НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.
(підпис) (ПІБ)

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення системи навчання та контролю знань з правил дорожнього руху»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н., доцент_____ _____ __Вайганг Г.О.___
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ __Хиленко В.В.____
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ _Немеш_О.О._____
(підпис) (ПІБ студента)

КИЇВ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук

_____ _____ Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)

"___" _____ 2025 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Немешу Олександру Олександровичу

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: «Програмне забезпечення системи навчання та контролю знань з правил дорожнього руху»

Затверджена наказом ректора НУБіП України від 16.12.2024 р. № 2248 «С».

Термін подання завершеної роботи на кафедру _____

Вихідні дані до бакалаврської кваліфікаційної роботи: порівняння існуючих рішень, опис програмного забезпечення та використаних бібліотек

Перелік питань, які потрібно розробити: аналіз предметної області, вибір засобів для розробки системи

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Хиленко В.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Немеш О.О.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5
ВСТУП 6
1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Опис предметної області 7
1.2 Огляд інформаційних джерел існуючих рішень 8
1.3 Аналіз вимог до програмної системи 12
1.3.1. Бізнес вимоги 12
1.3.2. Функціональні вимоги 13
1.3.3. Нефункціональні вимоги 13
1.4 Моделювання предметної області 14
1.5 Висновок до розділу 1 17
2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19
2.1 Логічна модель даних у вигляді ER-діаграми 19
2.2 Діаграма класів та кооперацій 22
2.3 Діаграма пакетів 26
2.4 Діаграма компонентів 29
2.5 Висновок по розділу 2 32
3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .. 33
3.1 Система управління інформаційною базою 33
3.2 Вибір інструментарію для створення прикладного програмного забезпечення 35
3.3 Розробка інформаційної бази 38
3.4 Алгоритмізація та програмування програмних модулів 40
Алгоритм процесу авторизації користувача 40
3.5 Висновок по розділу 3 52

4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 54
4.1 Тестування системи 54
4.2 Вимоги до апаратного та програмного забезпечення 57
4.3 Склад інсталяційного пакету 59
База даних 62
ВИСНОВКИ 64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 65
ДОДАТКИ 66
Додаток А 66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПДР – Правила Дорожнього Руху

МВС – Міністерство внутрішніх справ України

JWT – JSON Web Token

JSON – JavaScript Object Notation

XSS – Cross-Site Scripting

UML – Unified Modeling Language

СУБД – Система управління базами даних

SSR – Server-Side Rendering

IDE – Integrated Development Environment

API – Application Programming Interface

ORM – Object-relational mapping

OWASP - Open Web Application Security Project

ВСТУП

У сучасному світі автомобільний рух набирає дедалі більших обертів, що вимагає від учасників дорожнього руху високого рівня знань та відповідальності. Особливо важливим є забезпечення ефективного навчання правил дорожнього руху (ПДР) для майбутніх водіїв. Проте традиційні методи навчання зазвичай не враховують сучасних потреб користувачів — особливо молоді, яка звикла до інтерактивних рішень, гейміфікації та цифрових технологій.

Актуальність теми обумовлена необхідністю підвищення ефективності засвоєння знань з ПДР за допомогою інформаційних технологій, зокрема веб-застосунків, які забезпечують гнучкість, доступність і персоналізований підхід. Мета роботи — розробити програмне забезпечення для навчання та контролю знань ПДР, що базується на гейміфікованій системі з адаптивним тестуванням та інтерактивними елементами.

У процесі виконання роботи було проведено аналіз предметної області, досліджено та оцінено існуючі рішення, сформульовано вимоги до майбутньої системи, розроблено її архітектуру, реалізовано функціональні модулі та проведено тестування. Результатом є повнофункціональний веб-застосунок, який можна використовувати для самостійного вивчення ПДР та підготовки до іспитів.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасне суспільство характеризується активним збільшенням використання транспортних засобів та кількості учасників дорожнього руху, що створює більше проблем з безпекою на дорогах. У зв'язку з цим, виникає потреба у створенні ефективних систем навчання ПДР з використанням сучасних технологій.

