.05.2026).

16. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. Поточна редакція [Електронний ресурс]. - Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 01.06.2026).

17. Правила охорони праці під час експлуатації електронно-обчислювальних машин : Наказ Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010 № 65 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0293-10 (дата звернення: 01.06.2026).

Додаток А - Технічне завдання на розробку програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

1 Вступ

1.1 Назва програмного забезпечення

Вебзастосунок для інформаційного супроводу вступної кампанії до гімназії.

1.2 Сфера застосування

Вебзастосунок застосовується в освітній сфері для автоматизації створення заяв, управління розкладом вступних випробувань та оцінювання кандидатів у закладах загальної середньої освіти (гімназіях).

2 Підстава для розробки програмного забезпечення

Розробка програмного забезпечення виконується на підставі завдання на кваліфікаційну роботу, виданого кафедрою ПЗАС Національного університету кораблебудування імені адмірала Макарова.

3 Призначення розробки

3.1 Функціональне призначення

Функціональним призначенням вебзастосунку є забезпечення можливості користувачів виконувати такі функції:

- реєстрація та авторизація користувачів за ролями;
- перегляд та обробка інформації про кандидатів і заявки (додавання, редагування, зміна статусів);
- управління розкладом вступних випробувань та внесення результатів (оцінок і коментарів);
- формування рейтингових списків кандидатів та затвердження результатів.

3.2 Експлуатаційне призначення

Вебзастосунок призначений для оптимізації процесів обліку кандидатів, управління розкладом випробувань та взаємодії з батьками. Це забезпечує ефективне управління ресурсами приймальної комісії, знижує можливість помилок та полегшує роботу співробітників гімназії.

4 Технічні вимоги до програми

4.1 Вимоги до функціональних характеристик

4.1.1 Вимоги до переліку виконуваних функцій

Додатком повинні користуватися чотири типи користувачів: Адміністратор (завуч), Директор, Вчитель (екзаменатор) та Батьки (заявники).

- Адміністратор: створення нових користувачів, обробка заявок, формування розкладу іспитів.
- Вчитель: перегляд закріплених кандидатів, виставлення балів та внесення текстових коментарів за результатами випробувань.
- Директор: перегляд рейтингових списків та масове затвердження кандидатів.
- Батьки: створення заявок, перегляд результатів та відстеження статусу.

4.1.2 Вимоги до організації вхідних та вихідних даних

Дані зберігаються у реляційній базі даних SQLite. Вхідні дані здійснюються за допомогою клавіатури, миші та випадаючих списків (форми на базі React Hook Form). Вихідні дані виводяться на екран у вигляді інтерактивних таблиць, карток профілів та спливаючих сповіщень у режимі реального часу.

4.2 Вимоги до надійності

Час відповіді сервера та обробки більшості запитів не повинен перевищувати 3 секунд. Стійке функціонування забезпечується регулярним резервним копіюванням локального файлу бази даних (sqlite.db).

4.3 Умови експлуатації

Умови стандартного офісного та домашнього середовища. Користувач повинен мати підключення до мережі Інтернет та базові навички роботи з браузером.

4.4 Вимоги до складу і параметрів технічних засобів

Для роботи клієнтської частини: ПК або мобільний пристрій, процесор з тактовою частотою не менше 2.0 ГГц, мінімум 4 ГБ оперативної пам'яті.

Для розгортання серверної частини: середовище Node.js, підтримка бази даних SQLite.

4.5 Вимоги до інформаційної та програмної сумісності

Кросплатформність клієнтської частини у сучасних веббраузерах (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge). Серверна частина повинна функціонувати в середовищі Node.js із використанням фреймворку Express.

4.6 Вимоги до маркування та упаковки

Вимоги до маркування та упаковки відсутні.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання відсутні.

4.8 Спеціальні вимоги

Спеціальні вимоги відсутні.

5 Вимоги до програмної документації

Склад програмної документації повинен включати:

- технічне завдання;
- код програми;
- керівництво користувача (файл ReadMe);
- опис програми;
- програму і методику випробувань.

6 Техніко-економічні показники

У кваліфікаційній роботі техніко-економічні показники не розраховуються.

7 Стадії та етапи розробки

Етапи розробки програмного забезпечення наведено в таблиці А.1.

Таблиця А.1 – Етапи розробки програмного забезпечення

Номер	Назва етапів роботи	Термін виконання
1.	Аналіз практичної задачі інженерії ПЗ	12.04.2026
2.	Аналіз вимог до ПЗ	22.04.2026
3.	Розробка архітектури ПЗ	04.05.2026
4.	Розробка проєкту ПЗ	14.05.2026
5.	Тестування ПЗ	16.05.2026

8 Порядок контролю і прийомки

На кожному етапі створення програмного забезпечення контролюють процеси аналізу та проєктування його складових з урахуванням вимог, викладених у технічному завданні. Контроль здійснюється керівником кваліфікаційної роботи. Приймання виконується за затвердженою методикою і програмою випробувань, а результати фіксуються під час демонстрації готового продукту.

Додаток Б – Код програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Додаток містить фрагменти коду розробленого програмного забезпечення.

Приведено інтерфейс та база даних.

