

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Система виконання завдань та тестів для шкільного
закладу освіти»**

Виконав студент групи ІПЗ-20-3
НОВІЧКОВ Єгор Максимович

Керівник роботи:
КОНЦИДАЙЛО Андрій Михайлович

Рецензент:
ГОДЛЕВСЬКИЙ Юрій Олександрович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **НОВІЧКОВ Єгор Максимович**

Керівник роботи: **КОНЦИДАЙЛО Андрій Михайлович**

Тема роботи: **«Система виконання завдань та тестів для шкільного закладу освіти»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **використання сучасних технологій для проектування, розробки та реалізації системи виконання завдань та тестів для шкільного закладу освіти.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концидайло А.М.	27.02.2024р.	27.02.2024р.
2	Концидайло А.М.	27.02.2024р.	27.02.2024р.
3	Концидайло А.М.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з системи виконання завдань та тестів для шкільного закладу освіти та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 84 сторінках друкованого тексту, має 15 сторінок додатків. Список використаних джерел містить 34 найменування і займає 3 сторінки. В роботі наведено 64 рисунка та 12 таблиць. Загальний обсяг роботи – 99 сторінок.

Метою роботи є розробка вебдодатку для виконання завдань та тестів, який призначений підвищити ефективність процесу обміну інформацією між учасниками шкільного процесу навчання.

У роботі визначено основні завдання на розробку системи, проведено аналіз вимог та функціональності додатку, також було проведено аналіз аналогів створеного програмного продукту. Було обрано архітектуру системи, здійснено проектування бази даних та обрано набір сучасних технологій розробки, використання яких забезпечує високу продуктивність додатку. Описано програмну реалізацію системи.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, НАВЧАЛЬНИЙ ПРОЦЕС, ШКОЛА, NODE.JS, REACT.JS, NEXT.JS.

					ІПЗ.КР.Б-121-24-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Система виконання завдань та тестів для шкільного закладу освіти	Лім.	Арк.	Аркушів
Розроб.		Новічков Є.М.					3	
Керівник		Концидайло А.М.				Житомирська політехніка, група ІПЗ-20-3		
Керівник								
Н. контр.		Тичина О.П.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

A bachelor's degree thesis consists of tasks and tests completion system for a school and an explanatory note. The explanatory note to the final thesis contains the text part, which is laid out on 84 pages of printed text, and has 15 pages of appendices. The list of references includes 34 titles and covers 3 pages. The work contains 64 figures and 12 tables. The total volume of the work is 99 pages.

The purpose of the work is to develop a web application for completing tasks and tests, which is designed to increase the efficiency of the process of information exchange between participants in the school learning process.

The paper identifies the main tasks for the development of the system, analyses the requirements and functionality of the application, and analyses analogues of the created software product. The system architecture was chosen, the database was designed, and a set of modern development technologies was selected, the use of which ensures high performance of the application. The software implementation of the system is described.

KEYWORDS: WEB APPLICATION, EDUCATIONAL PROCESS, SCHOOL, NODE.JS, REACT.JS, NEXT.JS.

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ СИСТЕМИ ВИКОНАННЯ ЗАВДАНЬ ТА ТЕСТІВ.....	8
1.1 Постановка задачі.....	8
1.2 Аналіз аналогів програмного продукту.....	10
1.3 Вибір архітектури веборієнтованої системи виконання завдань та тестів	17
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	21
Висновки до першого розділу.....	25
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ЗДАЧІ ЗАВДАНЬ ТА ТЕСТІВ ДЛЯ ШКІЛЬНОГО ЗАКЛАДУ ОСВІТИ.....	26
2.1 Визначення варіантів використання та структури системи.....	26
2.2 Розробка бази даних системи.....	32
2.3 Проектування та реалізація алгоритмів роботи системи.....	36
Висновки до другого розділу.....	59
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ ЗДАЧІ ЗАВДАНЬ ТА ТЕСТІВ ДЛЯ ШКІЛЬНОГО ЗАКЛАДУ ОСВІТИ.....	61
3.1 Порядок встановлення та налаштування параметрів системи.....	61
3.2 Структура інтерфейсу вебдодатку та порядок роботи.....	62
3.3 Тестування роботи системи.....	78
Висновки до третього розділу.....	80
ВИСНОВКИ.....	81
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	82
ДОДАТКИ.....	85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

WEBSOCKET – протокол зв'язку між браузером та сервером

БД – база даних

ППЗ – проміжне програмне забезпечення

СУБД – система управління базами даних

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6

ВСТУП

Темою даного дипломного проєкту є розробка системи виконання завдань та тестів для шкільного закладу освіти.

Актуальність теми підтверджується наступними аспектами:

1. Необхідність підтримки дистанційного навчання: У часи світової пандемії COVID-19, була виявлена потреба в розвитку систем, що допоможуть ефективніше організовувати навчальний процес в дистанційних умовах

2. Застосування сучасних освітніх технологій: Розвиток інформаційних технологій вимагає впровадження сучасних засобів навчання, які забезпечують ефективну взаємодію між вчителями та учнями при виконанні навчального плану

Отже, розробка системи виконання завдань та тестів для шкільного закладу освіти має велике значення в контексті сучасних викликів освітньої галузі.

Метою дослідження є розробка алгоритмів роботи та реалізація системи виконання завдань та тестів для шкільного закладу освіти.

Встановлена мета обумовлює наступні завдання:

- проведення аналізу задачі;
- проведення аналізу аналогів програмного продукту;
- визначення архітектури та узагальненої структури системи;
- обґрунтування та вибір засобів реалізації системи;
- розробка системи виконання завдань та тестів;

Об'єктом дослідження є інформаційні технології обміну інформацією під час навчального процесу.

Предметом дослідження є використання веборієнтованих технологій для розробки системи виконання завдань та тестів для шкільного закладу освіти.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ СИСТЕМИ ВИКОНАННЯ ЗАВДАНЬ ТА ТЕСТІВ

1.1 Постановка задачі

Технічним завданням є розробка системи виконання завдань та тестів для шкільного закладу освіти.

Метою даної кваліфікаційної роботи є створення інструменту, що забезпечить зручний режим доступу зацікавлених осіб до завдань та зручний спосіб їх публікації, надасть ефективні способи комунікації вчителя та учня, полегшить проведення тестів, забезпечить надійне зберігання даних. Продукт закликаний охопити функції, пов'язані з виконанням завдань та тестів, що підлягають автоматизації для підвищення ефективності проведення навчального процесу та покращення швидкодії взаємодії навчального персоналу та учнів школи. Розробка системи повинна проводитись з використанням сучасних технологій, що дозволить легке її масштабування в майбутньому.

Система повинна надавати можливість вчителям публікувати завдання та тести. Учні повинні мати змогу переглянути назначені їм завдання та тести й виконувати їх. Системний адміністратор повинен мати змогу в повній мірі керувати користувачами системи задля забезпечення безпеки та надійності, керувати та створювати класи, навчальні предмети.

Розробка включатиме в себе створення веборієнтованої системи шляхом побудови її серверної та клієнтської частини, проектування графічного інтерфейсу користувача та налаштування відповідної бази даних.

Основними функціональними вимогами даного проекту є:

1. Можливість авторизації користувачів системи у створений системним адміністратором обліковий запис. Для кожного користувача буде створено обліковий запис, що використовуватиметься для доступу до системи.

2. Можливість створення та виконання завдань та тестів. Вчителі повинні мати можливість створювати завдання та тести, а учні повинні мати змогу виконувати та завантажувати результати виконання поставлених вчителем завдань та тестів.

3. Можливість зворотного зв'язку між вчителем та учнем. Учень та вчитель повинні мати змогу підтримувати контакт шляхом коментарів під пов'язаним завданням.

4. Можливість перегляду успішності. Необхідна наявність інформації про успішність учня на сторінці батьків, необхідна можливість перегляду результату оцінювання завдання чи тесту вчителем чи учнем.

5. Можливість керування ролями та повноваженнями користувачів, інформацією про їх облікові записи. Системний адміністратор повинен мати контроль над ролями користувачів для забезпечення безпеки.

Основними технічними вимогами даного проекту є:

1. Продуктивність: Система повинна бути оптимізована для одночасної роботи з великою кількістю користувачів.

2. Масштабованість: Система повинна бути спроектована таким чином, щоб впоратися з ростом, як з точки зору бази користувачів, так і з точки зору обсягу даних.

3. Надійність: Система повинна бути надійною та доступною для використання в будь-який час.

4. Зручність обслуговування: Система повинна бути розроблена таким чином, щоб її було легко оновлювати та підтримувати.

5. Зручність використання: Інтерфейс користувача повинен бути інтуїтивно зрозумілим і простим у використанні.

Отже, розробка системи виконання завдань та тестів для шкільного закладу освіти включає в себе великий перелік задач. Результатом виконання даних задач має бути система, що забезпечить ефективний спосіб підтримки навчального процесу в умовах дистанційної форми навчання, оптимізує

процес взаємодії навчального персоналу з учнями та надасть зручний спосіб контролю над інформацією, що зберігається у системі.

1.2 Аналіз аналогів програмного продукту

Проаналізувавши ринок доступних рішень, було знайдено перелік аналогічних систем, що розробляється програмних продуктів, які мають схожий функціонал та вирішують аналогічні проблеми.

Moodle – це система управління навчанням з відкритим кодом, яка використовується для створення онлайн-курсів та вебдодатків для навчання [1]. Вона використовується навчальними закладами, організаціями та окремими людьми для створення та надання доступу до навчальних матеріалів, взаємодії зі студентами, тестування та інших освітніх цілей.

Переваги Moodle:

1. Безкоштовність: Moodle - це безкоштовна платформа, яку можна використовувати без жодних ліцензійних зборів.

2. Відкритий код: Moodle - це проект з відкритим кодом, що означає, що його можна безкоштовно використовувати, модифікувати та розповсюджувати.

3. Гнучкість: Moodle може бути налаштована для різних потреб, завдяки великій кількості модулів та тем.

4. Масштабованість: Moodle може використовуватися для невеликих курсів з кількома учнями, а також для великих курсів з тисячами учнів.

5. Безпека: Moodle - це безпечна платформа, яка використовує різні методи аутентифікації та авторизації.

Недоліки Moodle:

1. Складність: Moodle може бути складною для вивчення та адміністрування.

2. Необхідність у досвідчених фахівцях: Для розробки та адміністрування складних систем на Moodle можуть знадобитися досвідчені фахівці.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						10

3. Застарілий дизайн: Moodle може мати застарілий дизайн, який не здається таким сучасним, як у інших систем.

Основний функціонал який надає платформа Moodle:

- можливість подавати навчальний матеріал у різних форматах, таких як текст, презентація, відеоматеріал, вебсторінка тощо (рис. 1.1);
- широкий спектр інструментів моніторингу навчальної діяльності учнів, загальної успішності учня/учениці або класу в процесі виконання тестових завдань тощо (рис. 1.2);
- можливість вибору часу та темпу навчання відповідно до індивідуальних потреб студента;
- зручність спілкування з викладачами та іншими студентами за допомогою електронної пошти, чату, дискусійних форумів тощо;
- можливість створювати навчальні заняття з різноманітним змістом і налагоджувати їх відповідно до навчальних цілей.

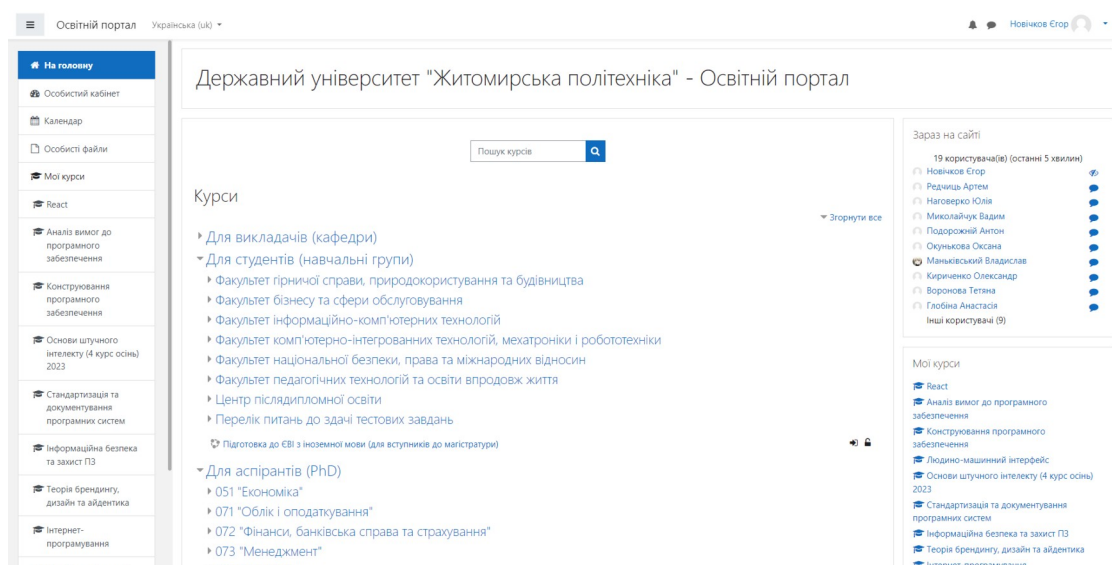


Рис. 1.1 Головна сторінка освітнього порталу «Житомирської Політехніки», який розроблений використовуючи Moodle

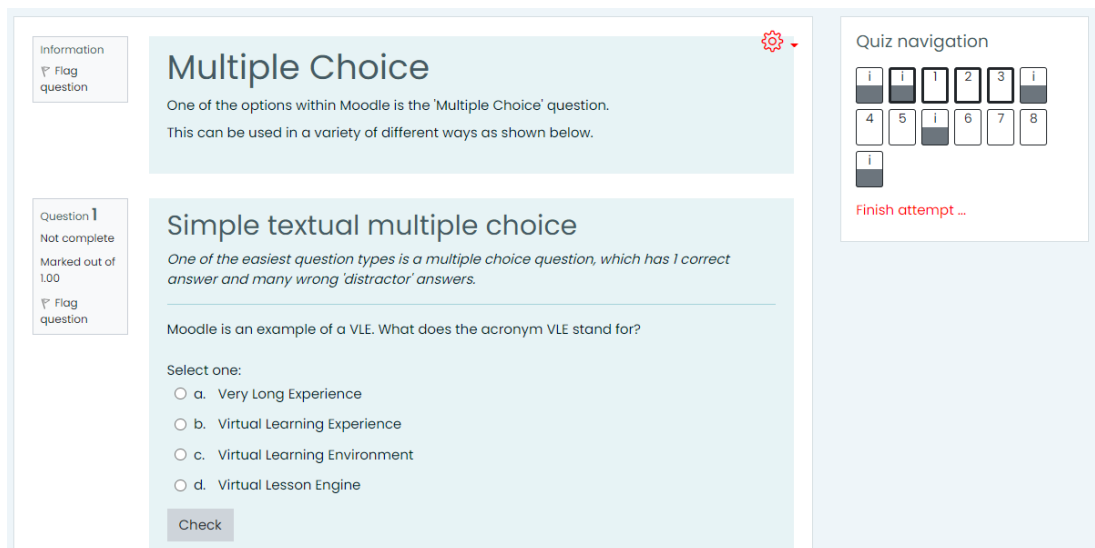


Рис. 1.2 Проходження тесту на платформі Moodle

HUMAN Школа – це система управління навчанням, що об'єднує всіх учасників освітнього процесу та допомагає вчителям та керівникам шкіл організувати внутрішні бізнес-процеси закладу освіти [2].

Переваги HUMAN Школа:

1. Повна інтегрованість: Human охоплює всіх учасників освітнього процесу, пропагує перехід на безпаперовий електронний документообіг.
2. Цілісність та централізованість: Коли всі школи міста об'єднані в Human, учень може переходити в інший навчальний заклад без втрати своїх освітніх здобутків.
3. Персоналізація: Human дає можливість учням навчатися в індивідуальному темпі та з урахуванням їхніх потреб.
4. Доступність: Система доступна з будь-якого пристрою з підключенням до Інтернету.
5. Безпека: Human має Сертифікат на відповідність КСЗІ та Сертифікати ISO, що підтверджують захист даних на платформі.

Недоліки HUMAN Школа:

1. Складність: Деяким вчителям може бути складно освоїти систему.
2. Вартість: Використання Human може бути платним для шкіл.
3. Необхідність у технічній підтримці: Школам може знадобитися технічна підтримка для використання Human.

Основний функціонал який надає платформа HUMAN Школа:

- можливість створювати уроки за допомогою конструктора, додаючи візуалізовані матеріали;
- можливість переглядати розклад уроків, подій, зміст та план уроків (рис. 1.3);
- можливість видавати, здавати та перевіряти домашні завдання (рис. 1.4);
- можливість проводити контроль знань за допомогою тестування;
- можливість роботи оголошення та опитування в загальній стрічці школи/класу;
- можливість автоматизувати збір рефлексії учнів;
- можливість комунікувати між учасниками освітнього процесу за допомогою чатів;
- можливість переглядати візуалізовану статистику успішності та відвідуваності по школі/курсу/класу/учню;
- можливість створювати гуртки та факультети, проводити олімпіади тощо.

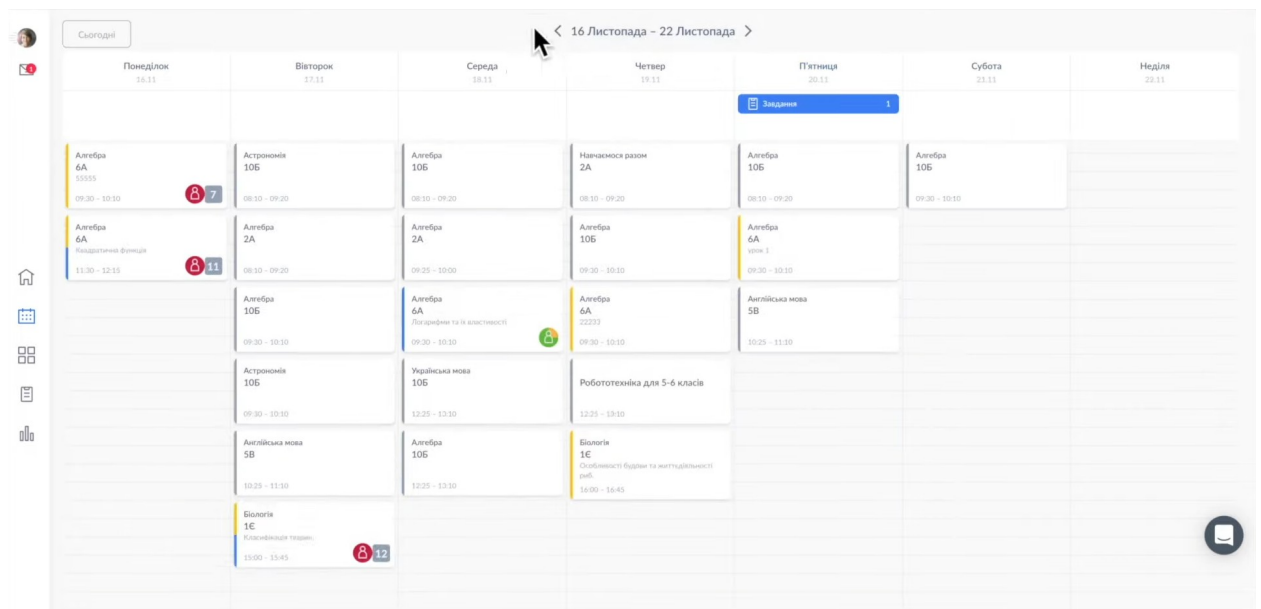


Рис. 1.3 Сторінка розкладу на платформі HUMAN школа

3. Конфіденційність даних: Деякі навчальні заклади можуть бути стурбовані конфіденційністю даних учнів, які зберігаються на серверах Google.