Зазвичай, вивчення правил дорожнього руху являє собою заучування теоретичного матеріалу та проходження стандартизованих тестів. Це не є дуже ефективним способом для молоді, яка більше звикла до інтерактивних систем та гейміфікованих додатків. Відсутність інтерактивного елементу знижує концентрацію та бажання навчатись, що може негативно відобразитись на засвоєнні матеріалу.

Найбільш актуальною проблемою є саме підтримка мотивації до навчання. Вивчення правил потребує не лише одноразового вивчення матеріалу, але й його регулярного повторення для закріплення знань. Система щоденних завдань та підтримки навчального "стріку" дозволяє сформувати звичку до систематичного навчання.

Не менш значущою є також проблема доступності навчання. Веб-застосунок забезпечить доступ до навчання у будь-який час та у будь-якому місці з доступом до Інтернету. Це може бути дуже важливо для людей з невеликою кількістю вільного часу, а також для людей з обмеженими можливостями пересування які не можуть відвідувати спеціалізовані центри підготовки водіїв. Адаптивність інтерфейсу до різних розмірів екрану розширить аудиторію користувачів.

Отже, створення програмного забезпечення системи навчання та контролю знань з правил дорожнього руху у формі інтерактивного веб-застосунку з елементами гейміфікації дозволить не лише підвищити ефективність навчання, але й сформувати мотивацію до систематичного повторення знань. Це допоможе

користувачам отримати необхідні знання для отримання водійського посвідчення та зробить їх більш відповідальними водіями.

1.2 Огляд інформаційних джерел існуючих рішень

Спочатку, потрібно розглянути приклади реалізації гейміфікованих систем навчання. Це допоможе зрозуміти які вимоги та функціонал системи необхідно передбачити для найкращого засвоєння матеріалу. Одним з найкращів прикладів такої системи можна назвати **Duolingo** – сервіс для вивчення різних мов у гейміфікованому форматі[1].

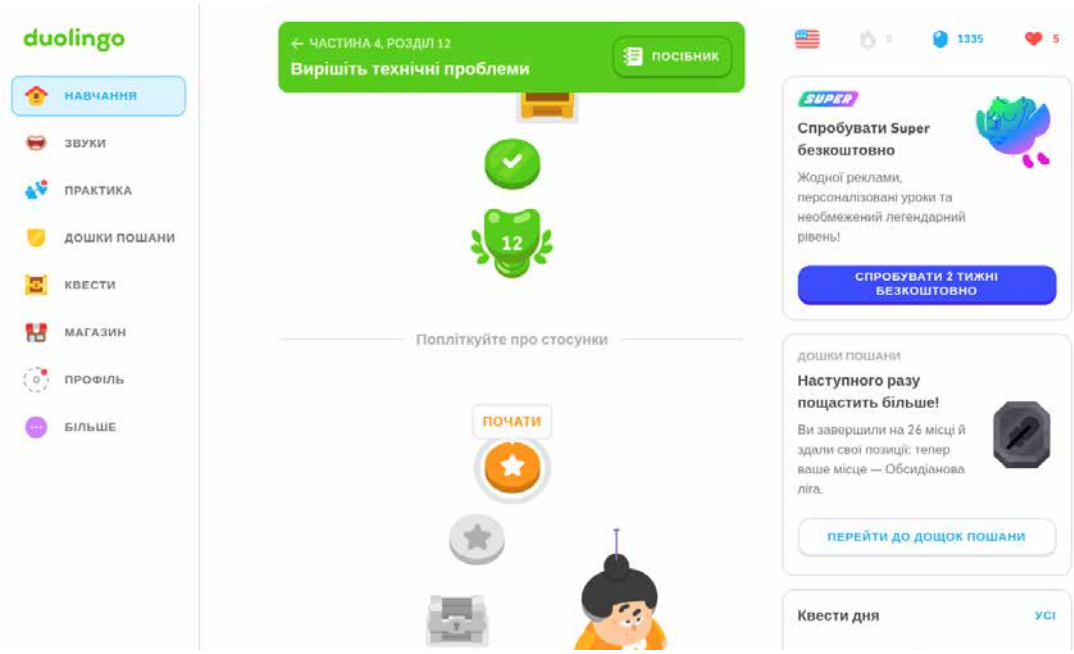

Рис 1.1. Скриншот платформи Duolingo

До переваг Duolingo можна віднести:

- Яскрава гейміфікація з системою очок, рівнів та досягнень
- Висока мотивація користувачів через щоденні завдання та “стріки”
- Адаптивна система навчання з поступовим ускладненням матеріалу
- Візуалізація прогресу та детальна статистика навчання
- Соціальні елементи та змагання між користувачами

[1] Duolingo – Найкращий спосіб вивчати мову [Електронний ресурс]. - https://uk.duolingo.com/

- Інтуїтивний та привабливий користувацький інтерфейс

Тепер можна розглянути існуючі рішення з навчання правил дорожнього руху. Першим таким рішенням можна назвати **Green Way** – один з найпопулярніших сервісів для підготовки до іспитів ПДР в Україні[2].

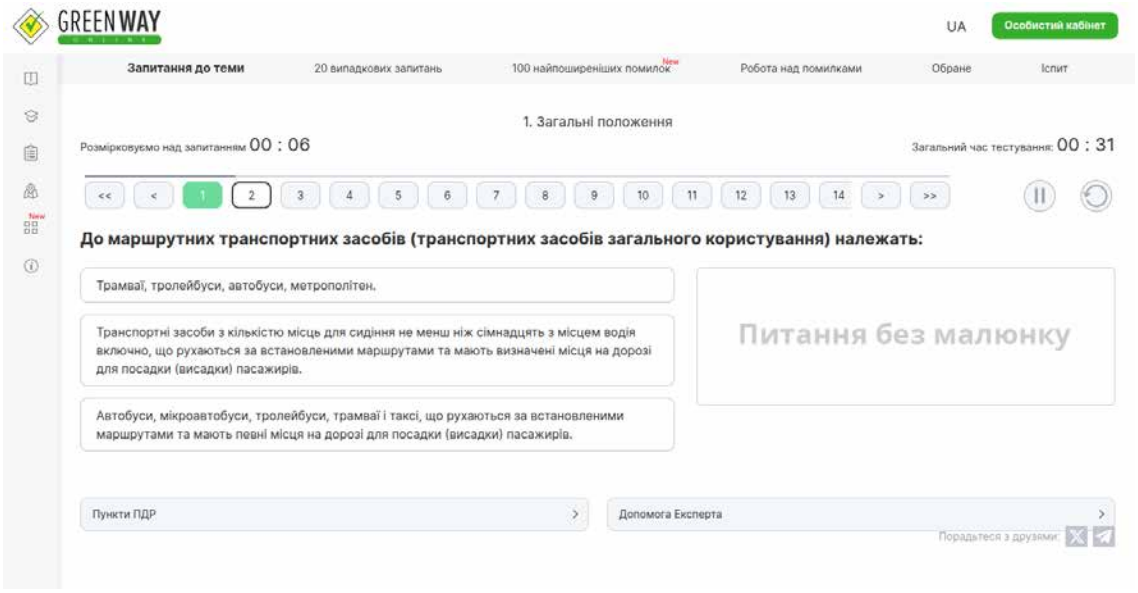

Рис 1.2. Скриншот платформи Green Way

Переваги даної системи:

- Актуальні питання відповідно до чинного законодавства України
- Офіційні екзаменаційні білети, затверджені МВС
- Зручний та інтуїтивний користувацький інтерфейс
- Можливість вибору категорії транспортного засобу
- Статистика результатів тестування
- Безкоштовний доступ до основного функціоналу

Недоліки:

- Повна відсутність елементів гейміфікації
- Статичний підхід до навчання без адаптації до рівня користувача
- Лише тестування без детальних пояснень правильних відповідей

[2] Green Way. Правила дорожнього руху України (ПДР 2025), вчити онлайн [Електронний ресурс]. - https://green-way.com.ua/uk/

- Відсутність мотиваційних механізмів для регулярного навчання
- Немає системи прогресу та досягнень

Наступною системою є сервіс з онлайн тестами **Vodiy.ua**, який являє собою добірку питань з офіційних білетів[3].