Файл конфігурації схеми бази даних (shared/schema.ts)

```
import { sqliteTable, text, integer } from "drizzle-orm/sqlite-
core";
import { createInsertSchema } from "drizzle-zod";
import { z } from "zod";

export const users = sqliteTable("users", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  email: text("email").notNull().unique(),
  password: text("password").notNull(),
  surname: text("surname").notNull().default(""),
  firstName: text("first_name").notNull().default(""),
  patronymic: text("patronymic"),
  fullName: text("full_name").notNull(),
  phone: text("phone").notNull(),
  role: text("role", { enum: ["PARENT", "TEACHER", "ADMIN",
"DIRECTOR"] }).notNull().default("PARENT"),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const children = sqliteTable("children", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  studentId: text("student_id").notNull().unique(),
  firstName: text("first_name").notNull(),
  lastName: text("last_name").notNull(),
  patronymic: text("patronymic"),
  birthDate: integer("birth_date", { mode: "timestamp" }),
  gender: text("gender", { enum: ["male", "female"] }),
  address: text("address"),
  phone: text("phone"),
  averageGrade: text("average_grade"),
  school: text("school"),
  parentId: integer("parent_id").references(() =>
users.id).notNull(),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const subjects = sqliteTable("subjects", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  name: text("name").notNull(),
  isActive: integer("is_active").default(1).notNull(),
```

```
});

export const exams = sqliteTable("exams", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  name: text("name").notNull(),
  date: integer("date", { mode: "timestamp" }).notNull(),
  teacherName: text("teacher_name"),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const examResults = sqliteTable("exam_results", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  childId: integer("child_id").references(() =>
children.id).notNull(),
  examId: integer("exam_id").references(() => exams.id).notNull(),
  grade: integer("grade").notNull(),
  comment: text("comment"),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const applications = sqliteTable("applications", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  fullName: text("full_name").notNull(),
  birthDate: integer("birth_date", { mode: "timestamp"
}).notNull(),
  gender: text("gender", { enum: ["male", "female"] }).notNull(),
  address: text("address").notNull(),
  parentPhone: text("parent_phone").notNull(),
  averageGrade: text("average_grade").notNull(),
  school: text("school").notNull(),
  birthCertificate: text("birth_certificate"),
  parentPassport: text("parent_passport"),
  residenceDocument: text("residence_document"),
  schoolTransferCertificate: text("school_transfer_certificate"),
  gradeReport: text("grade_report"),
  status: text("status", { enum: ["PENDING", "APPROVED",
"REJECTED"] }).notNull().default("PENDING"),
  directorComment: text("director_comment"),
  parentId: integer("parent_id").references(() => users.id),
  childId: integer("child_id").references(() => children.id),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const grades = sqliteTable("grades", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  childId: integer("child_id").references(() =>
children.id).notNull(),
  subjectId: integer("subject_id").references(() =>
subjects.id).notNull(),
  grade: integer("grade").notNull(),
```

```
  comment: text("comment"),
  date: integer("date", { mode: "timestamp" }).notNull(),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const feedback = sqliteTable("feedback", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  name: text("name").notNull(),
  email: text("email").notNull(),
  message: text("message").notNull(),
  createdAt: integer("created_at", { mode: "timestamp"
}).$defaultFn(() => new Date()).notNull(),
});

export const studentGrades = sqliteTable("student_grades", {
  id: integer("id").primaryKey({ autoIncrement: true }),
  childId: integer("child_id").references(() =>
children.id).notNull().unique(),
  mathGrade: integer("math_grade"),
  ukrainianGrade: integer("ukrainian_grade"),
  englishGrade: integer("english_grade"),
  averageGrade: integer("average_grade").notNull().default(0),
});

export const insertUserSchema = createInsertSchema(users).omit({
id: true, createdAt: true });
export const insertChildSchema =
createInsertSchema(children).omit({ id: true, createdAt: true });
export const insertSubjectSchema =
createInsertSchema(subjects).omit({ id: true });
export const insertExamSchema = createInsertSchema(exams).omit({
id: true, createdAt: true });
export const insertExamResultSchema =
createInsertSchema(examResults).omit({ id: true, createdAt: true
});
export const insertApplicationSchema =
createInsertSchema(applications).omit({ id: true, createdAt: true,
status: true });
export const insertGradeSchema = createInsertSchema(grades).omit({
id: true, createdAt: true });
export const insertFeedbackSchema =
createInsertSchema(feedback).omit({ id: true, createdAt: true });
export const insertStudentGradeSchema =
createInsertSchema(studentGrades).omit({ id: true });

export type InsertUser = z.infer<typeof insertUserSchema>;
export type User = typeof users.$inferSelect;
export type InsertChild = z.infer<typeof insertChildSchema>;
export type Child = typeof children.$inferSelect;
export type InsertSubject = z.infer<typeof insertSubjectSchema>;
export type Subject = typeof subjects.$inferSelect;
export type InsertExam = z.infer<typeof insertExamSchema>;
```

```
export type Exam = typeof exams.$inferSelect;
export type InsertExamResult = z.infer<typeof
insertExamResultSchema>;
export type ExamResult = typeof examResults.$inferSelect;
export type InsertApplication = z.infer<typeof
insertApplicationSchema>;
export type Application = typeof applications.$inferSelect;
export type InsertGrade = z.infer<typeof insertGradeSchema>;
export type Grade = typeof grades.$inferSelect;
export type InsertFeedback = z.infer<typeof insertFeedbackSchema>;
export type Feedback = typeof feedback.$inferSelect;
export type InsertStudentGrade = z.infer<typeof
insertStudentGradeSchema>;
export type StudentGrade = typeof studentGrades.$inferSelect;
```

Файл реалізації контролера обробки заявок на вступ (server/controllers/applications.ts)

```
import { Application, ApplicationStatus } from "@/types";

const API_BASE = '/api';

export const saveApplication = async (application:
Omit<Application, 'id' | 'status' | 'createdAt'>, parentId?:
string): Promise<Application> => {
  const response = await fetch(`${API_BASE}/applications`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      ...application,
      birthDate: new Date(application.birthDate),
      parentId: parentId ? parseInt(parentId) : null
    })
  });
  const data = await response.json();
  return transformApplication(data);
};

export const getApplications = async (): Promise<Application[]> =>
{
  const response = await fetch(`${API_BASE}/applications`);
  const data = await response.json();
  return data.map(transformApplication);
};

export const getApplicationsByChildId = async (childId: string):
Promise<Application[]> => {
  const response = await fetch(`${API_BASE}/applications`);
  const data = await response.json();
  return data.filter((a: any) => String(a.childId) ===
childId).map(transformApplication);
};
```

```
export const getApplicationsByParentId = async (parentId: string):
Promise<Application[]> => {
  const response = await
fetch(`${API_BASE}/applications/parent/${parentId}`);
  const data = await response.json();
  return data.map(transformApplication);
};

export const updateApplicationStatus = async (applicationId:
string, status: ApplicationStatus, childId?: string):
Promise<boolean> => {
  const body: any = { status };
  if (childId) body.childId = parseInt(childId);

  const response = await
fetch(`${API_BASE}/applications/${applicationId}`, {
    method: 'PATCH',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(body)
  });
  return response.ok;
};

export const updateApplication = async (id: string, updates:
Partial<Application>): Promise<Application | null> => {
  const response = await fetch(`${API_BASE}/applications/${id}`, {
    method: 'PATCH',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(updates)
  });
  if (!response.ok) return null;
  return transformApplication(await response.json());
};

export const deleteApplication = async (id: string):
Promise<boolean> => {
  const response = await fetch(`${API_BASE}/applications/${id}`, {
method: 'DELETE' });
  return response.ok;
};

function transformApplication(data: any): Application {
  return {
    id: String(data.id),
    fullName: data.fullName,
    birthDate: data.birthDate,
    gender: data.gender,
    address: data.address,
    parentPhone: data.parentPhone,
    averageGrade: data.averageGrade,
    school: data.school,
    status: data.status as "PENDING" | "APPROVED" | "REJECTED",
```

```
    createdAt: data.createdAt,
    parentId: data.parentId ? String(data.parentId) : undefined,
    childId: data.childId ? String(data.childId) : undefined,
    directorComment: data.directorComment,
    birthCertificate: data.birthCertificate,
    parentPassport: data.parentPassport,
    residenceDocument: data.residenceDocument,
    schoolTransferCertificate: data.schoolTransferCertificate,
    gradeReport: data.gradeReport,
  };
}
```