Основний функціонал Google Classroom:

- створення та керування віртуальними класами;
- публікація та розповсюдження навчальних матеріалів;
- видача, збір та оцінювання завдань (рис. 1.5);
- спілкування між вчителями та учнями;
- моніторинг активності та успішності учнів;

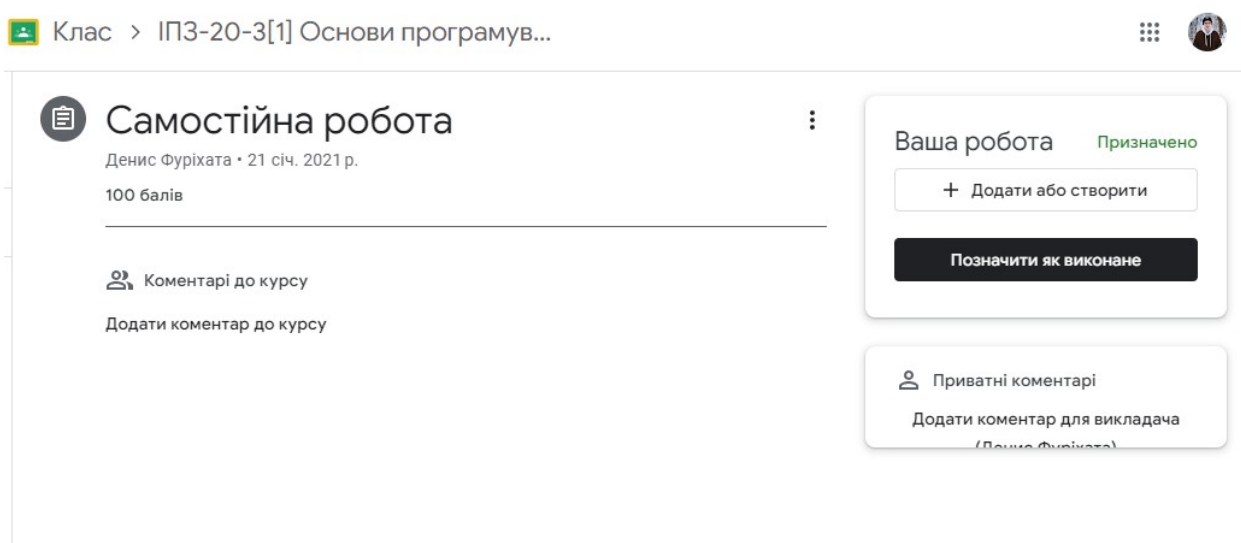


Рис. 1.5 Сторінка завдання в Google Classroom

Проаналізувавши дані продукти, було сформовано порівняння характеристик цих аналогів (табл. 1.1).

Таблиця 1.1

Характеристика	Moodle	HUMAN школа	Google Classroom	Мій продукт
Реєстрація користувачів	+	+	+	+
Поділ учнів на класи	-	+	+-	+
Публікація матеріалів	+	+	+-	-
Публікація й виконання завдань	+	+	+	+
Публікація й виконання тестів	+	+	-	+
Перегляд успішності	+	+	-	+-
Моніторинг навчальної діяльності	+	+	-	+

Продовження таблиці 1.1

Інтеграція з іншими сервісами	-	-	+	+-
Зручність використання	-	+	+	+
Мобільність	-	-	+-	+

Система навчання HUMAN та Moodle мають деякі спільні переваги, такі як можливість використання різноманітних форматів навчального матеріалу, зручність спілкування між учнями та вчителями, можливість моніторингу навчальної діяльності учнів. Однак, HUMAN має деякі особливості, які відрізняють її від Moodle, зокрема, єдина система забезпечує повноцінний освітній процес, можливість створення ID-профілю, що зберігає історію навчальної успішності, та можливість перевести документообіг у електронний формат. Moodle у свою чергу надає більшу гнучкість по відношенню створення онлайн-курсів та навчальних матеріалів у різних форматах.

Google Classroom, в свою чергу, є більш простим та інтуїтивно зрозумілим рішенням, орієнтованим на потреби шкіл. Воно надає базовий набір функцій для організації навчального процесу, таких як створення та управління віртуальними класами, публікація та розповсюдження матеріалів, видача та оцінювання завдань. Однак, Google Classroom має обмежений функціонал у порівнянні з повноцінними системами управління навчальним процесом, такими як Moodle чи HUMAN Школа.

Отже, HUMAN Школа має спеціалізовані функції, спрямовані на управління освітнім процесом в українських школах, тоді як Moodle надає більш загальні інструменти для створення онлайн-курсів та спілкування між учасниками навчального процесу. Google Classroom же є більш простим та доступним рішенням, орієнтованим на потреби шкільної освіти.

Проаналізувавши дані аналоги та розглянувши їх функціонал та можливості, було визначено перелік функціональних вимог, які мають бути виконані при розробці системи:

- створення облікових записів всіх учасників навчального процесу та авторизація;
- контроль над такими даними як предмети та класи;
- створення та виконання завдань та тестів;
- перегляд успішності учня;
- комунікація між учасниками навчального процесу.

1.3 Вибір архітектури веборієнтованої системи виконання завдань та тестів

Вибір архітектури веборієнтованої системи є ключовим етапом розробки вебсистем, що встановлює подальший курс розробки. Обрана архітектура повинна забезпечити прийнятну швидкодію та надійність системи, наявність можливостей для створення функціоналу додатку у повній мірі. Водночас, обрана архітектура не повинна не виправдано ускладнювати процес написання, розуміння чи підтримання програмного коду та робити структуру проекту більш комплексною.

Було розглянуто такі найпопулярніші архітектури веборієнтованих систем як монолітна архітектура, клієнт-серверна архітектура та чиста архітектура.

Монолітна архітектура - класичний і традиційний підхід до розробки програмного забезпечення, де всі компоненти системи інтегровані в один єдиний модуль або додаток. У такій архітектурі всі функціональні можливості, включаючи інтерфейс користувача, бізнес-логіку і доступ до даних, знаходяться всередині одного процесу. Це означає, що при зміні однієї частини, часто потрібно перескомпілювати і деплоїти весь додаток. Всі компоненти тісно зв'язані і розробляються як єдиний проєкт.

Переваги монолітної архітектури:

1. Простота: монолітні системи є більш простими для розробки, оскільки вони не вимагають координації між кількома сервісами. Весь код знаходиться в одному місці, що полегшує розуміння і управління проєктом.

2. Швидкість та ефективність: монолітні системи зазвичай швидші, ніж багаторівневі.

3. Ефективність: оскільки всі частини програми працюють у межах одного процесу, монолітні додатки зазвичай працюють швидше, ніж багаторівневі або розподілені системи. Також такі додатки часто можуть ефективніше використовувати системні ресурси, оскільки немає необхідності керувати декількома процесами або контейнерами.

Недоліки Monolithic архітектури:

1. Складність масштабування: монолітні додатки можуть стати великими і громіздкими, що ускладнює їх масштабування. Щоб обробляти більше навантаження, потрібно масштабувати весь додаток цілком, а не окремі його частини.

2. Складність тестування: у великих монолітних додатках, де всі частини інтегровані разом, складно ізолювати і тестувати окремі компоненти. Це ускладнює процес виявлення і виправлення помилок.

3. Гнучкість: монолітні системи можуть бути менш гнучкими, коли справа доходить до змін або адаптації. Зміна однієї частини системи може вимагати змін в інших частинах, що робить важким внесення змін або інтеграцію нових технологій.

Клієнт-серверна архітектура, це модель розподілених систем, де обробка даних та виконання задач розділені між клієнтами та серверами.

Клієнт – це програма або пристрій, який ініціює запити до сервера. Клієнти взаємодіють з користувачами та зазвичай відповідають за відображення даних, введення інформації та обробку введених даних перед відправленням на сервер. Клієнт може бути веббраузером, мобільним додатком або будь-якою іншою програмою, яка використовує мережу для зв'язку з сервером.

Сервер – це потужний комп'ютер або набір комп'ютерів, які обробляють запити від клієнтів, керують базами даних, виконують бізнес-логіку та відправляють відповіді назад клієнтам. Сервери можуть обробляти

велику кількість одночасних запитів і зазвичай мають доступ до великих обсягів даних та обчислювальних ресурсів.

Переваги клієнт-серверної архітектури:

1. Модульність: клієнти і сервери можуть розроблятися і тестуватися незалежно один від одного.

2. Зрозумілість: чіткий поділ між клієнтською частиною, яка взаємодіє з користувачем, і серверною частиною, яка виконує обробку даних, робить архітектуру зрозумілою та легкою для управління. Це полегшує розуміння і модифікацію коду.

Недоліки клієнт-серверної архітектури:

1. Складність: проектування та впровадження клієнт-серверних систем може бути складнішим у порівнянні з монолітними системами, де вся логіка знаходиться в одному місці. Це пов'язано з необхідністю забезпечення безпечної та ефективної взаємодії між клієнтом і сервером.

2. Продуктивність: передача даних між клієнтом і сервером може створювати затримки і впливати на загальну продуктивність системи.

Чиста архітектура, також відома як hexagonal архітектура або ports and adapters архітектура, підкреслює чіткий розподіл відповідальностей і залежностей між компонентами системи. Вона спрямована на забезпечення високого рівня модульності, гнучкості та легкості тестування.

Вона поділяє систему на два основних компоненти:

1. Ядро: це центральна частина системи, яка містить бізнес-логіку та правила предметної області. Ядро не залежить від зовнішніх компонентів, таких як користувацький інтерфейс або бази даних. Основна задача ядра – це реалізація бізнес-правил та управління даними у спосіб, що не залежить від деталей реалізації.

2. Інтерфейси: відповідають за взаємодію з зовнішнім світом. Це можуть бути користувацькі інтерфейси, API, бази даних, файлові системи або інші сервіси. Інтерфейси є адаптерами, які переводять зовнішні запити у внутрішні формати, що розуміє ядро, і навпаки.

Переваги чистої архітектури:

1. Модульність: сприяє високій модульності системи, що дозволяє легко розробляти та тестувати компоненти незалежно один від одного. Завдяки чітко визначеним залежностям, код стає легшим для тестування.

2. Гнучкість: Додавання нових функцій або змінення тих, що існують стає легшим, оскільки залежності між компонентами мінімізовані.

Недоліки Clean архітектури:

1. Складність: впровадження чистої архітектури може бути складним завданням, особливо для великих систем з багатьма інтерфейсами. Розробка та підтримка такої системи вимагає ретельного планування та архітектурного дизайну.

2. Продуктивність: додатковий рівень абстракції, який вводиться для розділення ядра і інтерфейсів, може призвести до зниження продуктивності.

Після детального аналізу завдань та вимог до майбутнього продукту, було прийнято рішення зупинитися на клієнт-серверній архітектурі. Це рішення забезпечує оптимальний баланс між структурованістю і гнучкістю, які пропонує Clean архітектура, та простотою реалізації, характерною для Monolithic архітектури.

У контексті клієнт-серверної архітектури, користувачі взаємодіють з системою через клієнтську частину. Ця частина виступає посередником, що формує та відправляє запити до серверної частини системи. Сервер, у свою чергу, обробляє ці запити, використовуючи вбудовану бізнес-логіку, і здійснює доступ до даних. Відповідь, сформована сервером, повертається клієнту для подальшого відображення або обробки.

Однією з ключових переваг клієнт-серверної архітектури є чіткий поділ обов'язків між різними компонентами системи. Клієнтська частина відповідальна за взаємодію з користувачем і забезпечення інтерфейсу для введення та отримання даних. Клієнтська частина не повинна здійснювати прямий доступ до бази даних або обробляти дані безпосередньо через інші засоби. Серверна частина, в свою чергу, обробляє запити, виконуючи

необхідну бізнес-логіку і операції з даними. Сервер не взаємодіє з кінцевими користувачами безпосередньо, а тільки через клієнтську частину.

Клієнт-серверна архітектура надає баланс між складністю та гнучкістю. Вона забезпечує ясний розподіл обов'язків, що спрощує розробку та підтримку системи, і при цьому надає достатню гнучкість для інтеграції нових функцій і змін. Це робить клієнт-серверну архітектуру ідеальним вибором для проєкту, оскільки вона поєднує в собі кращі риси чистої та монолітної архітектур, відповідаючи вимогам до функціональності та ефективності системи.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

З огляду на визначені задачі та вимоги щодо системи що розробляється та обрану її архітектуру було проаналізовано варіанти інструментальних засобів, доступних для використання при розробці даного проєкту. Оскільки система являється веборієнтованою та обраною для неї архітектурою є клієнт-серверна архітектура, було зроблено вибір на користь наступного переліку інструментів.

Розробка серверної частини системи виконуватиметься на базі кросплатформенного середовища виконання JavaScript з відкритим вихідним кодом – **Node.js** [4]. Node.js працює на движку JavaScript V8 і виконує JavaScript-код поза веббраузером. Той факт, що Node.js працює на основі движка V8 від Google Chrome, сам по собі корисний. Він компілює JavaScript у машинний код перед виконанням, що призводить до швидшої розробки. Google вкладає значні кошти в рушій V8 [5], покращуючи його ефективність та безпеку, що забезпечує стабільність та високопродуктивність додатків що розгортаються. Node.js – це легке середовище виконання. Він використовує асинхронну модель вводу/виводу, керовану подіями, яка гарантує, що майже жодна функція в Node.js не виконує ввід/вивід безпосередньо. Це економить час, який розробники витрачають на розгортання програми. Серед переваг

використання Node.js – його кросплатформенна підтримка. Додатки, створені за допомогою Node.js, можуть легко орієнтуватися на всі основні платформи, включаючи Windows, Linux і macOS – і все це за допомогою однієї кодової бази. Використовуючи певні інструменти, такі як Electron.js чи інші, кодову базу вебдодатку можна використати для розробки десктопного додатку без фундаментальних змін в коді та великих затрат часу та ресурсів.

Express.js відіграє ключову роль у побудові серверної частини вебдодатка в клієнт-серверній архітектурі [6]. Express.js забезпечує обробку HTTP-запитів, маршрутизацію, проміжні програмні забезпечення, доступ до бази даних, підтримку API.

Використання Node.js передбачає написання коду на мові програмування JavaScript [7]. JavaScript є дуже популярною мовою, незважаючи на її недоліки, такі як її динамічна типізація чи інші. Проте простота у вивченні, відносно гарний рівень ефективності та швидкодії й гнучкість при розробці робить її гарним інструментом для розробки вебсистем.

Збереження даних відбуватиметься шляхом використання системи управління реляційними базами даних з відкритим вихідним кодом PostgreSQL [8]. **PostgreSQL** – це гнучка та потужна СУБД, яка може обробляти великі обсяги даних, масштабуватися як вертикально так і горизонтально, безпечна й надійна та сумісна з великою кількістю платформ.

Взаємодія серверної частини додатку з базою даних буде реалізовано інструментами **Prisma ORM** – сучасного інструменту для взаємодії з базою даних, який спрощує роботу з даними в JavaScript додатках [9]. Prisma використовує файл конфігурації, в якому визначаються моделі даних на основі яких виконується генерація та робота з базою даних

Для розробки клієнтської частини системи було обрано **Next.js** – потужний фреймворк з відкритим вихідним кодом для веброзробки на основі React [10]. Next.js надає можливості серверного рендерингу (SSR) та

статичної генерації вебсторінок, що є важливими для створення швидких і ефективних вебдодатків.

Next.js надає такі переваги як:

1. Серверний рендеринг, що дозволяє генерувати HTML на сервері до його надсилання на клієнт. Це значно покращує продуктивність додатків, зменшуючи час першого відображення.

2. Статична генерація, що дозволяє створювати вебсторінки заздалегідь під час збору. Це означає, що HTML генерується під час розгортання, а не на запит, що робить сторінки надзвичайно швидкими і легко кешованими.

3. Інкрементальна статична регенерація, що дозволяє автоматично оновлювати статичні сторінки без повного перескладання. Це означає, що сторінки можуть бути відновлені на льоту, зберігаючи продуктивність статичних сайтів.

React.js, що є основою Nest.js – це фронт-енд JavaScript-бібліотека з відкритим вихідним кодом для побудови користувацьких інтерфейсів на основі компонентів [11]. React.js дозволяє будувати динамічні single-page аплікації, що гарно поєднуються з асинхронним принципом роботи Node.js. React.js використовує віртуальний DOM для ефективного оновлення інтерфейсу. Замість перезавантаження всієї сторінки при зміні стану, React оновлює лише ті частини, які дійсно змінилися. Це значно покращує продуктивність додатків, зменшуючи обсяг роботи, який браузер повинен виконати під час кожної зміни. У React.js дані передаються від батьківських компонентів до дочірніх, що забезпечує передбачуваність і контрольованість стану додатку. Це робить код зрозумілим і легким для відстеження. React.js використовує JSX, що дозволяє поєднувати HTML з JavaScript у одному файлі. Це спрощує розробку та знижує кількість коду, необхідного для створення складних інтерфейсів. React.js дозволяє будувати додатки з використанням компонентів, що є незалежними та багаторазово використовуваними частинами інтерфейсу. Це спрощує розробку та підтримку великих додатків.

Для дизайну компонентів графічного інтерфейсу системи було обрано CSS фреймворк з відкритим вихідним кодом Tailwind CSS, а також бібліотеку компонентів ShadCN. Ці інструменти забезпечують гнучкість і ефективність у розробці сучасних вебдодатків, дозволяючи створювати стильний, адаптивний та інтуїтивно зрозумілий інтерфейс користувача.

Tailwind CSS є утилітарним CSS-фреймворком, що надає набір низькорівневих класів для створення комплексних інтерфейсів без обмежень [12]. На відміну від традиційних CSS-фреймворків, таких як Bootstrap, Tailwind CSS не надає готових компонентів (кнопок, таблиць тощо), а натомість пропонує набір утилітарних класів, які можна комбінувати для досягнення бажаних стилів. Це забезпечує гнучкість при розробці та надає можливість не прив'язуватись до конкретних класів для реалізації того чи іншого функціоналу. Tailwind CSS надає потужну підтримку адаптивності через медіа-запити, які в Tailwind називаються варіантами (variants). Це дозволяє легко створювати інтерфейси, що змінюють свій вигляд залежно від розміру екрану чи стану елемента. Наприклад, можна задати різні стилі для кнопок при наведенні курсору (hover:) або при виборі за допомогою клавіатури (focus:), що значно покращує взаємодію користувача.

ShadCN є бібліотекою компонентів, що доповнює Tailwind CSS, надаючи набори готових, але легко налаштовуваних компонентів [13]. Вона інтегрується з Tailwind і дозволяє швидко створювати сучасні інтерфейси з мінімальними зусиллями.

Отже, для реалізації серверної частини системи було обрано середовище виконання Node.js використовуючи інструменти мови JavaScript. У ролі бази даних системи виступатиме PostgreSQL. Клієнтська частина буде виконана за допомогою Next.js. Графічний інтерфейс користувача буде реалізовано за допомогою React.js.

Висновки до першого розділу

У першому розділі було проведено аналіз предметної області та вимог до розробки програмного забезпечення. Як результат, було сформовано основні завдання та вимоги до системи, що створюється. Було визначено функціонал системи та етапи її реалізації.

Порівнявши аналоги програмного продукту, було визначено переваги та недоліки цих систем та сформовано відповідний перелік функціональних вимог до системи, що розробляється.

Дослідивши доступні інструментальні засоби, було сформовано набір технологій, що найкращим чином забезпечить реалізацію функціоналу системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ЗДАЧІ ЗАВДАНЬ ТА ТЕСТІВ ДЛЯ ШКІЛЬНОГО ЗАКЛАДУ ОСВІТИ

2.1 Визначення варіантів використання та структури системи

Для забезпечення неперервного виконання необхідних функцій і відповідності встановленим стандартам та специфікаціям, наведено високорівневі вимоги, які повинна задовольняти веборієнтована система підтримки дистанційної форми навчання у загальноосвітній школі. Ці вимоги допомагають забезпечити безперебійну роботу системи та виконання всіх необхідних стандартів і специфікацій.

Бізнес вимоги:

1. Основні цілі: проєкт створюється з метою спростити взаємодію між вчителями та учнями, надати централізований доступ до завдань й тестів та автоматизувати певні процеси.

2. Представлення проєкту: проєкт буде реалізовано у вигляді вебдодатку, що матиме необхідний функціонал та інформацію для виконання завдань та тестів під час шкільного навчального процесу.

Вимоги користувачів:

- можливість редагувати особисті данні користувачів системним адміністратором, керувати їх обліковими записами;
- можливість створювати та керувати тестами та завданнями, виконувати завдання у додатку або завантажувати результат виконання у вигляді фото;
- наявність ефективних засобів комунікації між викладачами та учнями у вигляді коментарів;
- можливість оцінювати учнів та переглядати їх успішність;
- наявність зручного та зрозумілого інтерфейсу, в якому легко орієнтуватися як викладачам, так і учням;
- сумісність з різними пристроями та операційними системами;

– можливість перегляду, створення та керування усіма даними пов'язаними з класами та навчальними предметами.

Користувач додатку буде мати можливість переглядати всю доступну йому інформацію пов'язану з навчальним процесом. Вчитель зможе створювати потрібні завдання та тести, використовуючи функціонал додатку, комунікувати з учнем за допомогою коментарів, оцінювати його та переглядати його успішність. Учень зможе виконувати створені для його класу завдання та тести, комунікувати з вчителем за допомогою коментарів, бачити результати оцінювання. Системний адміністратор зможе керувати обліковими записами користувачів системи, переглядати, керувати та створювати класи та навчальні предмети. Батьки зможуть переглядати успішність прив'язаних до їх облікового запису учнів. Користувачі зможуть ефективно використовувати функціонал додатку через його простоту та зручність інтерфейсу.

Функціональні вимоги:

1. Авторизація в системі: в системі повинна бути наявна можливість створення облікових записів користувачів адміністратором системи, присвоєння їм відповідних ролей та можливість реєстрації та авторизації використовуючи ці облікові записи.