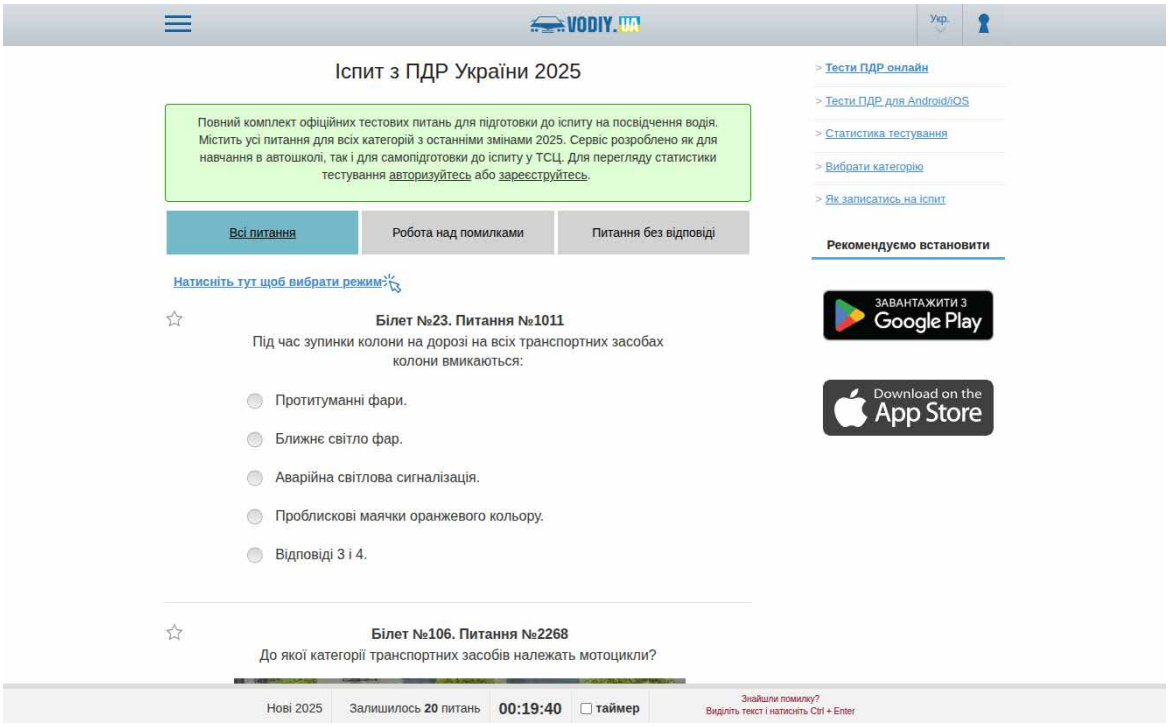

Рис 1.3. Скриншот сервісу Vodiy.ua

Переваги:

- Простота та доступність використання
- Зручний вибір категорій транспортних засобів
- Швидкий доступ до тестування без реєстрації
- Базова статистика проходження тестів
- Безкоштовність більшої частини сервісу (деякі функції потребують преміум-акаунту)

Недоліки:

- Відсутність адаптивного навчання та персоналізації
- Функціонал обмежений лише перевіркою знань

[3] Vodiy.ua. Тести з ПДР комплект Нові 2025. [Електронний ресурс]. - https://vodiy.ua/

- Немає навчальних матеріалів чи пояснень
- Відсутні мотиваційні елементи
- Застарілий дизайн та обмежена функціональність

Остання система яку ми розглянемо це мобільний застосунок “**ПДР 2025**”, який дозволяє вивчати правила за допомогою смартфону під управлінням Android чи iOS.