Головний компонент клієнтської маршрутизації (client/src/App.tsx)

```
import { Toaster } from "@/components/ui/toaster";
import { Toaster as Sonner } from "@/components/ui/sonner";
import { TooltipProvider } from "@/components/ui/tooltip";
import { QueryClient, QueryClientProvider } from "@tanstack/react-
query";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Index from "./pages/Index";
import About from "./pages/About";
import Contact from "./pages/Contact";
import Application from "./pages/Application";
import Login from "./pages/Login";
import Register from "./pages/Register";
import NotFound from "./pages/NotFound";
import { LanguageProvider } from "@/context/LanguageContext";
import { AuthProvider } from "@/context/AuthContext";
import Dashboard from "./pages/Dashboard";
import AdminPanel from "./pages/AdminPanel";
import Exams from "./pages/Exams";
import Director from "./pages/Director";
import Teacher from "./pages/Teacher";
import ProtectedRoute from "./components/ProtectedRoute";
import { UserRole } from "./types";

const queryClient = new QueryClient();

const App = () => (
  <QueryClientProvider client={queryClient}>
    <TooltipProvider>
      <AuthProvider>
        <LanguageProvider>
          <Toaster />
          <Sonner />
          <BrowserRouter>
            <Routes>
              <Route path="/" element={<Index />} />
              <Route path="/about" element={<About />} />
              <Route path="/contact" element={<Contact />} />
              <Route path="/application" element={<Application />}
/>
```

```
              <Route path="/login" element={<Login />} />
              <Route path="/register" element={<Register />} />
              <Route path="/dashboard" element={
                <ProtectedRoute allowedRoles={[UserRole.PARENT,
UserRole.TEACHER, UserRole.ADMIN, UserRole.DIRECTOR]}>
                  <Dashboard />
                </ProtectedRoute>
              } />
              <Route path="/admin" element={
                <ProtectedRoute allowedRoles={[UserRole.ADMIN,
UserRole.DIRECTOR]}>
                  <AdminPanel />
                </ProtectedRoute>
              } />
              <Route path="/teacher" element={
                <ProtectedRoute allowedRoles={[UserRole.TEACHER]}>
                  <Teacher />
                </ProtectedRoute>
              } />
              <Route path="/director" element={
                <ProtectedRoute
allowedRoles={[UserRole.DIRECTOR]}>
                  <Director />
                </ProtectedRoute>
              } />
              <Route path="/exams" element={
                <ProtectedRoute allowedRoles={[UserRole.PARENT,
UserRole.TEACHER, UserRole.ADMIN, UserRole.DIRECTOR]}>
                  <Exams />
                </ProtectedRoute>
              } />
              <Route path="*" element={<NotFound />} />
            </Routes>
          </BrowserRouter>
        </LanguageProvider>
      </AuthProvider>
    </TooltipProvider>
  </QueryClientProvider>
);

export default App;
```