2. Збереження та доступ до інформації: система повинна надавати можливість створення та контролю даних, доступ до них та їх збереження.

3. Система повинна надавати можливість створення та виконання завдань та тестів.

4. Зв'язок: система повинна надавати можливість комунікації у реальному часі між вчителем та учнем у текстовому форматі.

Нефункціональні вимоги:

1. Сприйняття: Час, необхідний для навчання звичайних користувачів – 3 робочих дня (24 години), для навчання просунутих користувачів – 1 робочий день (8 годин). Час відгуку для типових задач - не більше 3 секунди,

для складних завдань - не більше 10 секунд. Інтерфейс має бути простим та зрозумілим, з достатньою кількістю відповідних підказок.

2. Надійність: Доступність – час, що витрачається на обслуговування системи не повинно перевищувати 3% від загального часу роботи. Середній час безвідмовної роботи – 10 робочих днів. Максимальна норма помилок або дефектів – 1 помилка на десять тисяч рядків коду.

3. Продуктивність: Система повинна підтримувати мінімум 1000 одночасно активних користувачів, пов'язаних із загальною базою даних.

4. Можливість експлуатації: Масштабування - система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це не вплинуло негативно на її роботу.

Аналіз вимог користувачів дав підстави для визначення варіантів використання системи виконання завдань та тестів для шкільного закладу освіти. Побудовано діаграму варіантів використання програмної системи (рис. 2.1). Дана діаграма відображає варіанти використання функціоналу системи користувачами різних ролей: Адміністратор, Вчитель, Учень та Батько/Мати. Адміністратор має доступ до функціоналу керування обліковими записами, керування класами, керування навчальними предметами. Вчитель має доступ до функціоналу реєстрації та авторизації, перегляду, створення та керування завданнями, перегляду та оцінювання результату виконання завдання, комунікації в коментарях, перегляду, створення та керування тестами, перегляду результату виконання тесту. Учень має доступ до функціоналу реєстрації та авторизації, перегляду та виконання завдань, перегляду результату оцінювання завдання, комунікації в коментарях, перегляду та виконання тестів, перегляду результату оцінювання тесту. Батько/Мати має доступ до функціоналу перегляду результатів виконання завдань та тестів учнем та перегляду статистики результатів учня.

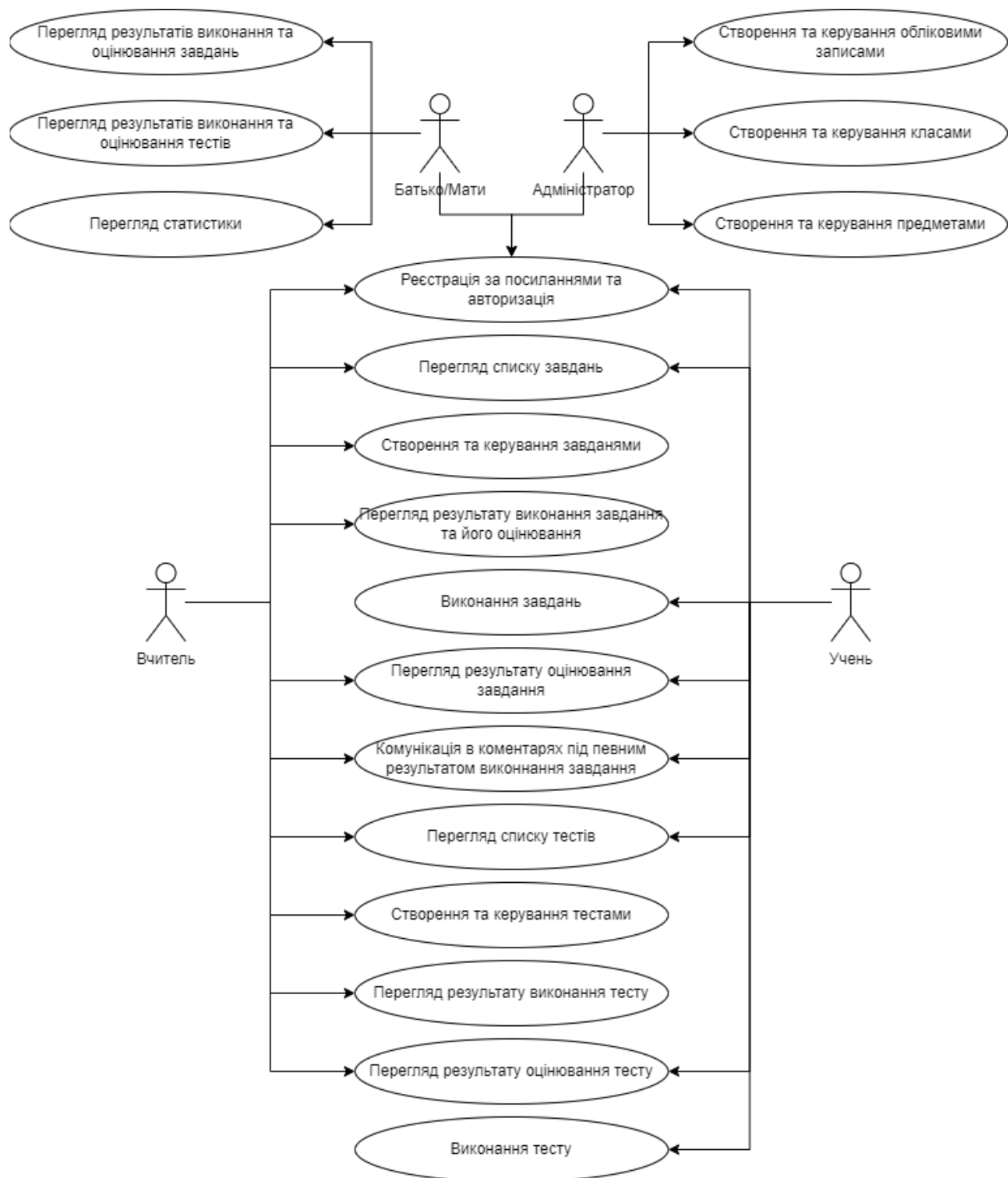


Рис. 2.1 Діаграма варіантів використання системи

Проаналізувавши функціональні вимоги системи, було сформовано необхідні сутності для реалізації необхідного функціоналу. Для візуалізації та опису взаємодій було побудовано діаграму сутностей системи (рис. 2.2) яка також відображає моделі даних системи. Також було побудовано діаграму компонентів системи (рис. 2.3) для відображення взаємодії між компонентами системи.

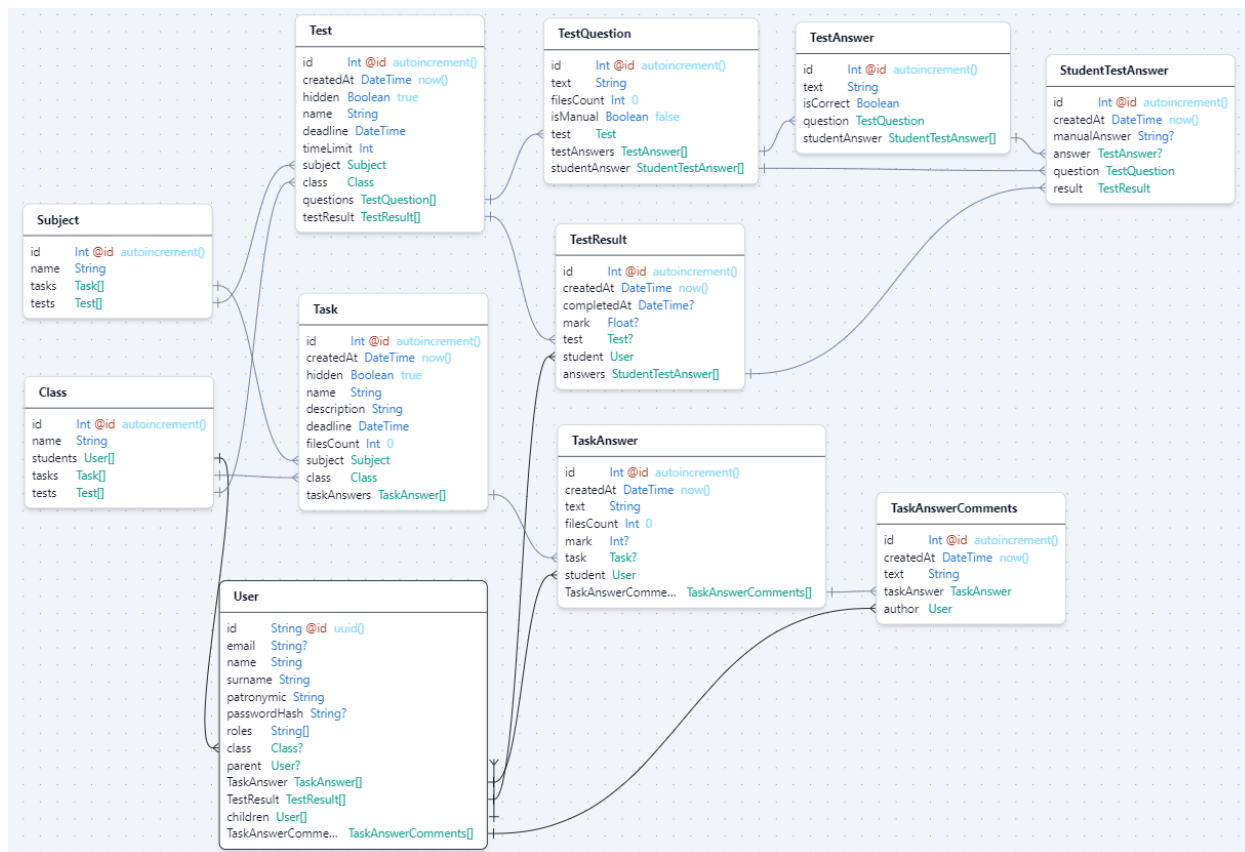


Рис. 2.2 Діаграма сутностей системи

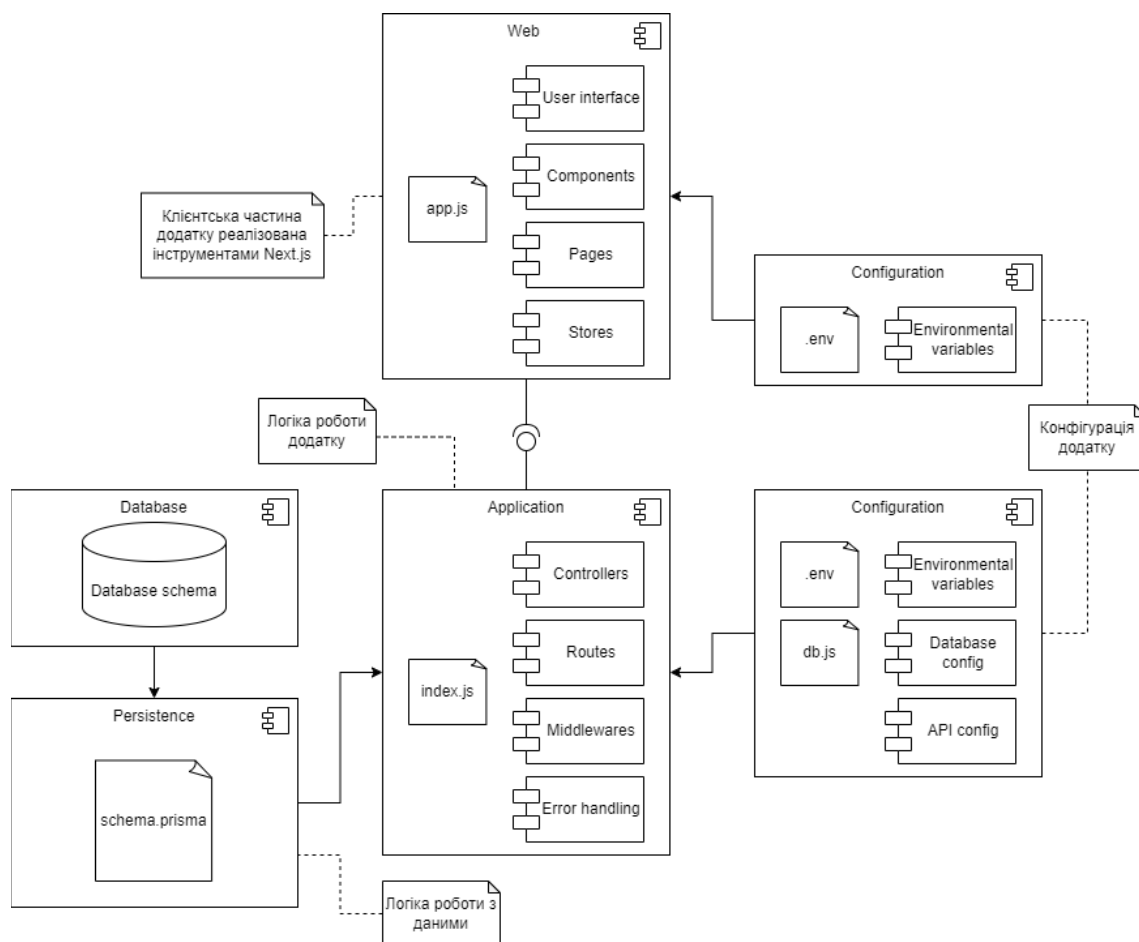


Рис. 2.3 Діаграма компонентів системи

Web – клієнтська частина додатку реалізована за допомогою Next.js, що складається з:

1. User Interface – графічний інтерфейс системи побудований інструментами React.js.

2. Components – React.js компоненти, з яких будується графічний інтерфейс.

3. Pages – окремі сторінки системи, які відображаються на основі Next.js. На основі їх розподілу по директоріям будується навігація клієнтською частиною системи.

4. Stores – механізм збереження даних на найвищому рівні клієнтської частини додатку, для доступу до них з будь якого компоненту.

5. app.js – кореневий файл клієнтської частини додатку.

Application – серверна частина додатку реалізована за допомогою Node.js та Express.js. Вона складається з:

1. Controllers – визначають, як обробляються запити користувача.

2. Routes – Визначення маршрутів для запитів до серверу

3. Middlewares – проміжні обробники запитів, що забезпечують додаткову логіку перед обробкою в контролерах.

4. Error handling – управління та обробка помилок, що виникають у додатку.

5. index.js – головний файл серверної частини додатку на серверній стороні.

Persistence – це компонент логіки роботи з даними. Включає в себе schema.prisma – схему бази даних, визначену з використанням Prisma ORM.

Configuration – це компонент, що включає в себе конфігураційні файли такі як .env файли, що містять змінні середовища та db.js файл, що визначає конфігурацію бази даних.

Database – місце збереження даних системи. Структура бази даних відповідає Prisma схемі.

2.2 Розробка бази даних системи

Для керування даними системи здачі завдань та тестів для шкільного закладу освіти буде використовуватись система управління базами даних PostgreSQL.

Сформована для системи база даних складається з 11 таблиць:

1. User – таблиця, що містить користувачів системи.
2. Class – таблиця, що містить класи системи.
3. Subject – таблиця, що містить навчальні предмети системи.
4. Task – таблиця, що містить завдання системи.
5. TaskAnswer – таблиця, що містить відповіді учнів на завдання системи.
6. TaskAnswerComments – таблиця, що містить коментарі до певної відповіді учня на завдання системи.
7. Test – таблиця, що містить тести системи.
8. TestQuestion – таблиця, що містить запитання, що відносяться до певного тесту системи.
9. TestAnswer – таблиця, що містить варіанти відповіді, що відносяться до певного запитання тесту системи.
10. TestResult – таблиця, що містить інформацію про проходження певного тесту учнем системи.
11. StudentTestAnswer – таблиця, що містить інформацію про обрані певним учнем варіанти відповіді на певне запитання тесту системи.

Наведено опис кожної із таблиць в таблицях 2.1 – 2.11.

Таблиця 2.1

Опис таблиці Users

Поле	Тип даних	Опис
id	string	Ідентифікатор користувача
email	string?	Пошта користувача
name	string	Ім'я користувача
surname	string	Прізвище користувача
patronymic	string	По-батькові користувача
passwordHash	string?	Хеш пароля
roles	string[]	Масив ролей користувача

Продовження таблиці 2.1

classId	int?	Ідентифікатор класу
parentId	int?	Ідентифікатор батька

Таблиця 2.2

Опис таблиці Class

Поле	Тип даних	Опис
id	int	Ідентифікатор класу
name	string	Назва класу

Таблиця 2.3

Опис таблиці Subject

Поле	Тип даних	Опис
id	int	Ідентифікатор предмету
name	string	Назва предмету

Таблиця 2.4

Опис таблиці Task

Поле	Тип даних	Опис
id	int	Ідентифікатор завдання
createdAt	dateTime	Дата створення завдання
hidden	boolean	Прихованість завдання
name	string	Назва завдання
description	string	Опис завдання
deadline	dateTime	Кінцевий термін виконання
filesCount	int	Кількість файлів, прикріплених до завдання
subjectId	int	Ідентифікатор предмету, до якого належить завдання
classId	int	Ідентифікатор класу, для якого призначене завдання

Таблиця 2.5

Опис таблиці TaskAnswer

Поле	Тип даних	Опис
id	int	Ідентифікатор відповіді
createdAt	dateTime	Дата створення відповіді
text	string	Текст відповіді
filesCount	int	Кількість файлів, прикріплених до відповіді
mark	int?	Оцінка за відповідь

Продовження таблиці 2.5

taskId	int	Ідентифікатор завдання, до якого належить відповідь
studentId	string	Ідентифікатор студента, який подав відповідь

Таблиця 2.6

Опис таблиці TaskAnswerComments

Поле	Тип даних	Опис
id	int	Ідентифікатор коментарю
createdAt	dateTime	Дата створення коментаря
text	string	Текст коментаря
taskAnswerId	int	Ідентифікатор відповіді, до якої належить коментар
authorId	string	Ідентифікатор користувача, який залишив коментар

Таблиця 2.7

Опис таблиці Test

Поле	Тип даних	Опис
id	int	Ідентифікатор тесту
createdAt	dateTime	Дата створення тесту
hidden	boolean	Прихованість завдання
name	string	Назва тесту
deadline	dateTime	Кінцевий термін виконання
timeLimit	int	Ліміт часу на виконання тесту
subjectId	int	Ідентифікатор предмету, до якого належить тест
classId	int	Ідентифікатор класу, для якого призначений тест

Таблиця 2.8

Опис таблиці TestQuestion

Поле	Тип даних	Опис
id	int	Ідентифікатор питання
text	string	Текст питання
filesCount	int	Кількість файлів, прикріплених до питання
isManual	boolean	Флаг, що вказує, чи питання потребує ручного введення відповіді
testId	int	Ідентифікатор тесту, до якого належить питання

Таблиця 2.9

Опис таблиці TestAnswer

Поле	Тип даних	Опис
id	int	Ідентифікатор відповіді
text	string	Текст відповіді
isCorrect	boolean	Флаг, що вказує, чи відповідь є правильною
questionId	int	Ідентифікатор питання, до якого належить відповідь

Таблиця 2.10

Опис таблиці TestResult

Поле	Тип даних	Опис
id	int	Ідентифікатор результату
createdAt	dateTime	Дата створення результату
completedAt	dateTime?	Дата завершення тесту
mark	float?	Оцінка за тест
testId	int	Ідентифікатор тесту, до якого належить результат
studentId	string	Ідентифікатор студента, який проходив тест

Таблиця 2.11

Опис таблиці StudentTestAnswer

Поле	Тип даних	Опис
id	int	Ідентифікатор відповіді студента
createdAt	dateTime	Дата створення відповіді
manualAnswer	string?	Текст ручної відповіді
answerId	int?	Ідентифікатор відповіді, що обрано
questionId	int	Ідентифікатор питання, на яке дана відповідь
resultId	int	Ідентифікатор результату тесту, до якого належить відповідь

З даних таблиць було сформовано діаграму бази даних (рис. 2.4):