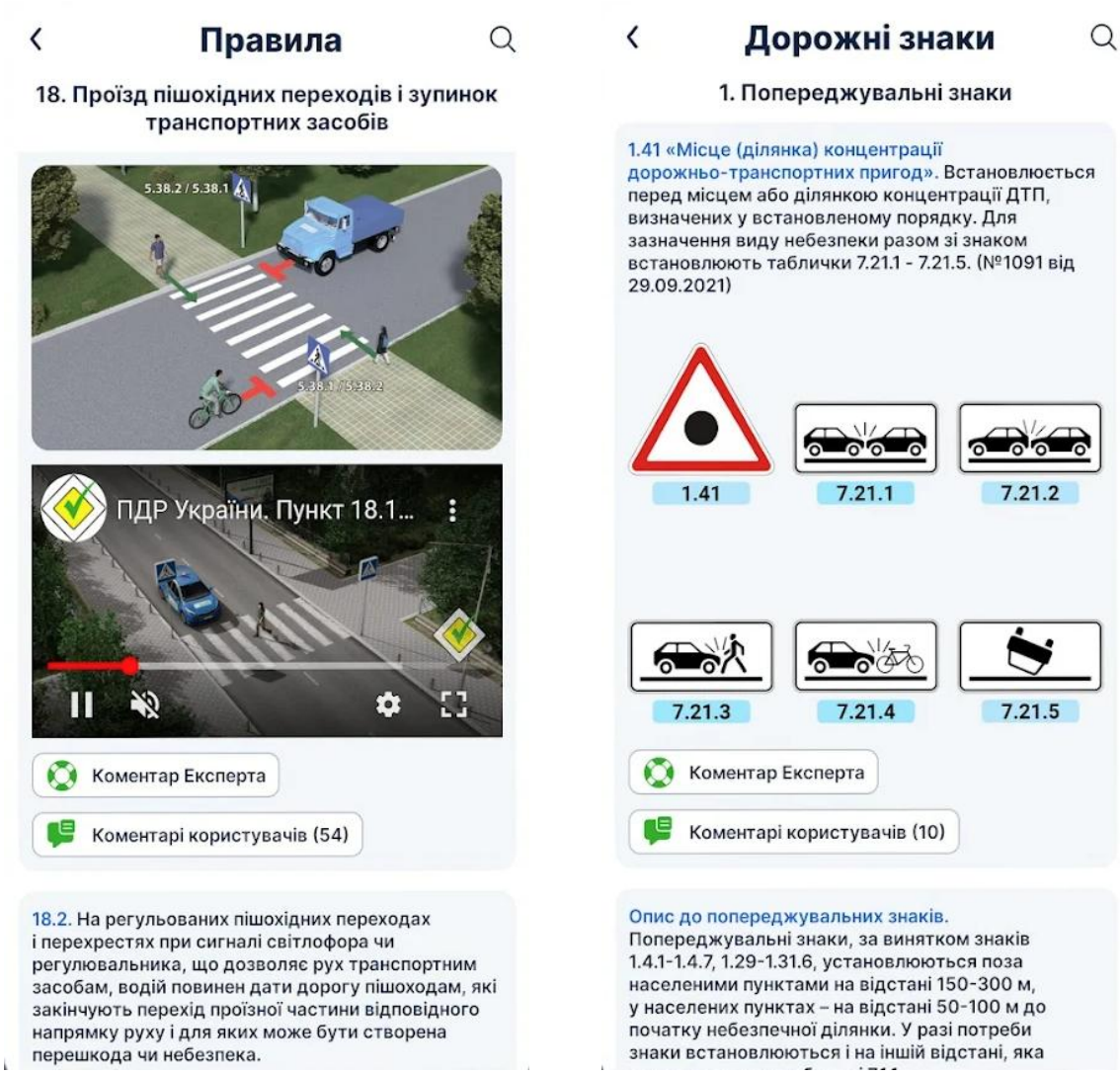

Рис 1.4. Скріншоти застосунку ПДР 2025

Переваги:

- Можливість роботи в офлайн-режимі
- Детальні пояснення до питань з посиланнями на ПДР
- Теоретичні матеріали для вивчення правил

- Зручна мобільна платформа для навчання в дорозі

Недоліки:

- Низький рівень інтерактивності та залученості користувача
- Відсутність тестів, тільки теорія
- Немає адаптації до індивідуального рівня знань користувача
- Відсутність системи мотивації та регулярного навчання

Аналіз існуючих рішень показує, що на ринку присутні два основні типи продуктів: високогейміфіковані навчальні платформи (як Duolingo) без спеціалізації на ПДР та спеціалізовані системи для вивчення правил дорожнього руху без елементів гейміфікації. Це створює нішу для розробки продукту, який поєднував би переваги обох підходів - глибоку спеціалізацію на ПДР із сучасними методами гейміфікованого навчання.

1.3 Аналіз вимог до програмної системи

Після аналізу систем аналогів, наступним кроком є визначення списку вимог до проєктованої програмної системи. Основними рівнями вимог можна визначити: бізнес-вимоги, функціональні вимоги, нефункціональні вимоги.