Компонент панелі адміністратора для зміни статусів (client/src/pages/AdminDashboard.tsx)

```
import { useState, useEffect } from "react";
import { motion } from "framer-motion";
import Navbar from "@/components/Navbar";
import Footer from "@/components/Footer";
import { useAuth } from "@/context/AuthContext";
import {
  getApplications,
```

```
  getExams,
  getAllUsers,
  getSubjects
} from "@/utils/dataService";
import { Application } from "@/types";
import { useNavigate } from "react-router-dom";
import { useIsMobile } from "@/hooks/use-mobile";
import { toast } from "sonner";

// Import refactored components
import StatisticsCards from "@/components/admin/StatisticsCards";
import ApplicationsSection from
"@/components/admin/ApplicationsSection";
import SubjectsSection from "@/components/admin/SubjectsSection";
import ExamsSection from "@/components/admin/ExamsSection";
import ActivityTable from "@/components/admin/ActivityTable";
import UserTable from "@/components/director/UserTable";
import { User } from "@/types";

interface EnhancedExam {
  id: string;
  name: string;
  date: string;
  createdAt: string;
  teacherName?: string;
  room?: string;
  time?: string;
}

const AdminPanel = () => {
  const { user } = useAuth();
  const navigate = useNavigate();
  const isMobile = useIsMobile();
  const [applications, setApplications] =
useState<Application[]>([]);
  const [exams, setExams] = useState<EnhancedExam[]>([]);
  const [users, setUsers] = useState<User[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      try {
        // Fetch applications
        const allApplications = await getApplications();
        setApplications(allApplications);

        // Fetch exams and enhance them with demo data
        const examsData = await getExams();
        const allExams = examsData.map(exam => ({
          ...exam,
          room: `Аудиторія ${Math.floor(Math.random() * 10) + 1}`,
          time: `${Math.floor(Math.random() * 7) + 9}:00`
```

```
        }));
        setExams(allExams);

        // Fetch users
        const allUsers = await getAllUsers();
        setUsers(allUsers);
      } catch (error) {
        console.error("Error fetching data:", error);
        toast.error("Помилка при завантаженні даних");
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  const handleExamsUpdate = (updatedExams: EnhancedExam[]) => {
    setExams(updatedExams);
  };

  if (loading) {
    return (
      <>
        <Navbar />
        <div className="container mx-auto px-4 py-16 flex justify-
center items-center min-h-screen">
          <div className="animate-spin rounded-full h-16 w-16
border-t-2 border-b-2 border-primary"></div>
        </div>
        <Footer />
      </>
    );
  }

  return (
    <>
      <Navbar />

      <div className="container mx-auto px-4 py-16">
        <motion.div
          className="max-w-6xl mx-auto"
          initial={{ opacity: 0 }}
          animate={{ opacity: 1 }}
          transition={{ duration: 0.5 }}
        >
          <div className="bg-white rounded-xl shadow-lg p-8 mb-8">
            <div className="flex flex-col sm:flex-row justify-
between items-start sm:items-center mb-8 gap-4">
              <h1 className="text-3xl font-bold text-[#1a365d]">
                Панель управління Завуча
              </h1>
              <div className="text-right">
```

```
                <p className="text-lg font-medium">
                  Вітаємо, {user?.fullName || "Завуч гімназії"}!
                </p>
                <p className="text-gray-500">{user?.email}</p>
              </div>
            </div>

            {/* Statistics Cards */}
            <StatisticsCards applications={applications}
exams={exams} />

            {/* Users Section */}
            <div className="mb-8">
              <h2 className="text-xl font-bold text-[#1a365d] mb-
4">Користувачі</h2>
              <UserTable users={users} />
            </div>

            {/* Applications Section */}
            <ApplicationsSection applications={applications} />

            {/* Exams Section */}
            <ExamsSection exams={exams}
onExamsUpdated={handleExamsUpdate} />

            {/* Subjects Section */}
            <SubjectsSection />

            {/* Activity Table */}
            <ActivityTable />
          </div>
        </motion.div>
      </div>

      <Footer />
    </>
  );
};

export default AdminPanel;
```

Додаток В - Опис програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

1 Загальні відомості

1.1 Повне найменування програми

Повне найменування: Програмне забезпечення для інформаційного супроводу вступної кампанії до гімназії.

1.2 Програмне забезпечення, необхідне для функціонування

Для функціонування серверної частини вебзастосунку необхідне середовище виконання Node.js. Клієнтська частина працює у будь-якому сучасному веббраузері (Google Chrome, Microsoft Edge, Mozilla Firefox, Safari). База даних використовує вбудовану рушійну систему SQLite, що не потребує встановлення окремого сервера СУБД.

1.3 Мови програмування та технології

Програмний продукт розроблено з використанням мови програмування TypeScript (для строгої типізації) та бази JavaScript. Для створення користувацького інтерфейсу використано бібліотеку React 18, стилізація виконана за допомогою фреймворку Tailwind CSS та компонентів shadcn/ui. Серверна частина реалізована на базі Express, а взаємодія з базою даних здійснюється за допомогою Drizzle ORM.

2 Функціональне призначення

Вебзастосунок призначений для комплексної автоматизації вступної кампанії до гімназії. До його основних функцій належать:

- реєстрація та авторизація користувачів із визначенням рівня доступу (Батьки, Вчитель, Адміністратор, Директор);
- створення, редагування та відстеження статусу заяв на вступ дитини до закладу;

– обробка заяв адміністраторами, зміна їхніх статусів та формування екзаменаційних груп;

– виставлення оцінок та фіксація текстових коментарів вчителями за підсумками вступних випробувань;

– агрегація статистичних даних, формування рейтингових списків та масове затвердження кандидатів на зарахування директором.

3 Опис логічної структури

Програма побудована за клієнт-серверною архітектурою.

Клієнтська частина (Frontend) включає модулі:

– модуль автентифікації та управління станом користувача (Context API);

– модуль маршрутизації (React Router) для захисту приватних сторінок;

– модуль візуалізації даних, що відповідає за відмальовування таблиць, форм, діалогових вікон та дашбордів.

– Серверна частина (Backend) включає:

– контролери (Controllers), які обробляють запити від клієнта (створення заявки, зміна статусу, додавання оцінки);

– модуль валідації (Zod), що перевіряє коректність отриманих даних;

– шар доступу до даних (Drizzle ORM), який конвертує об'єктні запити у SQL-інструкції для SQLite.

База даних містить реляційні таблиці: Користувачі (users), Заявки (applications), Кандидати (candidates), Іспити (exams) та Результати (results).

4 Використовувані технічні засоби

Для розгортання та коректної роботи програмного забезпечення необхідні такі апаратні ресурси:

Серверна частина:

– процесор з тактовою частотою від 2.0 ГГц;

- обсяг оперативної пам'яті: не менше 2 ГБ;
- вільне місце на жорсткому диску: від 500 МБ.
- Клієнтська частина:
- персональний комп'ютер, ноутбук або мобільний пристрій;
- обсяг оперативної пам'яті: від 2 ГБ;
- підключення до мережі Інтернет або локальної мережі закладу.

5 Виклик і завантаження

Запуск вебзастосунку в режимі експлуатації (або розробки) здійснюється через термінал командного рядка:

- Завантаження залежностей: виконання команди npm install у кореневій директорії проєкту.
- Створення та ініціалізація бази даних: виконання команди генерації міграцій та схеми Drizzle.
- Запуск застосунку: виконання команди npm run dev для локальної розробки або npm run start (після білду) для продуктового середовища.
- Доступ до графічного інтерфейсу здійснюється шляхом введення локальної (наприклад, http://localhost:5173) або доменної адреси у адресному рядку браузера.

6 Вхідні дані

До вхідних даних вебзастосунку належать:

- ідентифікаційні дані користувачів (логін, пароль, ПІБ, номер телефону, електронна пошта);
- персональні дані кандидатів (ПІБ дитини, дата народження, адреса проживання);
- екзаменаційні дані (числове значення отриманого бала, текстовий висновок-коментар екзаменатора);
- параметри фільтрації та пошуку в таблицях (ключові слова, обрані статуси зі списку).