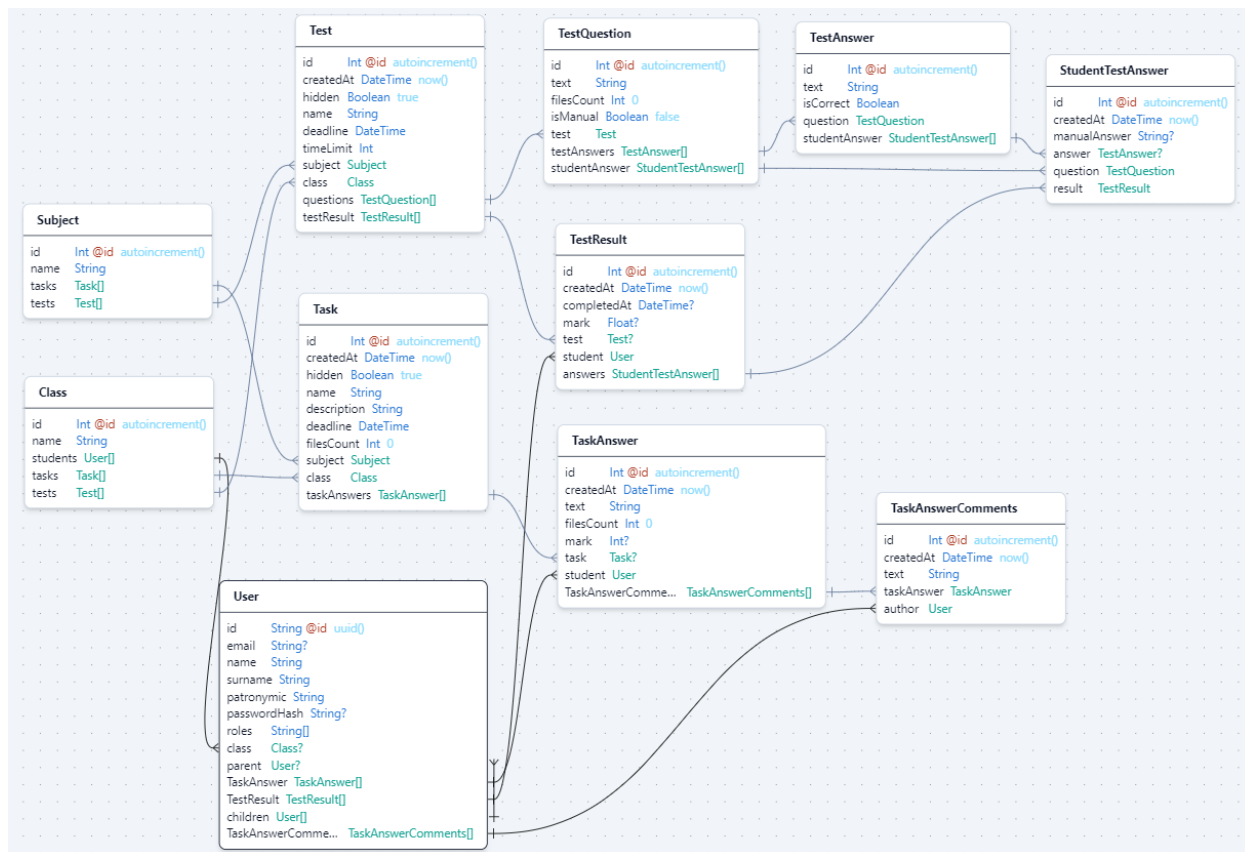


Рис. 2.4 Діаграма бази даних

2.3 Проектування та реалізація алгоритмів роботи системи

Перед тим як почати реалізацію алгоритмів системи потрібно спроектувати логіку взаємодії клієнтської та серверної частини додатку. Серверною частиною додатку виступає Express.js сервер. Це легкий і гнучкий вебфреймворк для Node.js, який спрощує створення вебдодатків та API. Він є одним з найбільш популярних фреймворків для розробки серверної частини вебдодатків на JavaScript, завдяки своїй простоті, масштабованості та багатій екосистемі. Для забезпечення зв'язку з серверною частиною додатку, потрібно ініціалізувати набір кінцевих точок API. Кінцеві точки є шляхами, при переході на які буде викликатись певний метод контролера, що визначає логіку обробки запиту та повернення відповіді з даними. Express.js дозволяє ініціалізувати шляхи з різними типами запитів, таких як GET, POST, PUT, PATCH, тощо (рис. 2.5) (рис. 2.6).

```
router.use('/task', taskRouter)
```

Рис. 2.5 Ініціалізація шляху

```
router.post('/', authMiddleware, checkRole(['teacher', 'admin']), taskController.createTask)
```

Рис. 2.6 Ініціалізація кінцевої точки

Можна побачити (див. рис. 2.6), що окрім методу контролера, в метод ініціалізації кінцевої точки передаються проміжні програмні забезпечення (middleware). Це код, який виконується перед обробкою запиту. У даному випадку, перед обробкою запиту виконується `authMiddleware`, що перевіряє дійсність токена авторизації, прикріпленого до запиту з клієнта, та `checkRole`, що перевіряє роль користувача. Ці два проміжні програмні забезпечення можуть припинити обробку запиту, якщо умови для запиту з клієнта не будуть виконані. Наприклад, `authMiddleware` може відхилити запит, якщо токен авторизації є недійсним або відсутнім, а `checkRole` може заборонити доступ до певного ресурсу, якщо роль користувача не відповідає необхідним правам доступу. Наведено приклад реалізації проміжного програмного забезпечення перевірки ролі (рис. 2.7). ППЗ отримує токен з заголовку авторизації і перевіряє його за допомогою методу `jwt.verify`. Якщо токен недійсний або відсутній, повертається відповідь 401, тобто клієнт неавторизований. Далі ППЗ отримує ролі користувача з бази даних та перевіряє, чи збігаються ролі користувача з будь-якими необхідними ролями, переданими як аргумент до функції ППЗ. Якщо ні, вона повертає відповідь 403, тобто доступ заборонено. Якщо користувач автентифікований і авторизований, ППЗ викликає наступну функцію для продовження конвеєра запитів.

```

module.exports = function (roles) {
  return async function (req, res, next) {
    if (req.method === 'OPTIONS') {
      next()
    }

    try {
      const token = req.headers.authorization.split(' ')[1]
      if (!token) {
        res.status(401).json({ message: 'unauthorised' })
      }

      const decoded = jwt.verify(token, process.env.SECRET_KEY)
      req.user = decoded

      const userRoles = await getUserRolesWithoutRequest(decoded.id)

      if (userRoles.every((r) => !roles.includes(r))) {
        return res.status(403).json({ message: 'forbidden' })
      }

      next()
    } catch (e) {
      res.status(401).json({ message: 'unauthorised' })
    }
  }
}

```

Рис. 2.7 ППЗ перевірки ролі користувача, що зробив запит

Відправлення запитів з клієнту виконується за допомогою бібліотеки Axios. Це популярна бібліотека для виконання HTTP-запитів на JavaScript [14]. Вона дозволяє легко взаємодіяти з сервером з браузера або з Node.js, забезпечуючи простий і зручний API для відправлення запитів. Axios підтримує як асинхронні запити, так і обіцянки, що робить його ідеальним вибором для сучасних вебдодатків. При відправленні запиту на кінцеву точку серверу, що потребує, щоб користувач був авторизованим, до запиту додається токен авторизації користувача що зберігається в сесії. Наведено приклад відправлення запиту з клієнту (рис. 2.8). Ця функція створює нове завдання в системі, надсилаючи POST-запит до кінцевої точки /api/task з наданими даними про завдання. Він повертає дані про створене завдання, які можуть бути використані кодом, що його викликає. \$authHost, з якого виконується запит на сервер, це екземпляр Axios з перехоплювачем для додавання токена автентифікації до запитів. Токен додається як заголовок авторизації у вигляді текстового рядку формату «Bearer {token}»

```
export const createTask = async (name, description, deadline, classId, subjectId) => {
  const { data } = await $authHost.post('api/task', {
    name,
    description: DOMPurify.sanitize(description),
    deadline,
    classId,
    subjectId,
  })
  return data
}
```

Рис. 2.8 Метод для відправлення запиту на створення завдання

Робота з системою починається з авторизації користувача у свій обліковий запис. Системний адміністратор створює облікові записи користувачів, що користуватимуться системою. Після створення облікового запису, користувач зможе зареєструвати свої данні входу до цього облікового запису за спеціальним посиланням. Після успішної реєстрації користувач має змогу зайти до свого облікового запису за допомогою введених ним раніше даних. Система перевіряє чи вона має відповідні дані входу та успішно автентифікує користувача, якщо знаходить відповідність. В іншому випадку, повідомляє користувача про неправильність введених даних. Побудована діаграма активності роботи з обліковими записами (рис. 2.9).

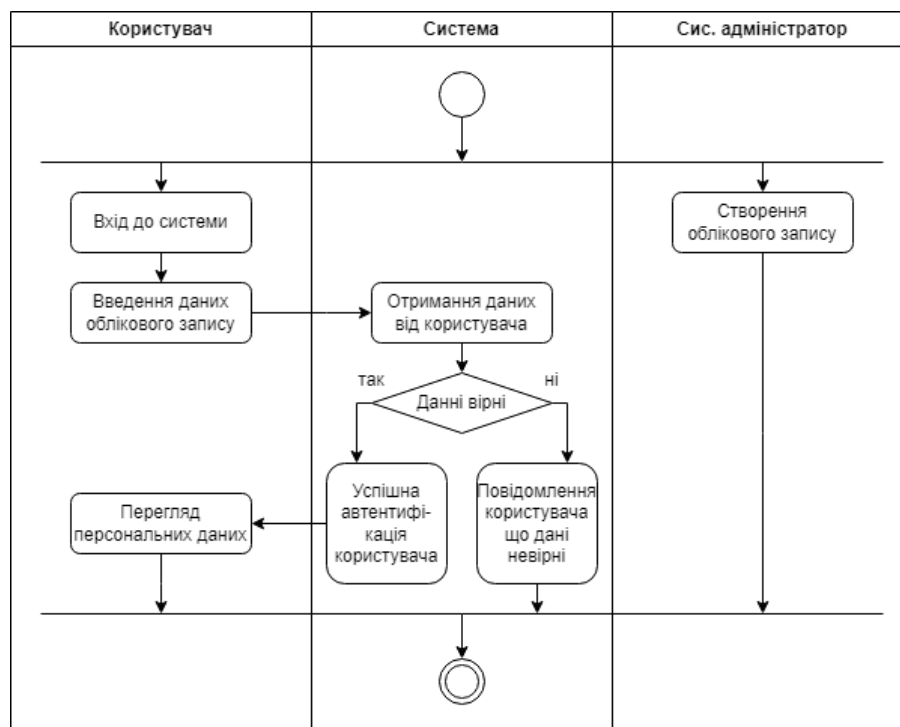


Рис. 2.9 Діаграма активності роботи з обліковими записами

Коли сервер отримує запит на автентифікацію, він проводить валідацію отриманих даних. Після успішної валідації, він шукає користувача з вказаною email адресою. Якщо користувача з такою адресою не існує в системі, сервер повертає помилку. Якщо сервер все ж знайшов користувача з отриманою адресою, він порівнює отриманий пароль з хешем паролю цього користувача, що зберігається в базі даних. Якщо паролі сходяться, генерує та повертає на клієнт jwt токен авторизації, створений за допомогою бібліотеки JSON Web Token [15]. Наведена кодова реалізація цього алгоритму (рис. 2.10).

```

async login(req, res, next) {
  const { email, password } = req.body

  if (!validate(email, password)) {
    return next(ApiError.badRequest('INVALID_DATA'))
  }

  const user = await db.user.findUnique({ where: { email } })
  if (!user) {
    return next(ApiError.badRequest('EMAIL_NOT_FOUND'))
  }

  let comparePassword = bcrypt.compareSync(password, user.passwordHash)
  if (!comparePassword) {
    return next(ApiError.badRequest('WRONG_PASSWORD'))
  }

  const roles = await getUserRolesWithoutRequest(user.id)
  const token = generateToken(user.id, roles)
  return res.json({ token })
}

```

Рис. 2.10 Алгоритм авторизації користувача

Авторизація в клієнтській частині системи реалізована за допомогою бібліотеки NextAuth.js. Це гнучка та повнофункціональна бібліотека для аутентифікації користувачів у додатках, створених на основі Next.js [16]. Вона спрощує процес налаштування різних методів аутентифікації, таких як сторонні провайдери (OAuth), базова аутентифікація за допомогою електронної пошти або власні схеми аутентифікації. Для авторизації використовується метод signIn бібліотеки. Цей метод виконує зазначений у конфігурації авторизації запит для отримання токена авторизації з сервера та зберігає його

в сесії, яка представлена HTTP-only cookie, що містить інформацію про сесію та отриманий з сервера токен авторизації. Оскільки cookie сесії є HTTP-only, клієнтський код браузера не має до нього доступу, тому в додатку Next.js створюється окреме API для доступу до сесії. Наведена частина коду конфігурації NextAuth.js (рис. 2.11).

```
export const authOptions = {
  providers: [
    CredentialsProvider({
      name: 'credentials',
      async authorize(credentials) {
        try {
          const user = await getToken(credentials.email, credentials.password)
          if (user) {
            return user
          }
        } catch (error) {
          throw new Error(`Authentication failed (${error})`)
        }
      },
    }),
  ],
  session: {
    strategy: 'jwt',
    maxAge: 7 * 24 * 60 * 60, // 7 days
  },
}
```

Рис. 2.11 Конфігурація NextAuth.js

Оскільки сесія NextAuth.js залишається дійсною поки не спливе встановлений час її придатності, до неї можна звертатись, щоб отримати токен авторизації авторизованого користувача при переході на вебсайт. Всі дані про користувача окрім токена авторизації (що зберігається в сесії) зберігаються в контексті React.js додатку [17]. Наведено приклад реалізації компоненту контексту (рис. 2.12).

```
const AppContext = createContext()
const userStore = new UserStore()
const parentStore = new ParentStore()

export function AppWrapper({ children }) {
  return <AppContext.Provider value={{ userStore: userStore, parentStore: parentStore }}>{children}</AppContext.Provider>
}

export function useAppContext() {
  return useContext(AppContext)
}
```

Рис. 2.12 Компонент контексту додатку

Для доступу до контексту з будь якого компоненту додатку, весь додаток огортається компонентом, описаним в коді (див. рис. 2.12).

На відміну від сесії, контекст не зберігається після закриття додатку. Тому при відкритті вебсайту додаток перевіряє, чи існує збережена дійсна сесія. Якщо вона існує, додаток відправляє запит до сервера на отримання даних користувача та записує їх у контекст, базуючись на ідентифікаторі користувача, який зашифрований у токени авторизації, збереженому в сесії. Також у цьому алгоритмі відбувається перевірка ролі користувача та надання доступу до певних сторінок вебдодатку. Це реалізовано шляхом визначення ключових шляхів додатку та перевірки ролі користувача шляхом читання інформації з контексту. Наведена частина кодової реалізації цього алгоритму зображена (рис. 2.13). Даний код отримує токен авторизації зі збереженої сесії користувача. Якщо токен наявний, декодує токен за допомогою функції `jwtDecode` та перевіряє, чи не закінчився термін дії токена, порівнюючи час закінчення терміну дії (`exp`) з поточним часом. Якщо токен дійсний, викликає функцію `setUser`, що отримує дані про користувача з бази даних та встановлює дані поточного користувача в контекст за допомогою декодованого токена. Далі код перевіряє, чи має користувач доступ до поточного шляху (`/workspace/`) на основі його ролі. Якщо користувач не має доступу, він буде перенаправлений на сторінку `/not-authorized`. В іншому випадку, користувач буде перенаправлений на сторінку на основі його ролі, якщо він має одну роль, або залишиться на сторінці `workspace` якщо має декілька ролей. Даний алгоритм забезпечує постійну наявність даних про користувач в контексті додатку. Також він керує доступом користувачів до певних частин додатку, в то же час покращуючи зручність використання системи шляхом перенаправлення користувача в разі якщо він має лише одну роль. По своїй суті даний алгоритм являє собою проміжне програмне забезпечення, оскільки код алгоритму знаходиться в файлі `layout.js` директорії, в якій знаходяться всі шляхи додатку, тому даний код виконується при кожному завантаженні сторінки.

```

getCurrentUserAccessToken().then((token) => {
  if (token) {
    const decodedToken = jwtDecode(token)
    const currentTime = Date.now().valueOf() / 1000

    if (decodedToken.exp > currentTime) {
      setUser(decodedToken).then(() => {
        if (!checkAccess(userStore.user.roles, pathname) && pathname.includes('/workspace/')) {
          router.push('/not-authorized')
          return
        } else {
          if (pathname === '/workspace') {
            if (userStore.user.roles.length === 1) {
              router.push(accessRules[userStore.user.roles[0]])
            }
            return
          }
        }
      })
    }
  }
})

```

Рис. 2.13 Відновлення сесії та перевірка доступу

Маршрутизація в додатку Next.js відбувається через чітку ієрархію директорій в корені додатку [18]. Кожна директорія відповідає частині шляху, за яким можна отримати доступ до певної сторінки. Для відображення контенту ця директорія повинна містити файли `page.js` або `layout.js` (або обидва). При переході на певний URL, ієрархія директорій визначає, яка сторінка буде відображена (рис. 2.14).

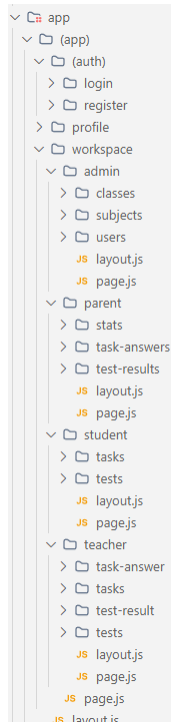


Рис. 2.14 Ієрархія директорій

Next.js також підтримує логічне групування директорій, яке не впливає на шлях. Для цього назву директорії потрібно взяти в дужки. Наприклад, ди-

ректорія (utils) не буде відображена у шляху, але дозволить логічно групувати файли утиліт.

Для визначення динамічного шляху в Next.js назву директорії або файлу потрібно взяти в квадратні дужки. У наведеному прикладі (рис. 2.15), при переході за шляхом формату "workspace/tasks/{ідентифікатор завдання}", на сторінці, що знаходиться в директорії [id], можна буде отримати значення ідентифікатора, яке міститься в шляху, та на його основі робити запити до сервера.

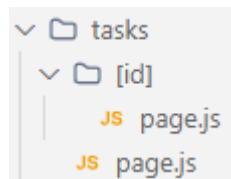


Рис. 2.15 Ієрархія директорії tasks

Для можливості авторизації в системі, користувач спочатку повинен зареєструватись за спеціальним посиланням. Це посилання генерує системний адміністратор після створення облікового запису користувача. Посилання містить в собі зашифрований ідентифікатор облікового запису в базі даних. Шифрування та розшифрування даного ідентифікатора виконується інструментами бібліотеки SimpleCrypto [19]. При переході за даним посиланням, введенні й відправленні даних і їх валідації перед відправленням, зашифрований ідентифікатор розшифровується та на його основі відправляється запит на додавання email адреси та хешу паролю до відповідного запису в базі даних. Наведено кодову реалізацію реєстрації користувача на сервері (рис. 2.16). Код викликає функцію валідації для перевірки введених адреси електронної пошти та пароля. Якщо перевірка не вдалася, повертає помилку badRequest за допомогою next функції. Звертається до бази даних для пошуку користувача з заданим ідентифікатором, використовуючи метод findUnique, і якщо знаходить, перевіряє чи даний користувач вже має email адресу та пароль. Якщо пароль та email існують, повертає помилку badRequest. Також повертає помилку, якщо користувача з даним ідентифікатором не знайдено. Дані код звертається до бази даних для пошуку користувача з заданим email

методом `findUnique`. Якщо такого користувача знайдено, повертає помилку `badRequest`. Після проходження всіх перевірок, код хешує пароль за допомогою бібліотеки `bcrypt` [20] з сольовим коефіцієнтом 5 та оновлює користувача з заданим ідентифікатором, задаючи поля `email` та `passwordHash` методом `update`. Після оновлення користувача генерує токен за допомогою функції `generateToken`, передаючи `updatedUser.id` та `roles` як аргументи.

```

async registration(req, res, next) {
  const { id } = req.params
  const { email, password } = req.body

  if (!validate(email, password)) {
    return next(ApiError.badRequest('INVALID_DATA'))
  }

  const existingUser = await db.user.findUnique({ where: { id } })
  if (existingUser) {
    if (existingUser.email || existingUser.passwordHash) {
      return next(ApiError.badRequest('USER_ALREADY_HAS_EMAIL_AND_PASSWORD'))
    }
  } else {
    return next(ApiError.badRequest('USER_DOES_NOT_EXIST'))
  }

  const candidate = await db.user.findUnique({ where: { email } })
  if (candidate) {
    return next(ApiError.badRequest('EMAIL_EXISTS'))
  }

  const hashPassword = await bcrypt.hash(password, 5)
  const updatedUser = await db.user.update({
    where: { id: id },
    data: {
      email,
      passwordHash: hashPassword,
    },
  })

  const roles = await getUserRolesWithoutRequest(updatedUser.id)
  const token = generateToken(updatedUser.id, roles)
  return res.json({ token })
}

```

Рис. 2.16 Алгоритм реєстрації користувача

Якщо сервер у відповідь на запит відправив помилку про те що дані невірні, користувач вже зареєстрований, користувача не існує або користувач з такою email адресою вже існує, помилка локалізується на клієнті та відобразиться у формі.

Адміністратор має повний контроль над обліковими записами системи. Він має змогу редагувати інформацію збережену в обліковому записі, а також може видалити email адресу та хеш паролю, прив'язані до певного користувача, тим самим «скинути» користувача та надати йому можливість знову зареєструватись з новими даними, в разі якщо користувач забув дані для входу в свій обліковий запис.