1.3.1. Бізнес вимоги

До основних бізнес-вимог такої системи відноситься:

- Підвищення ефективності навчання ПДР через застосування методів гейміфікації, що дозволить збільшити рівень засвоєння знань та мотивацію користувачів до регулярного навчання..
- Зниження аварійності на дорогах через підвищення якості знань правил дорожнього руху серед людей.

- Автоматизація процесу контролю знань з можливістю аналізу слабких місць у підготовці користувачів.
- Формування культури безперервного навчання через систему щоденних завдань та підтримки мотивації до регулярного повторення матеріалу.

1.3.2. Функціональні вимоги

- Користувач повинен мати можливість реєструватися, авторизуватися та керувати своїм профілем.
- Надавати адаптивні тести з ПДР відповідно до рівня знань користувача.
- Підтримувати геймізацію: досвід, рівні та систему очок.
- Генерувати щоденні завдання та відстежувати "стрік" навчання.
- Надавати детальну статистику результатів та аналіз слабких місць.
- Зберігати історію тестів та надавати можливість повторення невдалих питань.

1.3.3. Нефункціональні вимоги

Продуктивність:

- Забезпечувати час відповіді на запити не більше 2 секунд.
- Підтримувати одночасну роботу щонайменше 1000 користувачів.

Безпека:

- Забезпечити зберігання паролів у захищеному вигляді з використанням алгоритму Argon2.
- Забезпечити аутентифікацію через JWT-токени з обмеженим часом дії.
- Система повинна бути захищена від критичних вразливостей таких як SQL-ін’єкція, XSS (Cross-Site Scripting), тощо.

Надійність:

- Забезпечувати доступність 99% часу роботи.
- Виконувати автоматичне резервне копіювання бази даних щоденно.

Сумісність:

- Веб-застосунок повинен працювати у всіх сучасних браузерах.
- Система повинна бути адаптивною для роботи на мобільних пристроях та комп’ютерах.
- Підтримувати українську мову та українські правила дорожнього руху.

1.4 Моделювання предметної області

При моделюванні програмної системи, прийнято використовувати різноманітні UML-діаграми, які будуть відображати структуру чи поведінку певних частин системи.

UML (Unified Modeling Language) є стандартизованою мовою моделювання, що широко застосовується в програмній інженерії для візуалізації, специфікації, конструювання та документування програмних систем.[4]

Діаграма прецедентів (Use-case діаграма) є одним з основних типів UML-діаграм, призначених для моделювання функціональних вимог системи. Вона показує взаємодію між зовнішніми акторами (користувачами або іншими системами) та функціональністю системи, представленою у вигляді прецедентів (випадків використання). Діаграма прецедентів допомагає зрозуміти, що система повинна робити з точки зору користувача, не заглиблюючись в технічні деталі реалізації.

На рисунку 1.5 зображена діаграма прецедентів системи навчання ПДР. На ній визначено два основні актори:

Учень (Користувач) - основний користувач системи, який взаємодіє з наступними прецедентами:

- Перегляд статистики - можливість переглядати свої результати, прогрес навчання та аналітику успішності
- Проходити тест - основна функція системи, що дозволяє користувачу проходити тестування з ПДР

[4] Unhelkar B. Software Engineering with UML. Boca Raton : Auerbach Publications, 2020. 426 p.

- Перегляд тем - можливість ознайомлення з навчальними матеріалами та тематичними розділами ПДР

Адміністратор - користувач з розширеними правами, який має доступ до редагування контенту, а саме: управління питаннями, тестами та іншим контентом системи