7 Вихідні дані

До вихідних даних вебзастосунку належать:

- візуальні таблиці та переліки заявок з поточними статусами;
- графічні дашборди зі статистичними показниками вступної кампанії (загальна кількість заявок, відсоток прийнятих/відхилених);
- рейтингові списки абітурієнтів, згруповані за загальною кількістю балів;
- системні сповіщення (успішне збереження, помилка валідації, повідомлення про відсутність прав доступу).

Додаток Г - Інструкція програмісту, оператору й користувачеві до програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Призначення та умови застосування

Програма призначена для автоматизації прийому заяв, обліку результатів іспитів та формування списків зарахованих учнів. Для подальшої підтримки та модифікації вихідного коду програмісту необхідне встановлене середовище виконання Node.js (версії 18 або вище), пакетний менеджер npm та сучасний редактор коду (рекомендується Visual Studio Code).

Характеристика програми та архітектура

Вебзастосунок реалізовано як повноцінний Full-Stack проєкт. Клієнтська частина (Frontend) написана мовою TypeScript з використанням бібліотеки React 18, інструменту збирання Vite, фреймворку Tailwind CSS та компонентів shadcn/ui. Серверна частина (Backend) побудована на базі Express.js. Для роботи з даними використовується легка реляційна СУБД SQLite у зв'язці з інструментом Drizzle ORM.

Структура проєкту Робоча директорія містить такі основні папки та файли:

- client/ (або src/) – містить логіку інтерфейсу, компоненти React, маршрутизацію (React Router) та файли локалізації (i18n).
- server/ – логіка бекенду, налаштування Express-сервера, REST API маршрути та контролери.
- shared/ – спільна частина коду, яка містить схему бази даних (schema.ts) та типи валідації Zod, що використовуються одночасно клієнтом і сервером.
- sqlite.db – файл локальної бази даних (генерується під час запуску).

Налаштування середовища та запуск для розробки Для розгортання проєкту на робочій станції програміста необхідно:

1. Клонувати або скопіювати вихідний код у робочу папку.

2. Відкрити термінал у корені проєкту та виконати команду npm install для завантаження всіх залежностей.

3. Виконати синхронізацію схеми бази даних за допомогою Drizzle ORM (наприклад, npm run db:push або відповідну команду скрипта).

4. Запустити сервер розробки командою npm run dev. Проєкт буде доступний за локальною адресою (наприклад, http://localhost:5173).

Інструкція оператору (системному адміністратору)

Вимоги до технічних засобів Для розгортання вебзастосунку в локальній мережі гімназії або на хостингу необхідний сервер або ПК із двоядерним процесором, обсягом оперативної пам'яті від 2 ГБ, та мінімум 500 МБ вільного простору на жорсткому диску. Операційна система: Windows 10/11, Linux (Ubuntu/Debian) або macOS.

Запуск у робочому режимі (Production) Для переведення вебзастосунку в продуктовий режим необхідно:

- Згенерувати оптимізовані статичні файли клієнта та закомпілювати сервер за допомогою команди npm run build.
- Запустити збудований застосунок командою npm run start (або за допомогою процесного менеджера, наприклад PM2, для забезпечення безперервної роботи).

Обслуговування бази даних та безпека Оскільки програмне забезпечення використовує вбудовану базу даних SQLite, вся інформація фізично зберігається в одному файлі sqlite.db у корені серверної папки. Задачею оператора є налаштування регулярного резервного копіювання (бекапу) цього файлу на зовнішній носій або у хмарне сховище щонайменше раз на добу під час активної вступної кампанії.

Інструкція користувачеві

Початок роботи та авторизація Для початку роботи необхідно відкрити підтримуваний веббраузер (Google Chrome, Microsoft Edge тощо) та перейти за

посиланням, наданим адміністрацією гімназії. На стартовій сторінці відобразиться форма автентифікації.

Форма авторизації користувача зображено на рисунку Г.1.

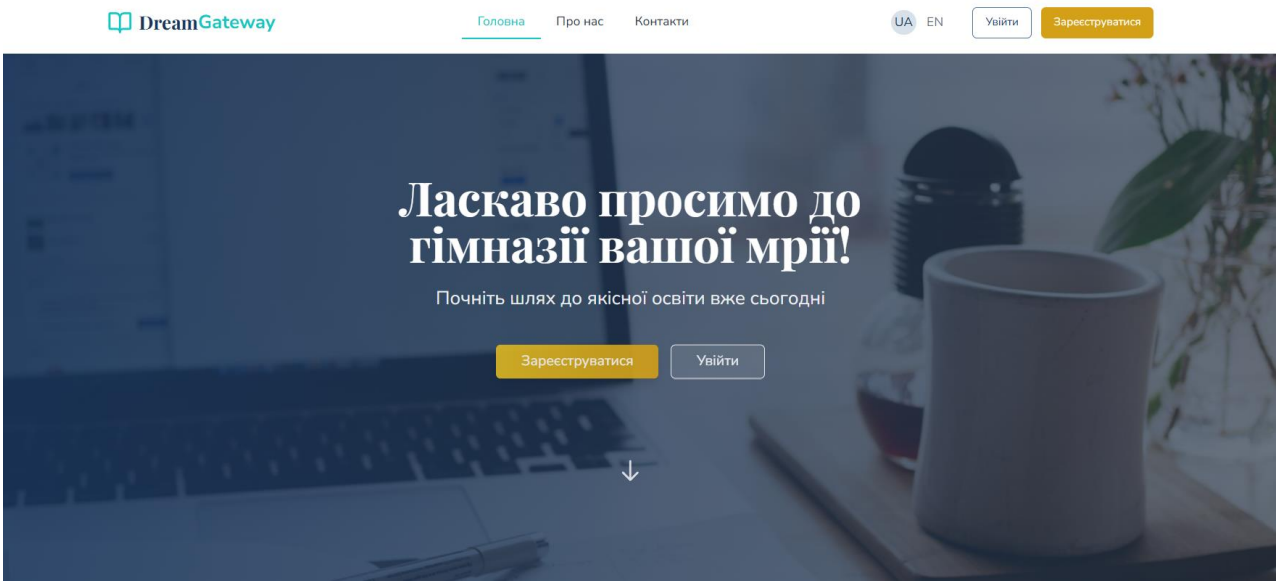

Рисунок Г.1 – Форма авторизації користувача

Користувачу необхідно ввести свою електронну пошту та пароль. Програмне забезпечення автоматично визначить роль користувача (Батьки, Вчитель, Адміністратор або Директор) і перенаправить у відповідний особистий кабінет. Для зручності підтримується перемикання мови інтерфейсу (українська/англійська).