При розробці функціоналу динамічного пошуку користувачів (а також функціоналу всіх інших полів динамічного пошуку в додатку) було використано функцію `debounce` бібліотеки `Lodash` [21]. Вона створює функцію, що затримує виклик коду чи функції, що передана в неї як аргумент, до тих пір, поки не пройде зазначена в аргументах кількість мілісекунд з моменту попереднього виклику цієї функції. Це дозволяє значно зменшити кількість запитів на сервер у випадку, якщо ці запити вправляються при зміні значення в полі пошуку.

Іншими можливостями системного адміністратора в системі є створення та керування класами та предметами. Ці об'єкти використовуються для класифікації завдань, тестів та учнів. Учні, в списку доступних їм завдань або тестів будуть бачити лише завдання та тести, створені для його класу. Також за класом та предметом можна виконувати фільтрування в списках завдань та тестів.

Авторизувавшись у системі, користувачі з ролями Вчитель та Учень можуть працювати з завданнями та тестами. Вчитель має змогу створювати та оцінювати завдання й тести, а Учень має змогу виконувати завдання. Було побудовано діаграму активності цього процесу (рис. 2.17).

Вчитель входить до системи та ініціалізує створення завдання чи тесту, вводить дані, які відправляються на сервер, де перевіряються на коректність. Якщо дані не коректні, система повідомляє про це вчителя. Якщо дані коректні, система створює відповідне завдання/тест у системі. Після цього учень може увійти до системи, обрати певне завдання чи тест та ініціювати його виконання, ввести дані які відправляються на сервер й на основі яких

вираховується результат виконання, який можуть переглянути як учень так і вчитель.

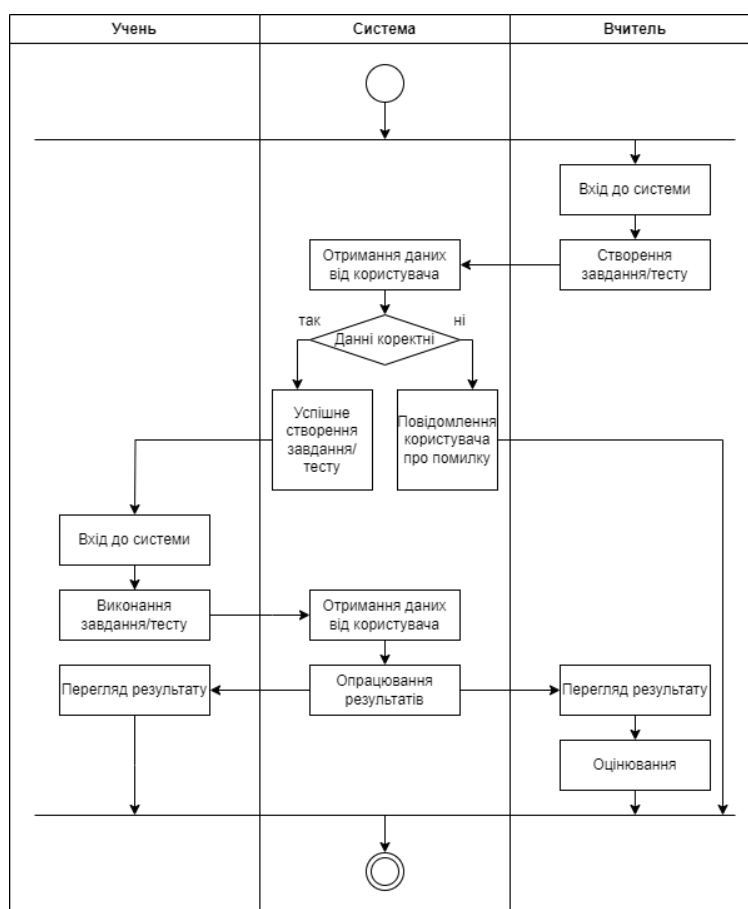


Рис. 2.17 Діаграма активності роботи з завданнями та тестами

Для створення завдання чи тесту, Вчитель повинен ввести коректні дані відносно цього завдання. Валідація цих даних в клієнтській частині системи виконується з використанням поєднання таких бібліотек як React Hook Form та Zod. React Hook Form – це популярна бібліотека для управління формами в React, яка надає простий і ефективний спосіб обробки форм та валідації в додатках [22]. Вона надає зручний спосіб побудови форм для введення даних, мінімізує кількість оновлення сторінки, дозволяє керувати станом форми локально в компоненті й надає можливість слідкувати та отримувати значення окремих полів форми. Zod в свою чергу – це бібліотека для валідації та синтаксичного аналізу даних яка забезпечує зручний і безпечний спосіб перевірки структур даних [23]. Zod має зручний синтаксис визначення схем форм, що полегшує їх побудову. Zod можна

використовувати як валідатор Reack Hook Form форм. Наведено приклади ініціалізації (рис. 2.18) та валідації форми (рис.2.19).

```
const form = useForm({
  resolver: zodResolver(taskSchema),
  defaultValues: {
    name: '',
    description: '',
    deadline: new Date(tomorrow.setHours(24, 0, 0, 0)),
    subjectQuery: '',
    subjectId: '',
    classQuery: '',
    classId: '',
  },
})
```

Рис. 2.18 Приклад ініціалізації форми

```
export const taskSchema = z.object({
  name: z.string().min(1, { message: 'Назва завдання не може бути пустою' }),
  description: z.string().trim().optional(),
  deadline: z.date().min(new Date(), { message: 'Дата не може бути в минулому' }),
  subjectId: z.number(),
  classId: z.number(),
})
```

Рис. 2.19 Приклад побудови схеми валідації форми

Після успішної валідації на клієнті, дані відправляються на сервер, де обробляються та відправляються у базу даних. Робота з базою даних виконується використовуючи інструменти Prisma ORM. Prisma ORM – це сучасний інструмент для управління базами даних, орієнтований на розробників, що використовують TypeScript та JavaScript. Він дозволяє взаємодіяти з базами даних у декларативний спосіб, забезпечуючи зручну роботу з SQL-запитами через TypeScript/JavaScript API. Наведено приклад створення запису в базі даних (рис. 2.20).

```

async createTask(req, res) {
  const { name, description, deadline, subjectId, classId } = req.body
  const task = await db.task.create({
    data: {
      name,
      description,
      deadline,
      subject: { connect: { id: subjectId } },
      class: { connect: { id: classId } },
    },
  })
  res.json(task)
}

```

Рис. 2.20 Створення запису завдання в базі даних

При виконанні завдання, після валідації введених даних, Учень відправляє запит на створення запису в таблиці TaskAnswer. Після успішного створення запису відповіді на завдання Учнем, Вчитель може переглянути відповідь Учня та оцінити її. Після появи оцінки, Учень може її побачити на сторінці завдання а Вчитель може її змінити на сторінці відповіді на завдання. Наведено частину кодової реалізації оновлення оцінки відповіді на завдання (рис. 2.21).

```

async updateTaskAnswerMark(req, res) {
  const { id } = req.params
  const { mark } = req.body
  const answer = await db.taskAnswer.update({
    where: { id: parseInt(id) },
    data: {
      mark,
    },
  })
}

```

Рис. 2.21 Оновлення оцінки відповіді на запитання

Поле для вводу тексту завдання Вчителем та поле для вводу тексту відповіді Учня підтримує форматування шляхом використання бібліотеки Tiptap [24]. Це сучасна, модульна, і розширювана бібліотека для створення rich-text (текст з форматуванням) редакторів на основі JavaScript. Бібліотека дозволяє встановлювати окремі розширення, що надають певний варіант форматування тексту. Підключені в додатку розширення надають можливість формувати текст як: заголовок, жирний текст, нахилений текст, перекреслений текст, символічний список та нумерований список.

Форматований текст відправляється на сервер та зберігається в базі даних як набір HTML тегів зі стилізацією, що потім відображається на сторінці. Перед тим як відправити HTML рядок на сервер, проводиться його чистка інструментами бібліотеки DOMPurify, що видаляє весь потенційно небажаний чи небезпечний код з рядку [25].

Додатково, поля для вводу підтримують вставку відео з YouTube та зображень з Imgur. Також підтримується синтаксис LaTeX [26] для якісного форматування математичних виразів. Цей функціонал реалізований шляхом пошуку патернів у тексті під час його відображення на сторінці. Знайдені патерни замінюються на відповідні елементи інтерфейсу: посилання на відео з YouTube замінюються на вбудований відео-iframe; посилання на зображення з Imgur замінюються на тег `<img>` з вказаним посиланням; формули, записані в синтаксисі LaTeX, перетворюються на HTML-код, стилізований за допомогою бібліотеки KaTeX [27]. Відображення HTML-коду на сторінці відбувається шляхом використання властивості `dangerouslySetInnerHTML`, яку надає React.js. Ця властивість дозволяє вставляти розмічений HTML-код у компонент React. Наведено алгоритм перетворення патернів на відповідні елементи інтерфейсу (рис. 2.22). Функція `renderLatex` перетворює рядок в синтаксисі LaTeX у HTML за допомогою методу `katex.renderToString`. Функція `convertLinksToEmbeds` визначає регулярні вирази для пошуку посилань на відео YouTube, зображень Imgur та тексту позначеного як текст в синтаксисі Latex та замінює знайдені об'єкти на відповідні елементи інтерфейсу: вбудовані iframe-відео YouTube, вбудовані зображення Imgur та HTML код за допомогою функції `renderLatex` відповідно. Функція `createMarkup` повертає React-сумісний об'єкт розмітки з властивістю `__html`, встановленою у значення HTML-рядка.

```

function renderLatex(latex) {
  try {
    return katex.renderToString(latex, {
      throwOnError: false,
      displayMode: latex.startsWith('$') || latex.startsWith('\\[',
    })
  } catch (e) {
    console.error('Failed to render LaTeX:', latex, e)
    return latex
  }
}

function convertLinksToEmbeds(text) {
  const youtubeRegExp = /(?:https?:\/\/)?(?:www\.)?(?:youtu\.be\/|youtube\.com\/(?:watc
  let newText = text.replace(
    youtubeRegExp,

'<div class="rounded-md" style="position: relative; padding-bottom: 56.25%; height: 0; o
)

const imgurRegExp = /(?:https?:\/\/)?(?:www\.)?(?:i\.imgur\.com\/)([a-zA-Z0-9\-\_])\
newText = newText.replace(imgurRegExp, '<img class="rounded-lg mt-2 mb-2 w-1/3" src='

// Inline LaTeX: '$...$' or '\( ... \)'
const inlineLatexRegExp = /\$(.*?)\$\|\\\((.*?)\\\)/g
newText = newText.replace(inlineLatexRegExp, (_, p1, p2) => renderLatex(p1 || p2))

// Block LaTeX: '$$...$$' or '\[ ... \]'
const blockLatexRegExp = /\$\$(.*?)\$\$|\\\[(.*?)\\\]/gs
newText = newText.replace(blockLatexRegExp, (_, p1, p2) => '<div class="katex-block":

return newText
}

export const createMarkup = (htmlContent) => {
  const htmlWithEmbeds = convertLinksToEmbeds(htmlContent)
  return { __html: htmlWithEmbeds }
}

```

Рис. 2.22 Алгоритм перетворення патернів на елементи інтерфейсу

Кожна відповідь на завдання має секцію коментарів, де Учень, Вчитель та Батьки можуть спілкуватися в реальному часі. Система коментарів реалізована за допомогою бібліотеки Socket.io [28]. Це популярна бібліотека для двостороннього зв'язку між вебклієнтами та серверами в реальному часі. Для зв'язку Socket.io використовує технологію WebSockets як основну, а WebTransport і HTTP long-polling – як резервні варіанти, якщо WebSockets не працює або не підтримується. Для забезпечення зв'язку між користувачами для кожної відповіді на завдання створюється окрема кімната зв'язку, в якій відбувається обмін повідомленнями між клієнтами, що під'єдналися до неї. Кімната створюється, коли користувач переходить на сторінку результату виконання завдання, що спричиняє відображення компоненту коментарів. При відправленні коментарю на сервер, повідомлення зберігається в базі даних та надсилається всім іншим клієнтам, під'єднаним до кімнати зв'язку.

Наведено приклади налаштування та реалізації зв'язку за допомогою Socket.io. Серверний код (рис. 2.23) чекає на вхідні з'єднання від клієнтів, на події `joinRoom`, `newComment` та `disconnect`, які видає клієнт. При події `joinRoom`, приєднує клієнта до кімнати з вказаним `taskAnswerId`. При події `newComment`, деструктурує об'єкт коментаря для вилучення властивостей `taskAnswerId`, `text` та `author`, створює новий коментар в базі даних за допомогою функції `createTaskAnswerComment` та транслює новий коментар усім клієнтам у кімнаті з вказаним `taskAnswerId` за допомогою методу `emit`. Клієнтський код (рис. 2.24) отримує коментарі для вказаного `taskAnswerId` та оновлює стан коментарів отриманими даними. Далі відсилає серверу подію `joinRoom`, приєднуючи клієнта до кімнати з вказаним `taskAnswerId` та чекає на подію `comment`. При отриманні нового коментаря оновлює стан коментарів, додаючи новий коментар до масиву коментарів. Функція `handleCommentSubmit` створює новий об'єкт коментаря з властивостями `taskAnswerId`, `text` та `author` та відправляє подію `newComment` на сервер, надсилаючи новий коментар.

```

io.on('connection', (socket) => {
  socket.on('joinRoom', (taskAnswerId) => {
    socket.join(taskAnswerId)
    console.log(`Socket ${socket.id} joined room ${taskAnswerId}`)
  })

  socket.on('newComment', async (comment) => {
    const { taskAnswerId, text, author } = comment

    try {
      const createdComment = await createTaskAnswerComment({
        text,
        taskAnswerId,
        authorId: author.id,
      })

      console.log(`Broadcasting new comment to room ${taskAnswerId}:`, createdComment)
      io.to(taskAnswerId).emit('comment', createdComment)
    } catch (error) {
      socket.emit('error', 'Error saving comment')
    }
  })

  socket.on('disconnect', () => {
    console.log('Client disconnected:', socket.id)
  })
})

```

Рис. 2.23 Логіка зв'язку у серверній частині системи

повинен ввести текст, який буде відповідати тексту хоча б одного варіанту відповіді, прив'язаного до цього питання та позначеного як правильний.

Учень, обравши тест зі списку тестів свого класу, може приступити до його виконання. При першому виборі питання зі списку питань, на сервер буде відправлено запит на створення запису в таблиці TestResult, що буде зберігати результат виконання цього тесту. При відповіді на питання, Учень може: обрати один варіант відповіді, якщо питання передбачає один правильний варіант; обрати декілька варіантів відповіді, якщо питання передбачає декілька правильних варіантів; ввести відповідь вручну, якщо питання це передбачає. Після надання відповіді, Учень повинен її закріпити, що відправить запит на сервер для створення записів у таблиці StudentTaskAnswer. Для кожного обраного варіанту відповіді буде створено окремий запис, а для питання з ручною відповіддю в записі буде записано текст відповіді Учня. Давши відповідь на бажану кількість питань, Учень може завершити виконання тесту, що відправить запит на сервер та ініціює підрахунок оцінки за тест. Алгоритм підрахунку порівнює кожен відповідь Учня з варіантами відповідей на відповідне питання та записує отриману оцінку у відповідний запис результату тесту в базі даних. Наведено частину алгоритму підрахунку результату (рис. 2.25). Цей алгоритм отримує результат виконання тесту за вказаним ідентифікатором, включаючи пов'язані з результатом відповіді Учня, які в свою чергу включають пов'язаний з відповіддю варіант відповіді на запитання тесту та саме запитання з його варіантами відповіді. Алгоритм перебирає відповіді, пов'язані з результатом тесту. Для кожної відповіді перевіряється, чи є в запитанні правильні чи взагалі будь-які відповіді. Якщо правильних відповідей на запитання немає, це запитання не буде враховано в оцінку. Якщо питання введене вручну, перевіряється, чи збігається відповідь учня, введена вручну, з будь-якою з правильних відповідей. Якщо питання не ручне, перевіряється, чи збігаються відповіді учня з правильними відповідями. Відповідно оновлюються змінні correctAnswers і

totalConsideredAnswers. Оцінка обчислюється шляхом ділення правильних відповідей на загальну кількість розглянутих відповідей, або встановлює її рівною 0, якщо жодна відповідь не була розглянута. Після підрахунку оцінки оновлює результат тесту з розрахованою оцінкою та часом завершення. Після запису результату в базу даних, Вчитель може переглянути результат виконання тесту Учень. Він зможе побачити кожну відповідь, яку дав Учень, та підрахований результат.

```
const completedAt = new Date()

let totalConsideredAnswers = 0
let correctAnswers = 0

testResult.answers.forEach((studentAnswer) => {
  const question = studentAnswer.question

  const correctTestAnswers = question.testAnswers.filter((testAnswer) => testAnswer.isCorrect)
  const hasCorrectAnswers = correctTestAnswers.length > 0
  const hasAnyAnswers = question.testAnswers.length > 0

  if (!hasAnyAnswers || !hasCorrectAnswers) {
    return
  }

  if (question.isManual) {
    const correctManualAnswers = correctTestAnswers.map((testAnswer) => testAnswer.text.toLowerCase().trim())

    if (correctManualAnswers.includes(studentAnswer.manualAnswer?.toLowerCase().trim())) {
      correctAnswers++
    }
  } else {
    const studentAnswersForQuestion = testResult.answers.filter((ans) => ans.questionId === question.id)

    const correctStudentAnswers = studentAnswersForQuestion.filter(
      (ans) => ans.answer && correctTestAnswers.some((correctAnswer) => correctAnswer.id === ans.answer.id)
    )

    if (correctStudentAnswers.length === correctTestAnswers.length) {
      correctAnswers++
    } else {
      correctAnswers += correctStudentAnswers.length / correctTestAnswers.length
    }
  }

  totalConsideredAnswers++
})

const mark = totalConsideredAnswers > 0 ? correctAnswers / totalConsideredAnswers : 0

const result = await db.testResult.update({
  where: { id: testResultId },
  data: {
    completedAt,
    mark,
  },
})
```

Рис. 2.25 Частина алгоритму підрахунку оцінки за тест

На основі даного алгоритму було побудовано діаграму станів виконання тесту (рис. 2.26).