Робота в кабінеті ролі «Батьки» (Заявники) Цей кабінет призначений для створення електронної заяви.

– Щоб подати заяву, перейдіть до особистого кабінету та натисніть «Подати заявку», після цього завантажте всі потібні документи і натисніть «Подати заявку».

– На головному екрані ви зможете відслідковувати поточний статус поданої заяви (наприклад, «Відхилено», «В очікуванні», «Зараховано») та переглядати результати після перевірки робіт вчителями.

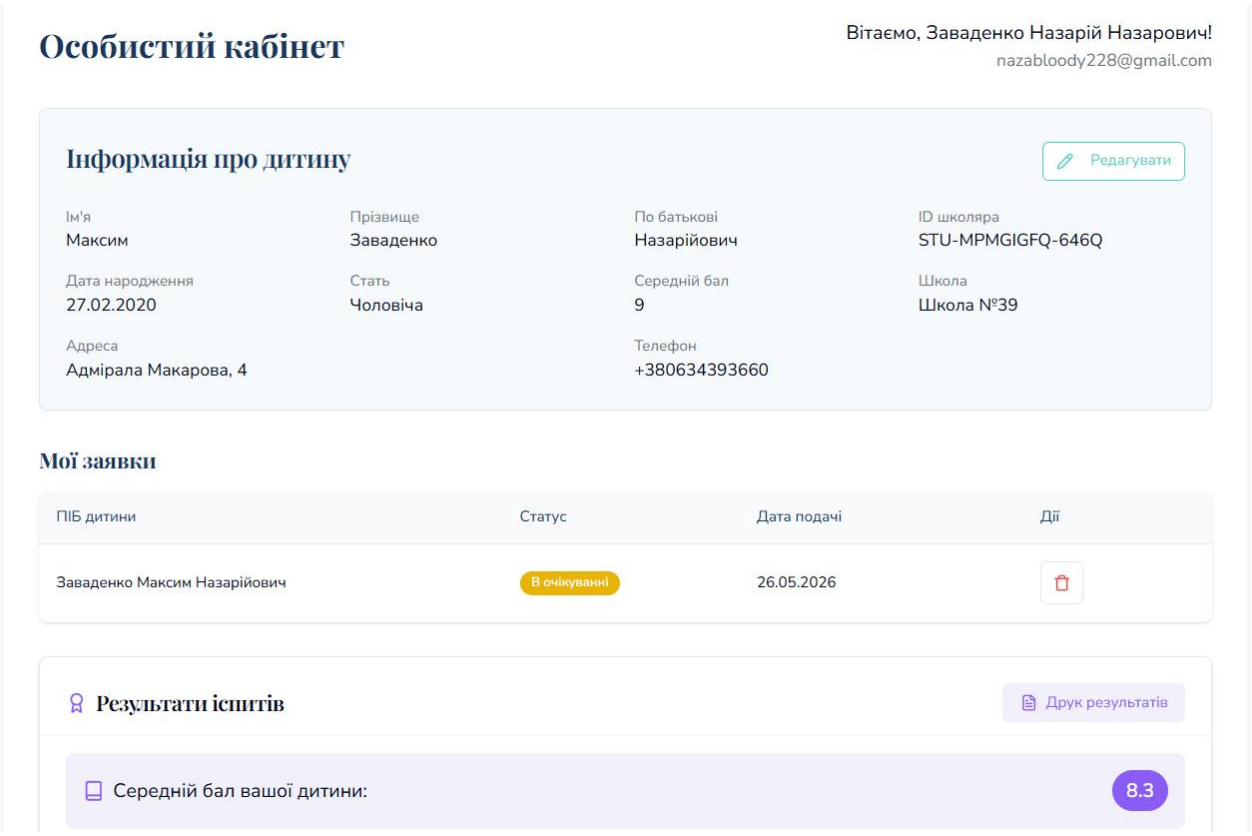

Рисунок Г.2 – Кабінет батьків та подача заяви

Робота в кабінеті ролі «Вчитель» (Екзаменатор) Кабінет вчителя має спрощений інтерфейс, призначений виключно для оцінювання.

– На у особистому кабінеті відобразиться список кандидатів, для оцінювання.

– Натисніть кнопку виставлення оцінки навпроти прізвища кандидата. У формі, що з'явиться, введіть кількість балів та не обов'язковий текстовий коментар (висновок).

– Збережіть результат. Після цього статус оцінювання зміниться, а бали будуть прикріпленні до профілю школяра.