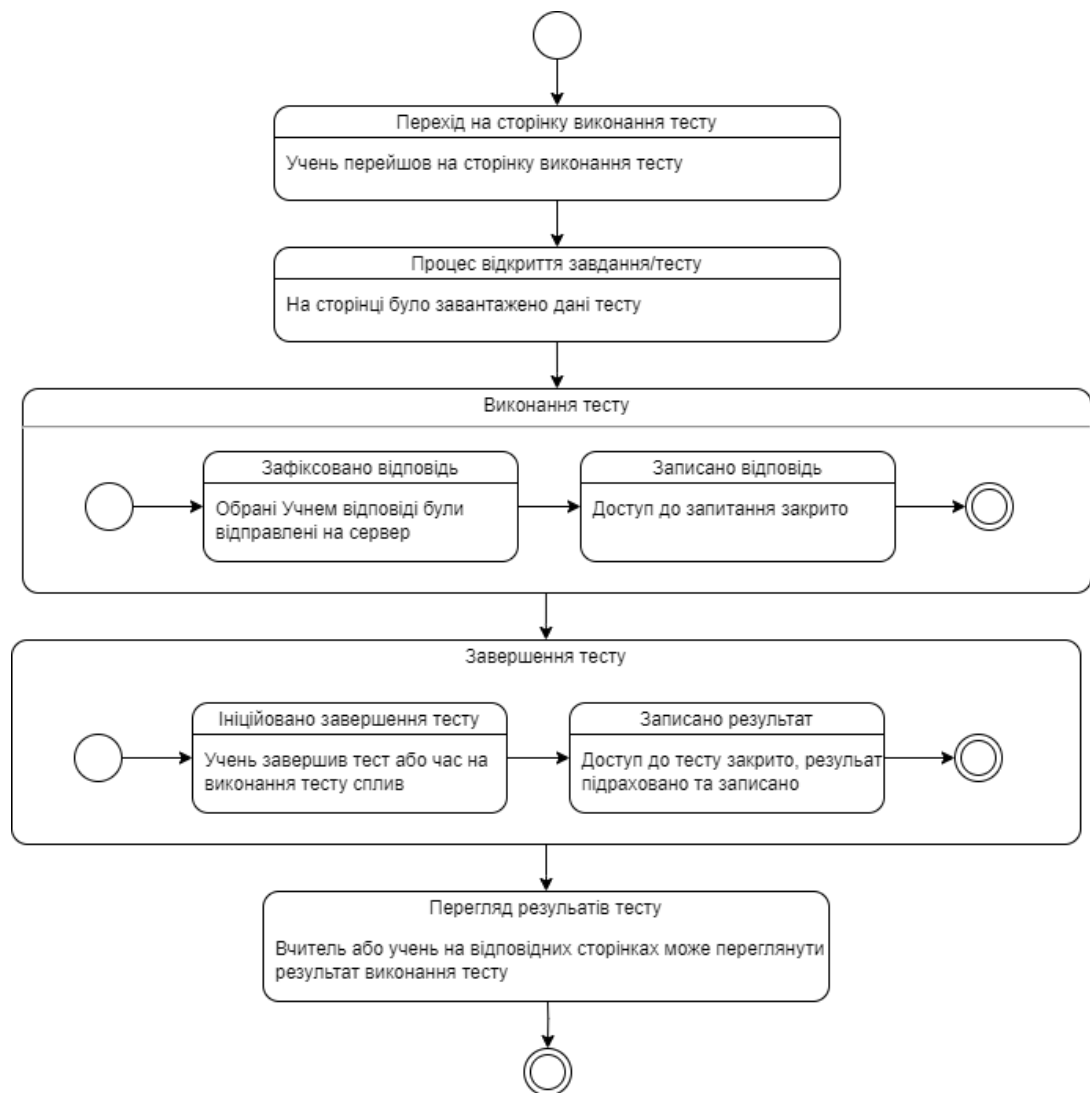


Рис. 2.26 Діаграма стану виконання тесту

Як було зазначено, в процесі виконання тесту Учнем, у випадку якщо час на виконання тесту сплине, результати будуть автоматично відправлені на сервер та доступ до тесту буде закрито. Наведено кодову реалізацію відліку часу (рис. 2.27). Алгоритм виконується одразу після того, як було створено запис про результат виконання тесту Учнем в базі даних. Спочатку алгоритм вираховує час, коли тест має бути завершений, який дорівнює часу `createdAt` плюс `timeLimit`. Далі вираховується `initialTimeRemaining` – початковий час, що залишився на виконання тесту який обчислюється шляхом віднімання поточного часу від кінцевого. Цей час вираховується на випадок, якщо користувач при заході на тест вже мав запис результату виконання тесту, проте ще не закінчив його. Це можливо, оскільки всі данні про виконання тесту зберігаються в базі даних, що дозволяє продовжити

виконання тесту без втрати даних при збої чи випадковому закритті додатку. Якщо `initialTimeRemaining` більше 0, це означає, що тест ще триває. Встановлюється стан `timeRemaining` на значення `initialTimeRemaining`. Далі алгоритм створює інтервал за допомогою функції `setInterval` що надає Node.js, який оновлюватиме стан `timeRemaining` кожні 1,000 мілісекунд (1 секунду). У виклику інтервалу віднімається 1,000 мілісекунд від попереднього значення `timeRemaining`. Якщо нове значення дорівнює 0 або менше, інтервал очищується, а тест завершується викликанням функції `completeTestResult`. Якщо `initialTimeRemaining` дорівнює 0 або менше, це означає, що час тесту закінчився. В такому випадку тест також завершиться.

```

if (testResult && testResult.createdAt && !testResult.completedAt) {
  const createdAt = new Date(testResult.createdAt)
  const timeLimit = test.timeLimit * 60 * 1000
  const now = new Date()
  const endTime = addMilliseconds(createdAt, timeLimit)
  const initialTimeRemaining = differenceInMilliseconds(endTime, now)

  if (initialTimeRemaining > 0) {
    setTimeRemaining(initialTimeRemaining)

    const intervalId = setInterval(() => {
      setTimeRemaining((prevTime) => {
        const newTime = prevTime - 1000
        if (newTime ≤ 0) {
          clearInterval(intervalId)
          completeTestResult(testResult.id).then((data) => {
            setTestResult(data)
            toast.success('Час тестування завершено, результат збережено.')
          })
        }
        return newTime
      })
    }, 1000)

    return () => clearInterval(intervalId)
  } else {
    completeTestResult(testResult.id).then((data) => {
      setTestResult(data)
      toast.success('Час тестування завершено, результат збережено.')
    })
  }
}

```

Рис. 2.27 Алгоритм відліку часу

До завдань, відповідей до завдань та запитань тесту, користувачі мають змогу прикріпляти файли зображень. Для керування файлами використовується сервіс ImageKit [29]. Це хмарний сервіс для обробки та оптимізації зображень, який допомагає розробникам ефективно керувати,

зберігати і доставляти зображення у веб- і мобільних додатках. Він пропонує набір інструментів для автоматичної оптимізації зображень, трансформації, управління CDN (Content Delivery Network), і кешування, що спрощує роботу з мультимедійними файлами. Завантаження зображення на сховище ImageKit відбувається за допомогою компоненту IKUpload. Наведено приклад використання компоненту (рис. 2.28). Даний компонент виступає діалогом завантаження файлу. При успішному завантаженні файлу, він буде збережений у вказаній директорії з вказаним ім'ям, а на сервер буде відправлено запит на оновлення поля filesCount. Для завантаження файлів на сервер ImageKit, в серверній частині додатку було налаштовано API аутентифікації при завантаженні файлу, а в клієнтській частині було реалізовано функцію authenticator, що звертається до цього API. Для роботи компонентів ImageKit, додаток потрібно загорнути в компонент IKContext, що містить в собі конфігурацію доступу до файлів.

```
<IKUpload
  fileName={task.id + '_' + (parseInt(task.filesCount) + 1)}
  folder={'/task'}
  validateFile={(file) => file.size < 20000000}
  useUniqueFileName={false}
  style={{ display: 'none' }}
  ref={ikUploadRef}
  onError={handleUploadError}
  onSuccess={handleUploadSuccess}
/>
```

Рис. 2.28 Використання компоненту IKUpload

Завдяки встановленим при завантаженні назвам зображень, їх можна відображати на сторінці в залежності від обраного завдання, відповіді на завдання чи питання до тесту. Відображення відбувається за допомогою компонента Image від Next.js. Для того, щоб цей компонент підтримував параметри ImageKit, в компонент передається imageKitLoader – функція, що генерує URL-адресу для завантаження зображення з ImageKit. Вона додає до URL параметри, що має приймає компонент Image. Наведено кодову реалізацію функції imageKitLoader (рис. 2.29).

```

export const imageKitLoader = ({ src, width, quality }) => {
  if (src[0] === '/') src = src.slice(1)
  const params = ['w-${width}']
  if (quality) {
    params.push(`q-${quality}`)
  }
  const paramsString = params.join(',')
  var urlEndpoint = process.env.NEXT_PUBLIC_IMAGEKIT_URL_ENDPOINT
  if (urlEndpoint[urlEndpoint.length - 1] === '/') urlEndpoint = urlEndpoint.substring(0, urlEndpoint.length - 1)
  return `${urlEndpoint}/${src}?tr=${paramsString}`
}

```

Рис. 2.29 Функція imageKitLoader

Користувачі з роллю Батьки, мають змогу переглядати результати виконання завдань та тестів Учня, що прив'язаний до його облікового запису. Перегляд результатів виглядає ідентично тому, до якого мають доступ Вчителі, проте Батьки не мають змоги змінювати оцінку, дану завданням. Також Батьки можуть переглянути лаконічну статистику успішності Учня, що містить кількість відповідей на завдання та тести а також середні оцінки за завдання та тести.

Висновки до другого розділу

Під час проектування системи виконання завдань та тестів для шкільного закладу освіти було визначено сценарії взаємодії з системою та варіанти її використання. Результатом є спроектована модель системи з чітко визначеною взаємодією між її компонентами.

Основними цілями проекту було підвищити ефективність взаємодії учасників шкільного навчального процесу та надати зручний спосіб виконання завдань і тестів, що має значно пришвидшити цей процес порівняно з традиційним підходом.

Було спроектовано структуру бази даних, яка забезпечує ефективне зберігання та обробку даних про користувачів системи, завдання, тести та пов'язані з ними дані, дані про класи та навчальні предмети. На основі розроблених сутностей були створені таблиці в базі даних, а також діаграма бази даних для їх візуалізації.

Основні етапи проектування та реалізації системи включали аналіз функціональних вимог, розробку логіки взаємодії компонентів системи, розробку алгоритмів роботи з системою, що охоплює: алгоритми роботи з даними системи, алгоритми авторизації та реєстрації, алгоритми створення та виконання завдань, алгоритми створення, проходження та оцінювання тестів, алгоритм форматування та стилізації тексту, алгоритм завантаження та отримання зображень.

Було розглянуто та детально описано алгоритми та методи, використані при реалізації системи, що забезпечують її стабільну та надійну роботу.

Спроектowana система забезпечує зручний спосіб взаємодії між учасниками навчального процесу, ефективно створення та виконання завдань і тестів, а також надійне зберігання та обробку даних, пов'язаних з навчальним процесом.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ ЗДАЧІ ЗАВДАНЬ ТА ТЕСТІВ ДЛЯ ШКІЛЬНОГО ЗАКЛАДУ ОСВІТИ

3.1 Порядок встановлення та налаштування параметрів системи

Система виконання завдань та тестів для шкільного закладу освіти являє собою набір компонентів, коректною взаємодією між якими досягається надійна та стабільна її робота. Їх встановлення та налаштування вимагає розгортання кожного елементу. Було побудовано діаграму розгортання компонентів системи (рис. 3.1), що показує алгоритм взаємодії компонентів системи між собою.

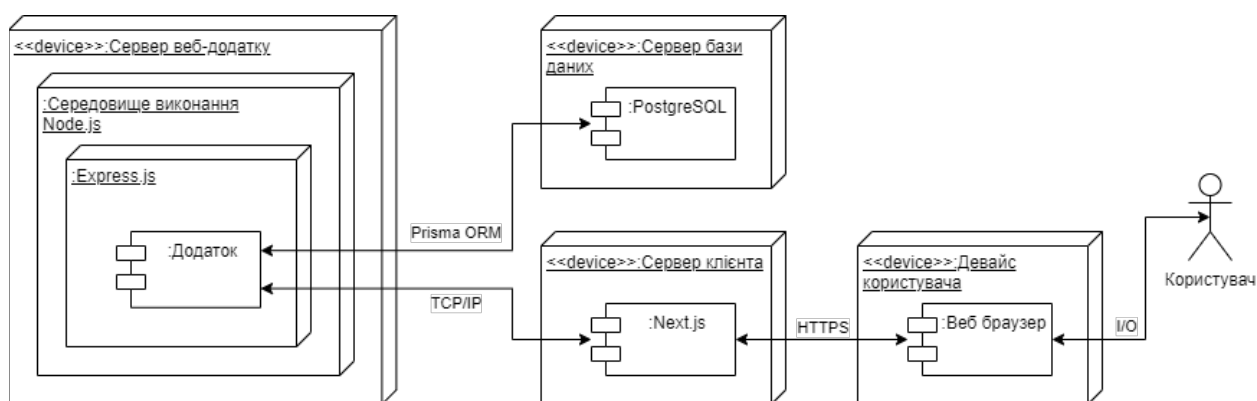


Рис. 3.1 Діаграма розгортання системи

Сервер вебдодатку – сервер, на якому розміщується серверна частина веб додатку, що керує логікою доступу до даних. Він включає в себе:

1. Node.js: Середовище виконання для JavaScript на стороні сервера.
2. Express.js: Фреймворк для Node.js, який спрощує розробку вебдодатків і API. Він надає інструменти для управління маршрутизацією, обробки запитів, відповіді на них та інших серверних функцій.
3. Додаток: Власне серверний додаток, який написаний за допомогою Express.js і виконує бізнес-логіку вебдодатку. Він обробляє HTTP-запити від клієнтів, виконує операції з базою даних та відповідає на ці запити.

Сервер бази даних – сервер на якому розміщується реляційна база даних PostgreSQL.

Зв'язок між сервером вебдодатку та базою даних реалізується інструментами Prisma ORM та вбудованих в нього драйверів баз даних, що комунікують з базою даних за допомогою протоколу TCP.

Сервер клієнта – сервер, на якому розміщена клієнтська частина додатку, побудована за допомогою Next.js – фреймворку для створення React-додатків.

Сервер клієнта – це сервер, на якому розміщується клієнтський додаток React.js. Він включає в себе клієнт React.js – фронтенд-додаток. Він надсилає запити до сервера та оновлює інтерфейс на основі отриманих відповідей. Клієнт React.js взаємодіє з додатком на вебсервері за допомогою TCP/IP (Transmission Control Protocol/Internet Protocol).

Комунікація між сервером клієнта та сервером вебдодатку відбувається за допомогою протоколу TCP/IP.

Пристрій користувача – на пристрої користувача розміщується веббраузер, що відправляє HTTP(S)-запити до сервера клієнта та отримує відповіді у вигляді вебсторінок.

Користувач взаємодіє зі своїм девайсом за допомогою пристроїв вводу-виводу (I/O).

Даний підхід до побудови структури системи забезпечує ефективну, масштабовану і безпечну роботу всього додатку.

3.2 Структура інтерфейсу вебдодатку та порядок роботи

При переході на головну сторінку вебдодатку користувач бачить привітання та прохання увійти до свого облікового запису (рис. 3.2).

Вітаємо!

Щоб продовжити роботу, будь-ласка увійдіть.

Вхід

Рис. 3.2 Головна сторінка додатку

Після переходу на сторінку входу, користувач може бачити форму входу з валідацією (рис. 3.3).

Вхід

Введіть ваш email та пароль нижче, для входу.

Email

exarpmle

Невірний формат email

Пароль

.....

Пароль повинен містити мінімум 8 символів

Увійти

Рис. 3.3 Форма входу

Після успішної авторизації, користувач перенаправлений на сторінку /workspace (рис. 3.4). Ця сторінка містить посилання на різні компоненти додатку, що відповідають переліку ролей користувачів системи. Набір посилань залежить від ролей, що має користувач. Якщо користувач має лише одну роль, зі сторінки /workspace його буде автоматично перенаправлено на сторінку відповідної ролі

Оберіть роль

Оберіть до робочої зони якої з ваших ролей ви перейдете

Адмін Учень Вчитель Батько/мати

Рис. 3.4 Компонент вибору сторінки ролі

Сторінка Адміністратора має три вкладки: Класи, Предмети та Користувачі.

На сторінці класів відображено перелік класів, створених в системі, у вигляді акордеону, для компактного та зручного розміщення та кнопка для відображення форми створення нового класу (рис. 3.5).

Класи

1 класи

2 класи

3 класи

3-А

3-Б

3-В

4 класи

5 класи

Рис. 3.5 Список класів, створених в системі

Натиснувши на певний клас, Адміністратор перейде до сторінки цього класу, де зможе побачити список учнів, що пов'язані з цим класом та кнопку для відображення форми редагування класу (рис. 3.6).

3-А

Дмитро Дмитров Дмитрович

Сергій Сергієв Сергійович

Владислав Влад Владиславович

Рис. 3.6 Сторінка інформації про клас та його учнів

Форми створення та редагування класу містять одне поле для введення назви та кнопку збереження.

Наступною сторінкою, доступною Адміністратору є сторінка предметів (рис. 3.7). Перейшовши на дану сторінку, користувач побачить список предметів, створених в системі, разом з кнопками для відображення форми їх редагування, а також кнопку для відображення форми створення нового предмету.

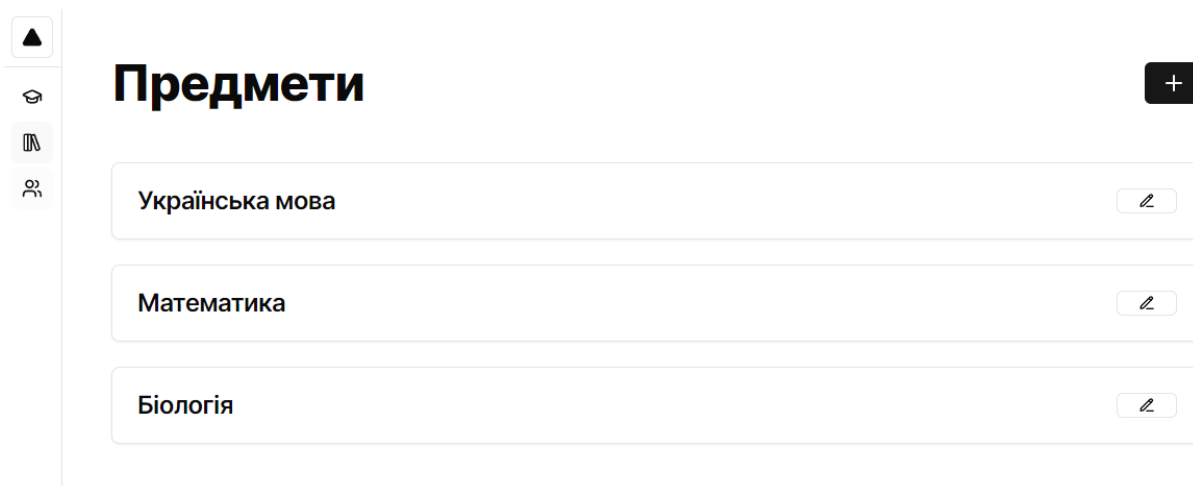


Рис. 3.7 Сторінка реєстрації облікового запису

Форми створення та редагування предмету, подібним до класів чином, мають одне поле для введення назви та кнопку збереження.

Останньою сторінкою, що пов'язана з роллю Адміністратора, є сторінка керування користувачами (рис. 3.8). На даній сторінці користувач може побачити список користувачів, розділений на чотири вкладки за ролями. В кожній із вкладок можна виконувати пошук за ім'ям користувача, що буде шукати введений текст незалежно в ім'ї, прізвищі та по-батькові користувачів, для надання найточніших результатів.

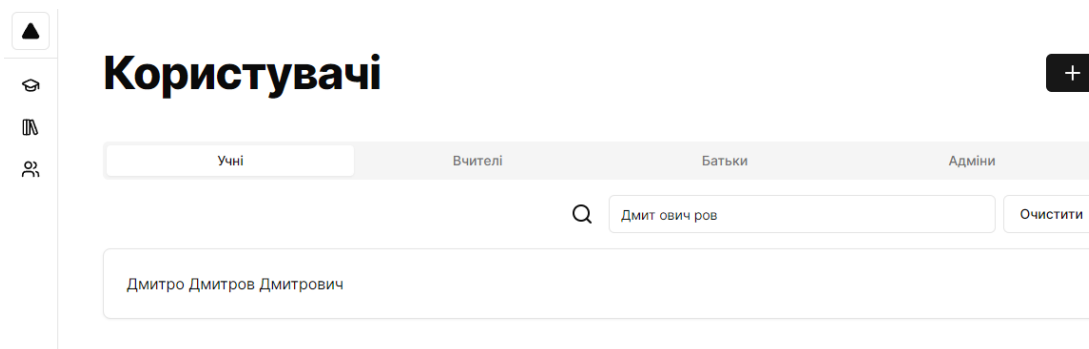


Рис. 3.8 Приклад використання пошуку на сторінці користувачів

×

Додати користувача

Введіть дані користувача та натисніть зберегти

Ім'я

Іван

Прізвище

Іванов

По-батькові

Іванович

Ролі

Оберіть які ролі матиме користувач

☒ Учень

☐ Вчитель

☐ Батьки

☐ Адміністратор

Клас

Оберіть до якого класу буде відноситись учень

Обраний клас: 3-А

Батько/Мати

Оберіть користувача, що буде вважатись батьком/матю учня та зможе переглядати його успішність

Обраний користувач: Name Surname Patronymic

Зберегти

✕

Додати користувача

Введіть дані користувача та натисніть зберегти

Ім'я

Іван

Ім'я обов'язкове

Прізвище

Іванов

Прізвище обов'язкове

По-батькові

Іванович

По-батькові обов'язкове

Ролі

Оберіть які ролі матиме користувач

☐ Учень

☐ Вчитель

☐ Батьки

☐ Адміністратор

Користувач повинен мати хоча б одну роль

Зберегти

					ИПЗ.КР.Б-121-24-ПЗ	Арк.
						66

При натисканні на певного користувача, Адміністратор перейде на сторінку цього користувача, на якій можна побачити інформацію про нього, його ролі, пов'язаний з ним клас та батьківський обліковий запис. Також на сторінці користувача присутні кнопки для відображення форми редагування користування, видалення даних входу користувача для його повторної реєстрації та кнопка, що скопіює в буфер обміну посилання для реєстрації цього користувача, якщо цей користувач не має даних входу. Профіль будується в залежності від наявних даних про користувача в системі. Якщо користувач ще не зареєстрований в системі, в профілі буде відображатись кнопка для копіювання посилання на реєстрацію (рис. 3.11). Якщо користувач зареєстрований, замість кнопки буде відображено email адресу користувача, за якою відбувалась реєстрація (рис. 3.12).

Форма редагування користувача має таких же набір полів, як і форма його створення, проте має відповідну назву та автоматично заповнюється даними про обраного для редагування користувача.

При натисканні на кнопку скидування email та паролю, додаток відобразить попередження, на якому можна буде обрати опцію відміни дії, або її підтвердження.