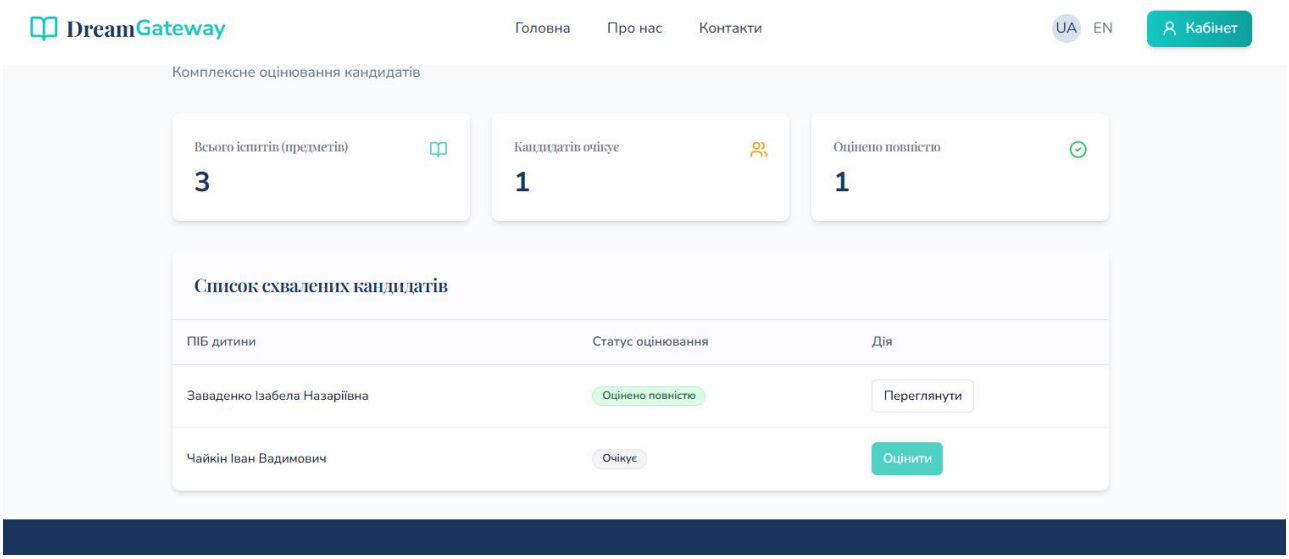

Рисунок Г.3 – Інтерфейс виставлення оцінок вчителем

Робота в кабінеті ролі «Адміністратор» (Завуч) Адміністратор здійснює повний контроль за документообігом:

– У головній таблиці зібрані всі заяви. Використовуйте рядок пошуку для знаходження конкретного учня або фільтр для відображення заяв за певним статусом.

– Для зміни статусу заяви натисніть на відповідний бейдж (індикатор статусу) у таблиці та оберіть (наприклад, з «Схвалити» на «Відхилено» чи «В очікуванні»). Зміни збережуться автоматично.

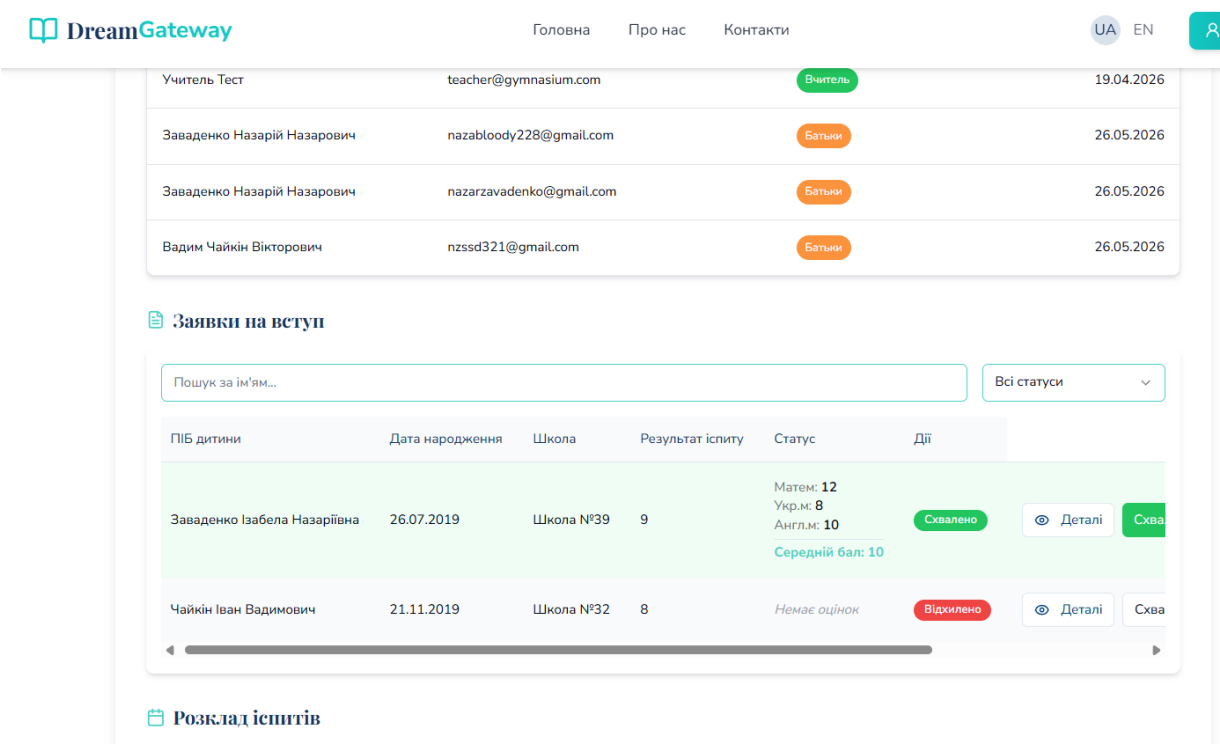

Рисунок Г.4 – Панель адміністратора та таблиця управління заявками

Робота в кабінеті ролі «Директор» Кабінет керівника призначений для аналізу та прийняття кінцевих рішень:

– Після входу відкривається аналітична панель (дашборд), де наведено загальну статистику: кількість поданих заяв, відсоток успішності, кількість перевірених робіт.

– У розділі затвердження директор бачить зведений рейтинговий список. Обравши кандидатів з найвищими балами, необхідно натиснути кнопку затвердження, що автоматично переведе їхні заяви у фінальний статус «Зараховано».