Дмитро Дмитров Дмитрович

Скопіювати посилання для реєстрації

учень

3-А
Клас

Name Surname Patronymic
Батько/мати

Рис. 3.11 Профіль незареєстрованого Учня

Name Surname Patronymic

example@email.com

адмін

учень

вчитель

батько/мати

Рис. 3.12 Профіль зареєстрованого користувача

При переході за посиланням для реєстрації, що може отримати Адміністратор у профілі незареєстрованого користувача, користувача зустріне форма реєстрації (рис. 3.13). Ця форма є логічно схожою до вже описаної форми входу, проте має додаткове поле для повтору паролю та відображає повне ім'я облікового запису реєстрація якого відбувається. При успішній реєстрації, користувач буде перенаправлений на сторінку входу, зможе ввести створені ним дані входу.

Реєстрація

Введіть ваш email та пароль нижче, для завершення реєстрації користувача Дмитро Дмитров Дмитрович

Email

example@email.com

Пароль

Повтор паролю

Зареєструватись

Рис. 3.13 Форма реєстрації користувача

Наступною частиною додатку є сторінки, пов'язані з роллю Вчитель. Для Вчителів доступні сторінки створення та перегляду завдань й створення та перегляду тестів.

На сторінці завдань, користувач може побачити список створених завдань в системі (рис. 3.14). Список можна сортувати за назвою, датою та часом створення та дедлайном. Доступна фільтрація за класом та предметом. Також можливий пошук завдань за назвою. Наявна можливість приховати або відобразити завдання, дедлайн яких сплив.

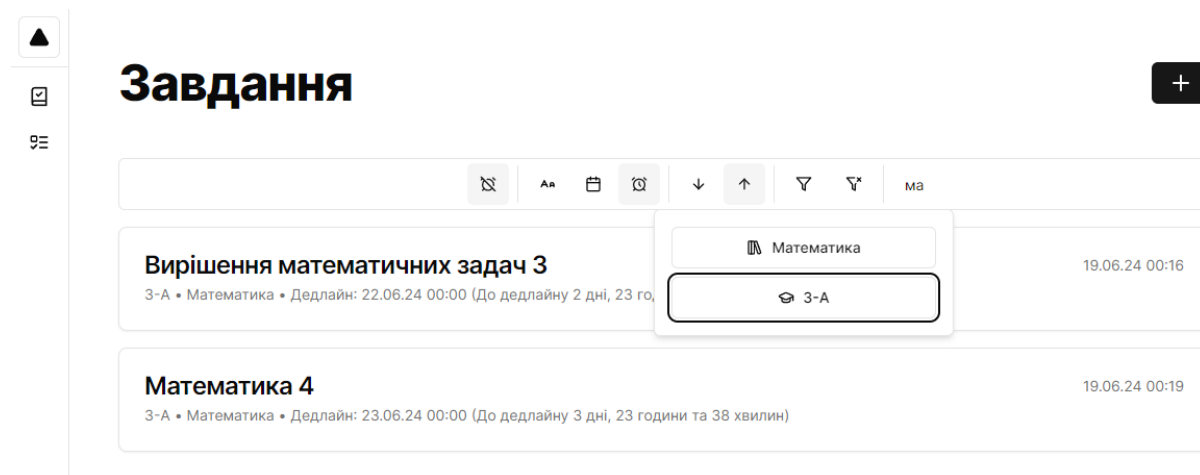


Рис. 3.14 Список завдань

Натиснувши на кнопку, що відображає форму створення завдання (рис. 3.15), користувач матиме змогу ввести назву завдання, його опис, що підтримує форматування тексту, обрати дедлайн завдання за допомогою зручного календаря (рис. 3.16) та обрати клас та предмет, до якого відноситиметься завдання. Перед відправленням на сервер, форма валідується. Всі модальні вікна в додатку для вибору класу, предмету чи користувача у формі мають поле динамічного пошуку та секцію результатів пошуку, для зручного та швидкого вибору потрібних даних (рис. 3.17).

Додати завдання

Введіть дані завдання та натисніть зберегти

Назва

Вирішення математичних задач 3

Опис

Н1BIS≡≡

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse scelerisque lacinia tincidunt. Nam condimentum lorem id sem dictum efficitur. Vivamus lobortis ac eros id posuere. Etiam pellentesque tristique orci id sodales. Suspendisse potenti. Phasellus nec rutrum augue. Aenean sollicitudin ultricies enim, non molestie eros pulvinar non. In hac habitasse platea dictumst. Suspendisse nec sapien a ante imperdiet ultricies id et risus. Duis laculis arcu in libero sodales, ac sodales nisl dictum. ~~Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.~~ Vivamus id lacinia risus. Nullam at aliquet justo. In quis elit augue. Suspendisse tempus tempus orci maximus faucibus. Phasellus ac mattis turpis, eu fringilla mauris.

Формули:

$E = mc^2$

$3_2 * 10 = x^2, 6\log_2(30)$

<https://www.youtube.com/watch?v=szqlTSx8QeQ>

Дедлайн

22-е червня 2024 р. 00:00:00

Клас

Оберіть клас до якого буде відноситись завдання

Обраний клас: 3-А

Предмет

Оберіть предмет до якого буде відноситись завдання

Обраний предмет: Математика

Зберегти

Рис. 3.15 Форма створення завдання

Дедлайн

21-е червня 2024 р. 00:00:00

<

червень 2024

>

пн

вт

ср

чт

пт

сб

нд

27

28

29

30

31

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

год.

хв.

сек.

00

00

00

🕒

Рис. 3.16 Календар для вибору дати та часу

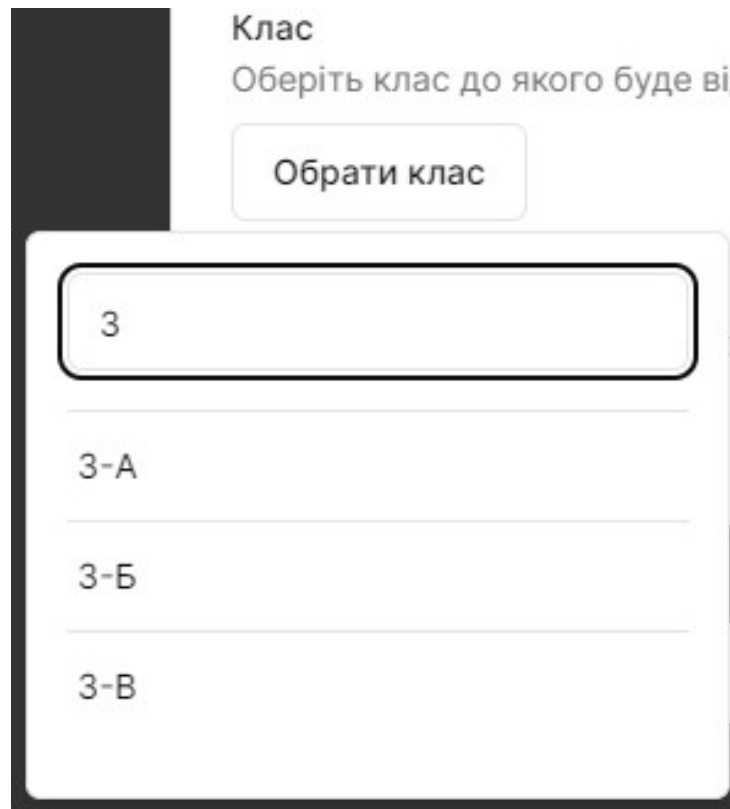


Рис. 3.17 Спливаюче вікно вибору класу

Натиснувши на завдання в списку, користувач перейде до сторінки перегляду цього завдання (рис. 3.19), де зможе переглянути інформацію про це завдання, його форматований текст, прикріплені зображення та відповіді учнів (рис. 3.18). Також на сторінці завдання присутній перемикач для зміни стану прихованості завдання. Форма редагування є дуже схожою до форми створення завдання, проте автоматично завантажує данні про завдання у форму. Присутня кнопка видалення завдання. Зображення відображаються у вигляді каруселі. Кожне із зображень може бути відкрите у повний розмір шляхом натискання на нього.

Відповіді учнів:

Дмитро Дмитров Дмитрович	19.06.24 00:54
☒ Не оцінено	

Рис. 3.18 Відповіді учнів на завдання

Вирішення математичних задач 3

3-А • Математика • Дедлайн: 22.06.24 00:00 (До дедлайну 1 день, 0 годин та 52 хвилини)

Приховане: ☐

19.06.24 00:16

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse scelerisque lacinia tincidunt. Nam condimentum lorem id sem dictum efficitur. Vivamus lobortis ac eros id posuere. Etiam pellentesque tristique orci id sodales. Suspendisse potenti. Phasellus nec rutrum augue. Aenean sollicitudin ultricies enim, non molestie eros pulvinar non. In hac habitasse platea dictumst. Suspendisse nec sapien a ante imperdiet ultricies id et risus. Duis iaculis arcu in libero sodales, ac sodales nisl dictum. ~~Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.~~ Vivamus id lacinia risus. Nullam at aliquet justo. In quis elit augue. Suspendisse tempus tempus orci maximus faucibus. Phasellus ac mattis turpis, eu fringilla mauris.

Формули:

$$E = mc^2$$

$$3_2 * 10 = x^2, 6\log_2(30)$$

 Лекція. Асинхронні запити на сервер та їх обробка

Старий підхід

Share

POST-запит з клієнта за допомогою JavaScript

```
var xhr = new XMLHttpRequest();
xhr.open("POST", 'your-endpoint-url', true);
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onreadystatechange = function () {
  if (xhr.readyState === 4 & xhr.status === 200) {
    console.log(xhr.responseText);
  }
};
var data = JSON.stringify({"key": "value"});
xhr.send(data);
```

Watch on  YouTube

Слайд 2

Зображення



Рис. 3.19 Сторінка завдання

При натисканні на відповідь Учня, Вчитель перейде на сторінку перегляду відповіді (рис. 3.20). На ній він зможе переглянути текст відповіді Учня, вказати або змінити оцінку, переглянути зображення, що він завантажив, та залишити коментар.

Відповідь учня Дмитро Дмитров Дмитрович

Оцінка: 6 / 12

Відповідь на завдання

Зображення завантажені учнем



Коментарі

ДД

Дмитро Дмитров Дмитрович

коментар 1

19.06.24 00:55

NS

Name Surname Patronymic

коментар 2

19.06.24 01:23

Ваш коментар...

Рис. 3.20 Сторінка перегляду відповіді учня

На сторінці тестів Вчитель може побачити список створених в системі тестів. Функціонал фільтрації, сортування та пошуку є ідентичним до списку завдань.

Натиснувши на кнопку, що відображає форму створення тесту (рис. 3.21), Вчитель зможе ввести назву тесту, обмеження тесту по часу, дедлайн, обрати клас та предмет, які будуть прив'язані до тесту.

Додати тест

Введіть дані тесту та натисніть зберегти

Назва

Тест 2

Обмеження по часу (в хвилинах)

20

Дедлайн

21-е червня 2024 р. 00:00:00

Клас

Оберіть клас до якого буде відноситись завдання

Обраний клас: 3-А

Предмет

Оберіть предмет до якого буде відноситись завдання

Обраний предмет: Математика

Зберегти

Рис. 3.21 Форма створення тесту

Натиснувши на тест в списку, користувач перейде до сторінки перегляду та конструювання тесту (рис. 3.22). На цій сторінці, він зможе додати запитання тесту за допомогою форми з полем для вводу форматowanego тексту та перемикачем, що символізує чи запитання передбачає ручний ввід відповіді та додати варіанти відповіді до запитань.

Рис. 3.22 Сторінка тесту

Форма редагування тесту, яка відкривається при кліку на кнопку редагування є дуже схожою на форму створення тесту, проте вона одразу вводить дані про тест у відповідні поля. Присутня кнопка видалення тесту. Кожне питання можна редагувати, за допомогою схожої до форми створення питання форми, та видалити. Під кожним завданням можна створювати варіанти відповіді, ввівши дані у відповідну форму та обравши чи даний варіант буде вважатись правильним. Кожен створений варіант відповіді можна видалити.

Під кожним тестом є список результатів виконання тесту учнями, що є схожим до списку виконаних завдань. Натиснувши на результат виконання тесту, Вчитель перейде до сторінки перегляду результату виконання тесту (рис. 3.23) де може переглянути обрані Учнем варіанти відповіді, оцінку за тест, час початку й завершення тесту та час відповіді на запитання.

Результат учня Дмитро Дмитров Дмитрович: 75.00% (9)
Дата та час початку: 19.06.24 02:30
Дата та час завершення: 19.06.24 02:31

Питання
Відповідь дано: 19.06.24 02:31

запитання 2

Відповідь(-і):

вірно

вірно 2

Список питань

1 2 3

Рис. 3.23 Перегляд результату виконання тесту

Наступним компонентом системи є набір сторінок, пов'язаних з Учнями. Користувачі з роллю Учень можуть виконувати завдання та тести, а також переглядати оцінку за них.

Список завдань на сторінці учня виглядає так само як і на сторінці вчителя, єдина відмінність в тому, що в списку відображаються лише неприховані завдання, що відносяться до класу, членом якого є Учень. Обравши певне завдання, учень має змогу дати на нього відповідь за допомогою форми, що включає в себе поле для введення тексту з можливістю форматування. Після відправлення відповіді, учень може переглянути свою відповідь та оцінку (рис. 3.24), якщо вона була виставлена вчителем, може прикріпити зображення до своєї відповіді. Додатково, учень може залишити коментар.

Оцінка: 6/12

Твоя відповідь
Відповідь на завдання

Твої зображення

←



→

⊕

Рис. 3.24 Результат оцінювання відповіді на завдання

Список тестів також виглядає так само як і список тестів на сторінці вчителя, проте відображаються лише неприховані тести, що відносяться до класу учня. Обравши певний тест, учень може приступити до його виконання. Інтерфейс виконання схожий на інтерфейс тесту вчителя, відмінностями є наявність таймеру, відображення варіантів відповідей у вигляді перемикачів, чек-боксів чи поля вводу, в залежності від типу запитання та наявність кнопки завершити тест (рис. 3.25).

The screenshot shows a test interface with a light gray background. At the top left, the word "Питання" (Question) is displayed in bold. At the top right, a timer shows "Час до завершення: 19:44" (Time to completion: 19:44). Below the question title, the text "запитання 2" (question 2) is visible. A horizontal line separates the question from the answer section. Under the heading "Відповідь:" (Answer:), there are three radio button options: "вірно" (correct), "вірно 2" (correct 2), and "невірно" (incorrect). Below these options is a dark gray button with white text that says "Закріпити відповідь" (Lock answer).

Рис. 3.25 Інтерфейс відповіді на запитання тесту

Останнім компонентом додатку є перелік сторінок, пов'язаних з роллю Батьки. Користувачі з цією роллю мають змогу переглянути результати виконання завдань та тестів учнем. Перегляд результатів виглядає майже так само там має майже такий же функціонал як перегляд результатів на сторінці вчителя. Відмінність лише в відсутності можливості редагувати оцінку відповіді на завдання. Також користувач має змогу переглянути статистику результатів виконання завдань та тестів учнем (рис. 3.26).

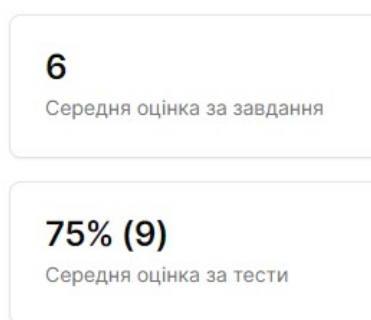


Рис. 3.26 Статистика результатів

При виконанні деяких ключових дій, таких як створення чи редагування певних записів, відображається повідомлення, що сповіщає про успішність виконання дії чи помилку (рис. 3.27). Дані повідомлення реалізовані за допомогою бібліотеки Sonner [30]. Бібліотека дозволяє ініціювати відображення повідомлення програмно, що підвищує інтерактивність системи.

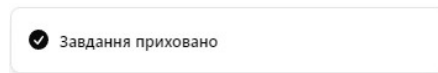


Рис. 3.27 Повідомлення про успішне приховання завдання

При розробці інтерфейсу, враховувалися різні розміри екранів, що робить його адаптивним (рис. 3.28). Це дозволяє комфортно використовувати додаток як з десктопу, так і з мобільного пристрою. Елементи на сторінці динамічно реагують на розмір екрану, змінюючи свою форму та положення за потреби, для забезпечення максимальної зручності користування.

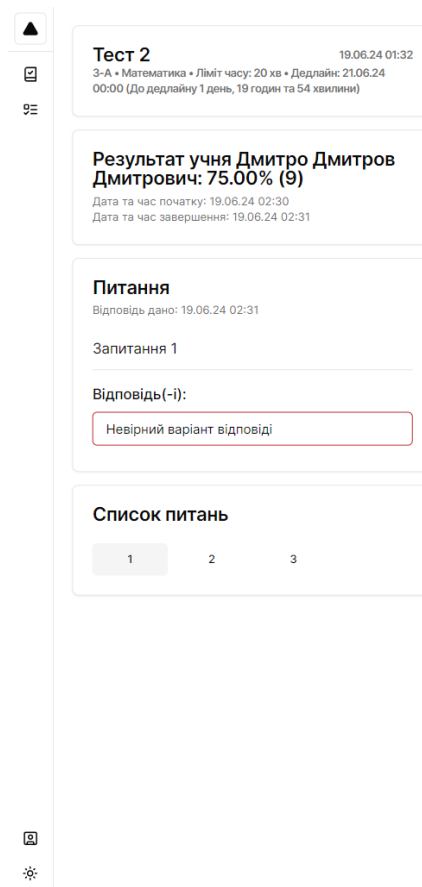


Рис. 3.28 Сторінка результату проходження тесту на маленькому екрані

3.3 Тестування роботи системи

Для забезпечення надійності функціоналу в процесі реалізації системи проводилось її ретельне тестування. Це допомагало виявляти помилки та дефекти системи на ранніх етапах, коли проблеми ще легко виправити. Тестування забезпечує чітку відповідність системи функціональним та нефункціональним вимогам.

В процесі розробки було використано різні техніки тест дизайну. При тестуванні форм додатку, було використано техніки еквівалентного поділу та аналізу граничних. Це допомогло протестувати поведінку системи та виявити помилки при введенні різних типів даних та їх граничних значень.

Для виявлення помилок у функціоналі додатку використовувались техніки причина/наслідок та передбачення помилки. Причина/наслідок допомагає знайти помилки в роботі системи після виконання певних дій (наприклад, при відправленні запиту на сервер, що є причиною, сервер повертає помилку, що є наслідком). Техніка передбачення помилки допомагає передбачити компонент системи, в якому найвірогідніше може відбутись помилка.

При знаходженні та виправленні проблем, виконувалось повторне тестування, щоб переконатися, що проблема була дійсно вирішена.

При впровадженні нових компонентів системи періодично виконувалось регресійне тестування для перевірки, чи не завдав новий компонент шкоди функціональності старих.

По завершенню роботи над системою було проведено її функціональне тестування для перевірки відповідності всіх функцій системи встановленим функціональним вимогам. Функціональне тестування визначило, що система та її функціонал працюють як передбачено та без збоїв.

Також було проведено тестування графічного інтерфейсу користувача для перевірки коректності всіх графічних елементів системи та правильності їх взаємодії з користувачем. Паралельно з цим проводилось тестування зручності використання з метою перевірити ефективність взаємодії

користувача з системою. Результати цих тестувань показали, що інтерфейс працює справно, відповідно до вимог, та є зручним у використанні.

Ретельне тестування на різних етапах розробки з використанням різних технік тест-дизайну забезпечило високу якість та надійність створеної системи, її повну відповідність вимогам та зручність використання.

Після завершення ручного тестування, було проведено тестування додатку за допомогою сервісу Lighthouse. Це потужний інструмент від Google для аудиту вебсайтів, що є частиною інструментів розробника у браузері Google Chrome [31]. Інструмент працює шляхом запуску серії тестів та симуляцій на вебсторінці, щоб оцінити її продуктивність, доступність, дотримання найкращих практик і пошукову оптимізацію. Він генерує докладний звіт із загальними оцінками в кожній із категорій, переліком проблем та порадами щодо їх вирішення, рекомендаціями щодо поліпшення. Звіт дозволяє робити оптимізації, базуючись на отриманих даних та кращих практиках. Інструментом було протестовано сторінку вчителя зі списком завдань (рис. 3.29).

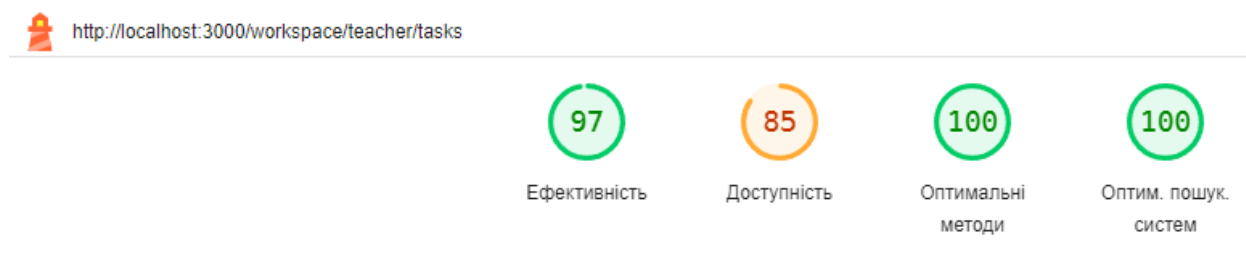


Рис. 3.29 Результат тестування сторінки

При тестуванні ефективності вебдодатку, інструмент надає можливість переглядати різні метрики швидкості, такі як: перший змістовний макет, найбільший змістовний макет, загальний час блокування, кумулятивний зсув макета та індекс швидкості (рис. 3.30). Ці метрики відображають швидкодію додатку та допомагають визначити конкретні аспекти, які потребують оптимізації.

● First Contentful Paint	■ Largest Contentful Paint
0,2 с	1,3 с
● Total Blocking Time	● Cumulative Layout Shift
0 мс	0
● Speed Index	
0,6 с	

Рис. 3.30 Метрики швидкодії вебдодатку

Інструмент надає детальний список рекомендацій та практичних порад щодо підвищення ефективності додатку, його доступності та пошукової оптимізації. Перевіряється відповідність вебдодатку загальноприйнятим практикам та оптимальним методам.

Lighthouse є важливим інструментом для сучасних веброзробників, який допомагає забезпечити високоякісні, швидкі та доступні вебсторінки.

Висновки до третього розділу

У даному розділі було розглянуто процес встановлення та налаштування параметрів системи. Описано порядок розгортання системи та взаємодію її компонентів, що дозволило чітко визначити архітектуру рішення. Була наведена діаграма розгортання, яка ілюструє, як компоненти системи співпрацюють між собою для забезпечення належної роботи додатку.

Також була представлена структура інтерфейсу системи виконання завдань та тестів. Описано основні функції користувачів: реєстрація та авторизація, створення та виконання завдань та тестів, а також можливості адміністратора, включаючи керування обліковими записами користувачів, створення та керування класами та навчальними предметами.

Було висвітлено процес тестування додатку, наведено способи застосування методів тестування.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано систему виконання завдань та тестів для шкільного закладу освіти. На початку роботи було проведено детальний аналіз предметної області та виконано порівняння аналогів програмного продукту, що допомогло визначити основні вимоги до системи. На основі цього аналізу були сформовані основні завдання для створення системи, визначені ключові функціональні можливості та етапи їх реалізації.

Було виконано проектування системи, що включало розгляд та аналіз варіантів використання та сценаріїв взаємодії користувачів із системою.

Спроектовано структуру бази даних, для забезпечення безпечного та ефективного зберігання та обробки даних про користувачів, класи, предмети, завдання й тести та результати їх виконання. На основі розроблених сутностей були створені таблиці у базі даних, а також діаграма бази даних для їх візуалізації.

Була представлена структура інтерфейсу додатку виконання завдань та тестів. Описано основний функціонал, доступ до якого мають звичайні користувачі, такий як реєстрація та авторизація, створення та виконання завдань та тестів, а також можливості адміністратора, включаючи створення та контроль над обліковими записами, створення класів та предметів. Використовуючи розроблену систему, користувачі можуть легко створювати та виконувати завдання, створювати та проходити тести, комунікувати шляхом коментарів та переглядати результати оцінювання.

Було описано основні процеси тестування системи, техніки та методи тестування, використані при розробці додатку. Завдяки проведеному тестуванню, кінцевий продукт відповідає поставленим вимогам та є зручним у використанні.

Результатом виконання кваліфікаційної роботи є справний програмний продукт, що відповідає поставленим вимогам, має інтуїтивний інтерфейс та надає очікуваний функціонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Moodle [Електронний ресурс]. Режим доступу до ресурсу: <https://moodle.org/>
2. HUMAN Школа [Електронний ресурс]. Режим доступу до ресурсу: <https://www.human.ua/>
3. Google Classroom [Електронний ресурс]. Режим доступу до ресурсу: <https://classroom.google.com/>
4. Node.js [Електронний ресурс]. Режим доступу до ресурсу: <https://nodejs.org/uk/>
5. V8 [Електронний ресурс]. Режим доступу до ресурсу: <https://v8.dev/>
6. Express.js [Електронний ресурс]. Режим доступу до ресурсу: <https://expressjs.com/>
7. JavaScript [Електронний ресурс]. Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
8. PostgreSQL [Електронний ресурс]. Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
9. Prisma [Електронний ресурс]. Режим доступу до ресурсу: <https://www.prisma.io/docs>
10. Next.js [Електронний ресурс]. Режим доступу до ресурсу: <https://nextjs.org/docs>
11. React.js [Електронний ресурс]. Режим доступу до ресурсу: <https://uk.reactjs.org/docs/getting-started.html>
12. Tailwind CSS [Електронний ресурс]. Режим доступу до ресурсу: <https://tailwindcss.com/docs>
13. ShadCN [Електронний ресурс]. Режим доступу до ресурсу: <https://ui.shadcn.com/docs>
14. Axios [Електронний ресурс]. Режим доступу до ресурсу: <https://axios-http.com/docs/intro>

15. JSON Web Token [Электронный ресурс]. Режим доступа до ресурсу: <https://jwt.io/introduction>
16. NextAuth.js [Электронный ресурс]. Режим доступа до ресурсу: <https://next-auth.js.org/getting-started/introduction>
17. React useContext [Электронный ресурс]. Режим доступа до ресурсу: <https://react.dev/reference/react/useContext>
18. Next.js Routing using App Router [Электронный ресурс]. Режим доступа до ресурсу: <https://nextjs.org/docs/app/building-your-application/routing>
19. SimpleCrypto [Электронный ресурс]. Режим доступа до ресурсу: <https://simplecrypto.js.org/docs/>
20. Bcrypt [Электронный ресурс]. Режим доступа до ресурсу: <https://www.npmjs.com/package/bcrypt>
21. Lodash [Электронный ресурс]. Режим доступа до ресурсу: <https://lodash.com/docs/4.17.15#debounce>
22. React Hook Form [Электронный ресурс]. Режим доступа до ресурсу: <https://react-hook-form.com/docs>
23. Zod [Электронный ресурс]. Режим доступа до ресурсу: <https://zod.dev/>
24. TipTap [Электронный ресурс]. Режим доступа до ресурсу: <https://tiptap.dev/docs>
25. DOMPurify [Электронный ресурс]. Режим доступа до ресурсу: <https://www.npmjs.com/package/dompurify>
26. LaTeX [Электронный ресурс]. Режим доступа до ресурсу: <https://www.latex-project.org/about/>
27. KaTeX [Электронный ресурс]. Режим доступа до ресурсу: <https://katex.org/docs>
28. Socket.IO [Электронный ресурс]. Режим доступа до ресурсу: <https://socket.io/docs>
29. ImageKit.io [Электронный ресурс]. Режим доступа до ресурсу: <https://docs.imagekit.io/>

30. Sonner [Электронный ресурс]. Режим доступа до ресурсу:
<https://sonner.emilkowal.ski/getting-started>

31. Lighthouse [Электронный ресурс]. Режим доступа до ресурсу:
<https://developer.chrome.com/docs/lighthouse/overview>

32. ESLint [Электронный ресурс]. Режим доступа до ресурсу:
<https://eslint.org/docs/latest/>

33. React useState [Электронный ресурс]. Режим доступа до ресурсу:
<https://react.dev/reference/react/useState>

34. React useEffect [Электронный ресурс]. Режим доступа до ресурсу:
<https://react.dev/reference/react/useEffect>

ДОДАТКИ

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						85

```

class UserController {
  async registration(req, res, next) {
    const { id } = req.params
    const { email, password } = req.body

    if (!validate(email, password)) {
      return next(ApiError.badRequest('INVALID_DATA'))
    }

    const existingUser = await db.user.findUnique({ where: {
id } })
    if (existingUser) {
      if (existingUser.email || existingUser.passwordHash)
{
        return
next(ApiError.badRequest('USER_ALREADY_HAS_EMAIL_AND_PASSWORD'))
      }
    } else {
      return
next(ApiError.badRequest('USER_DOES_NOT_EXIST'))
    }

    const candidate = await db.user.findUnique({ where: {
email } })
    if (candidate) {
      return next(ApiError.badRequest('EMAIL_EXISTS'))
    }

    const hashPassword = await bcrypt.hash(password, 5)
    const updatedUser = await db.user.update({
      where: { id: id },
      data: {
        email,
        passwordHash: hashPassword,
      },
    })

    const roles = await
getUserRolesWithoutRequest(updatedUser.id)
    const token = generateToken(updatedUser.id, roles)
    return res.json({ token })
  }

  async login(req, res, next) {
    const { email, password } = req.body

```

```

        if (!validate(email, password)) {
            return next(ApiError.badRequest('INVALID_DATA'))
        }

        const user = await db.user.findUnique({ where: { email }
    })

    if (!user) {
        return next(ApiError.badRequest('EMAIL_NOT_FOUND'))
    }

    let comparePassword = bcrypt.compareSync(password,
user.passwordHash)
    if (!comparePassword) {
        return next(ApiError.badRequest('WRONG_PASSWORD'))
    }

    const roles = await getUserRolesWithoutRequest(user.id)
    const token = generateToken(user.id, roles)
    return res.json({ token })
}

async addUser(req, res, next) {
    const { name, surname, patronymic, roles, classId,
parentId } = req.body

    const newUser = await db.user.create({
        data: {
            name,
            surname,
            patronymic,
            roles,
            classId: classId === '' ? null :
parseInt(classId),
            parentId: parentId === '' ? null : parentId,
        },
    })

    res.json(newUser)
}

async updateUser(req, res, next) {
    const { id } = req.params
    const { name, surname, patronymic, roles, classId,
parentId } = req.body

```

```

        const updatedUser = await db.user.update({
            where: { id: id },
            data: {
                name,
                surname,
                patronymic,
                roles,
                classId: classId === '' ? null :
parseInt(classId),
                parentId: parentId === '' ? null : parentId,
            },
        })

        res.json(updatedUser)
    }

    async getUser(req, res, next) {
        const { id } = req.params
        const user = await db.user.findUnique({
            where: { id: id },
            select: {
                id: true,
                email: true,
                name: true,
                surname: true,
                patronymic: true,
                roles: true,
                classId: true,
                parentId: true,
                children: {
                    select: {
                        id: true,
                        name: true,
                        surname: true,
                        patronymic: true,
                    },
                },
            },
        })
        res.json(user)
    }

    async getUserRoles(req, res, next) {
        const { id } = req.params
        const userRoles = await db.user.findUnique({
            where: { id: id },

```

```

        select: {
            roles: true,
        },
    })

    res.json(userRoles.roles)
}

async getUsersByRole(req, res, next) {
    const { role } = req.params
    const { limit = 10, page = 1 } = req.query
    const limitInt = parseInt(limit, 10)
    const pageInt = parseInt(page, 10)
    const offsetInt = (pageInt - 1) * limitInt

    const users = await db.user.findMany({
        where: { roles: { has: role } },
        select: {
            id: true,
            email: true,
            name: true,
            surname: true,
            patronymic: true,
            classId: true,
            parentId: true,
        },
        take: limitInt,
        skip: offsetInt,
    })

    const total = await db.user.count({
        where: { roles: { has: role } },
    })

    res.json({ users, total })
}

async getUsersByClass(req, res, next) {
    const { classId } = req.params
    const users = await db.user.findMany({
        where: { classId: parseInt(classId) },
        select: {
            id: true,
            email: true,
            name: true,
            surname: true,

```

```

        patronymic: true,
        roles: true,
        parentId: true,
    },
})

res.json(users)
}

async getUsersByPartOfFullNameAndRole(req, res, next) {
    const { query, role } = req.body
    const { limit, offset = 0 } = req.query

    const offsetInt = parseInt(offset, 10)

    const terms = query
        .split(' ')
        .map((term) => term.trim())
        .filter((term) => term.length > 0)
    if (terms.length === 0) {
        return res.status(400).json({ message: 'Query
parameter is empty' })
    }

    const searchConditions = terms.map((term) => ({
        OR: [
            { name: { contains: term, mode:
'insensitive' } },
            { surname: { contains: term, mode: 'insensitive'
} },
            { patronymic: { contains: term, mode:
'insensitive' } },
        ],
    }))

    let roleCondition = {}
    if (role) {
        roleCondition = {
            roles: {
                has: role,
            },
        }
    }

    const findManyParams = {
        where: {

```

```

        AND: [...searchConditions, roleCondition],
      },
      skip: offsetInt,
    }

    if (limit !== undefined) {
      findManyParams.take = parseInt(limit, 10)
    }

    const users = await db.user.findMany(findManyParams)
    res.json(users)
  }

  async resetUser(req, res, next) {
    const { id } = req.params

    const updatedUser = await db.user.update({
      where: { id: id },
      data: {
        email: null,
        passwordHash: null,
      },
      select: {
        id: true,
        email: true,
        name: true,
        surname: true,
        patronymic: true,
        roles: true,
        classId: true,
        parentId: true,
        children: {
          select: {
            id: true,
            name: true,
            surname: true,
            patronymic: true,
          },
        },
      },
    })

    res.json(updatedUser)
  }

  async getUserRolesWithoutRequest(id) {

```

```

        const userRoles = await db.user.findUnique({
          where: { id: id },
          select: {
            roles: true,
          },
        })

        return userRoles.roles
      }
    }

    class TestResultController {
      async createTestResult(req, res) {
        const { studentId, testId } = req.body
        const testResult = await db.testResult.create({
          data: {
            student: { connect: { id: studentId } },
            test: { connect: { id: parseInt(testId) } },
          },
        })
        res.json(testResult)
      }

      async getTestResultsByTestId(req, res) {
        const { testId } = req.params
        const testResults = await db.testResult.findMany({
          where: { testId: parseInt(testId) },
          include: {
            student: {
              select: {
                id: true,
                name: true,
                surname: true,
                patronymic: true,
              },
            },
          },
        })
        res.json(testResults)
      }

      async getTestResultsByStudentId(req, res) {
        const { studentId } = req.params
        const testResults = await db.testResult.findMany({
          where: { studentId: studentId },
          include: {

```

```

        test: {
            include: {
                class: true,
                subject: true,
            },
        },
        student: {
            select: {
                id: true,
                name: true,
                surname: true,
                patronymic: true,
            },
        },
    },
})
res.json(testResults)
}

async getOneTestResult(req, res) {
    const { id } = req.params
    const testResult = await db.testResult.findUnique({
        where: { id: parseInt(id) },
        include: {
            answers: {
                include: {
                    answer: true,
                    question: {
                        include: {
                            testAnswers: true,
                        },
                    },
                },
            },
        },
        student: {
            select: {
                id: true,
                name: true,
                surname: true,
                patronymic: true,
            },
        },
        test: {
            include: {
                class: true,
                subject: true,
            },
        },
    })
    res.json(testResult)
}

```

```

        questions: true,
      },
    },
  },
})
res.json(testResult)
}

async getTestResultByStudentIdTestId(req, res) {
  const { studentId, testId } = req.body
  const testResult = await db.testResult.findFirst({
    where: { studentId: studentId, testId:
parseInt(testId) },
  })
  res.json(testResult)
}

async updateTestResult(req, res) {
  const { id } = req.params
  const { studentId, testId } = req.body
  const testResult = await db.testResult.update({
    where: { id },
    data: {
      student: { connect: { id: studentId } },
      test: { connect: { id: testId } },
    },
  })
  res.json(testResult)
}

async completeTestResult(req, res) {
  const { id } = req.params
  const testResultId = parseInt(id)

  const testResult = await db.testResult.findUnique({
    where: { id: testResultId },
    include: {
      answers: {
        include: {
          answer: true,
          question: {
            include: {
              testAnswers: true,
            },
          },
        },
      },
    },
  })
}

```

```

        },
    },
})

    if (!testResult) {
        return res.status(404).json({ error: `TestResult
with ID ${testResultId} not found.` })
    }

    const completedAt = new Date()

    let totalConsideredAnswers = 0
    let correctAnswers = 0

    testResult.answers.forEach((studentAnswer) => {
        const question = studentAnswer.question

        const correctTestAnswers =
question.testAnswers.filter((testAnswer) =>
testAnswer.isCorrect)
        const hasCorrectAnswers = correctTestAnswers.length
> 0

        const hasAnyAnswers = question.testAnswers.length >
0

        if (!hasAnyAnswers || !hasCorrectAnswers) {
            return
        }

        if (question.isManual) {
            const correctManualAnswers =
correctTestAnswers.map((testAnswer) =>
testAnswer.text.toLowerCase().trim())

            if
(correctManualAnswers.includes(studentAnswer.manualAnswer?.toLow
erCase().trim())) {
                correctAnswers++
            }
        } else {
            const studentAnswersForQuestion =
testResult.answers.filter((ans) => ans.questionId ===
question.id)

            const correctStudentAnswers =
studentAnswersForQuestion.filter(

```

```

        (ans) => ans.answer &&
correctTestAnswers.some((correctAnswer) => correctAnswer.id ===
ans.answer.id)
    )

    if (correctStudentAnswers.length ===
correctTestAnswers.length) {
        correctAnswers++
    } else {
        correctAnswers +=
correctStudentAnswers.length / correctTestAnswers.length
    }
}

totalConsideredAnswers++
})

const mark = totalConsideredAnswers > 0 ? correctAnswers
/ totalConsideredAnswers : 0

const result = await db.testResult.update({
    where: { id: testResultId },
    data: {
        completedAt,
        mark,
    },
})

res.json(result)
}

async getAvarageMarkByStudentId(req, res) {
    const { studentId } = req.params
    const aggregations = await db.testResult.aggregate({
        _avg: {
            mark: true,
        },
        where: {
            studentId,
        },
    })

    res.json({ avgMark: aggregations._avg.mark })
}

async deleteTestResult(req, res) {

```

```

        const { id } = req.params
        const testResult = await db.testResult.delete({ where: {
id } })
        res.json(testResult)
    }
}

export const authOptions = {
  providers: [
    CredentialsProvider({
      name: 'credentials',
      async authorize(credentials) {
        try {
          const user = await
getToken(credentials.email, credentials.password)
          if (user) {
            return user
          }
        } catch (error) {
          throw new Error(`Authentication failed (${
{error}})` )
        }
      },
    }),
  ],
  session: {
    strategy: 'jwt',
    maxAge: 7 * 24 * 60 * 60, // 7 days
  },
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.accessToken = user.token
      }

      return token
    },
    async session({ session, token }) {
      session.accessToken = token.accessToken

      return session
    },
  },
  secret: process.env.NEXTAUTH_SECRET,
  debug: process.env.NEXT_ENV === 'development',
}

```

```

export const authenticator = async () => {
  try {
    const response = await
fetch(process.env.NEXT_PUBLIC_IMAGEKIT_AUTHENTICATION_ENDPOINT)

    if (!response.ok) {
      const errorText = await response.text()
      throw new Error(`Request failed with status $
{response.status}: ${errorText}`)
    }

    const data = await response.json()
    const { signature, expire, token } = data
    return { signature, expire, token }
  } catch (error) {
    throw new Error(`Authentication request failed: $
{error.message}`)
  }
}

export const imageKitLoader = ({ src, width, quality }) => {
  if (src[0] === '/') src = src.slice(1)
  const params = [`w-${width}`]
  if (quality) {
    params.push(`q-${quality}`)
  }
  const paramsString = params.join(',')
  var urlEndpoint =
process.env.NEXT_PUBLIC_IMAGEKIT_URL_ENDPOINT
  if (urlEndpoint[urlEndpoint.length - 1] === '/') urlEndpoint
= urlEndpoint.substring(0, urlEndpoint.length - 1)
  return `${urlEndpoint}/${src}?tr=${paramsString}`
}

const imagekit = new ImageKit({
  urlEndpoint: process.env.IMAGEKIT_URL_ENDPOINT,
  publicKey: process.env.IMAGEKIT_PUBLIC_KEY,
  privateKey: process.env.IMAGEKIT_PRIVATE_KEY,
})

router.get('/imagekit-auth', (req, res) => {
  var result = imagekit.getAuthenticationParameters()
  res.send(result)
})

```

```

module.exports = function (req, res, next) {
  if (req.method === 'OPTIONS') {
    next()
  }

  try {
    const token = req.headers.authorization.split(' ')[1]
    if (!token) {
      return res.status(401).json({ message:
'unauthorised' })
    }

    const decoded = jwt.verify(token,
process.env.SECRET_KEY)
    req.user = decoded
    next()
  } catch (e) {
    res.status(401).json({ message: 'unauthorised' })
  }
}

```