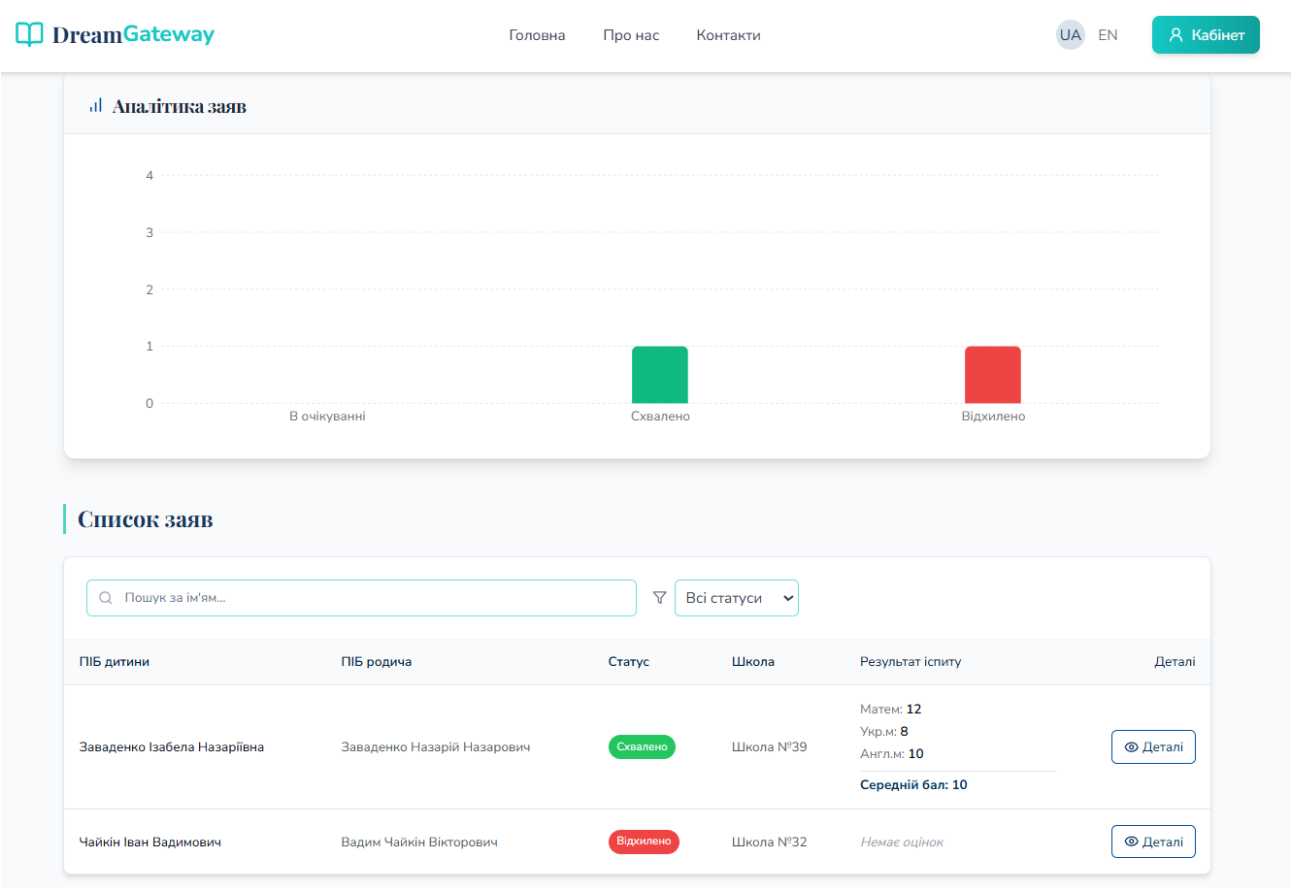

Рисунок Г.5 – Аналітична панель та розділ затвердження у кабінеті директора

Додаток Д - Програма та методика випробувань програмного забезпечення для інформаційного супроводу вступної кампанії до гімназії

Об'єкт випробувань

- Найменування програми: Вебзастосунок для інформаційного супроводу вступної кампанії до гімназії.
- Область застосування: Автоматизація створення заяв, моніторингу статусу документів, виставлення екзаменаційних оцінок, управління розкладом та затвердження списків на зарахування.
- Позначення: DreamGateway-1.0

Мета випробувань

Метою випробувань є перевірка відповідності функціональних можливостей розробленого вебзастосунку вимогам технічного завдання.

Вимоги до програми

Під час випробувань перевіряються наступні вимоги:

- Коректна реєстрація та авторизація користувачів із розмежуванням ролей (Батьки, Вчитель, Адміністратор, Директор).
- Створення, редагування заяв та зміна їх статусів адміністратором.
- Внесення результатів іспитів (балів та коментарів) вчителями.
- Відображення актуальної статистики та затвердження фінального списку директором.
- Захист від введення некоректних даних (валідація полів форм).

Вимоги до програмної документації

- Склад документації:

1) Технічне завдання;
2) Вихідний текст програмних модулів;

3) Опис програмного забезпечення;

4) Інструкція користувача;

5) Програма та методика випробувань.

Засоби і порядок випробувань

– Технічні засоби:

1) ПК з ОС Windows 10/11;

2) Оперативна пам'ять: від 4 ГБ;

3) Процесор: від 2 ядер.

– Програмні засоби:

1) Node.js (для запуску серверної частини);

2) Браузер (Google Chrome/Microsoft Edge).

– Порядок випробувань:

1) Запуск програми на тестовому ПК;

2) Перевірка функцій подачі заяви, зміни статусів, виставлення оцінок та генерації списків;

3) Порівняння результатів із очікуваними згідно з ТЗ.

Методи випробувань

Методом випробувань розробленого вебзастосунку є поетапна перевірка кожної функції (метод «чорної скриньки») під час роботи з ним в інтерфейсі браузера.

Перелік випробувань наведений у таблиці Д.1.

Таблиця Д.1 – Перелік випробувань

Назва випробування	Вхідні дані	Очікуваний результат	Реальний результат
1	2	3	4
Авторизація користувача	Введення email (admin@gymnasium.com) та пароля	Успішний вхід до кабінету адміністратора (завуча)	Успішно

Кінець таблиці Д.1

1	2	3	4
Створення заяви (роль Батьки)	ПІБ дитини: "Заваденко Ізабела Назаріївна", дата народження, контакти батьків	Успішне створення заяви зі статусом «В очікуванні»	Успішно
Зміна статусу заяви (роль Адміністратор)	Вибір статусу «Схвалено» зі списку для створеної заяви	Оновлення статусу заяви в базі даних та відображення в таблиці	Успішно
Виставлення оцінки (роль Вчитель)	Вибір кандидата, введення балу (наприклад, 10), додавання коментаря	Оцінку успішно збережено, статус кандидата змінено на «Оцінено повністю»	Успішно
Затвердження списків (роль Директор)	Натискання кнопки масового затвердження для обраних кандидатів	Статус обраних кандидатів змінюється на «Зараховано»	Успішно
Перевірка валідації	Введення від'ємного балу при оцінюванні	Виведення повідомлення про помилку, блокування збереження даних	Успішно