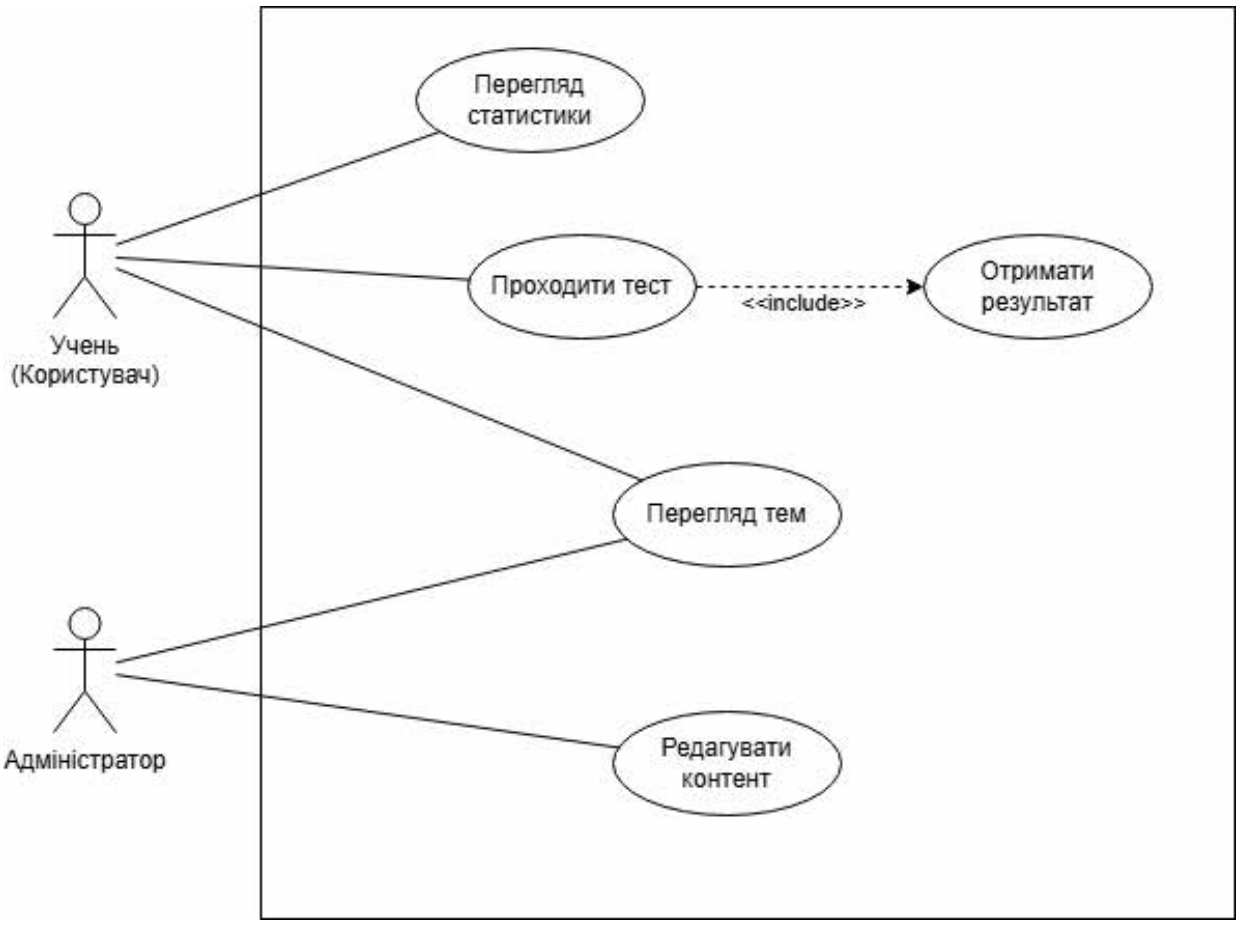

Рис 1.5. Діаграма прецедентів

Діаграма також показує відношення включення (include) між прецедентами "Проходити тест" та "Отримати результат", що означає, що отримання результату є невід'ємною частиною процесу проходження тесту.

Діаграма послідовності є UML-діаграмою, яка показує взаємодію між об'єктами системи в часі. Вона демонструє порядок обміну повідомленнями між різними компонентами системи для виконання конкретного сценарію або

прецеденту. Діаграма послідовності особливо корисна для розуміння динамічної поведінки системи та виявлення потенційних проблем у логіці взаємодії.

Представлена на рисунку 1.6 діаграма послідовності описує процес проходження тесту користувачем та включає такі основні компоненти:

- **Учень** - ініціатор процесу тестування
- **Тест** - об'єкт, що представляє тестове завдання
- **Правила Дорожнього Руху** - компонент, що містить логіку обробки відповідей та правил
- **Результати Тесту** - об'єкт для зберігання та відображення результатів

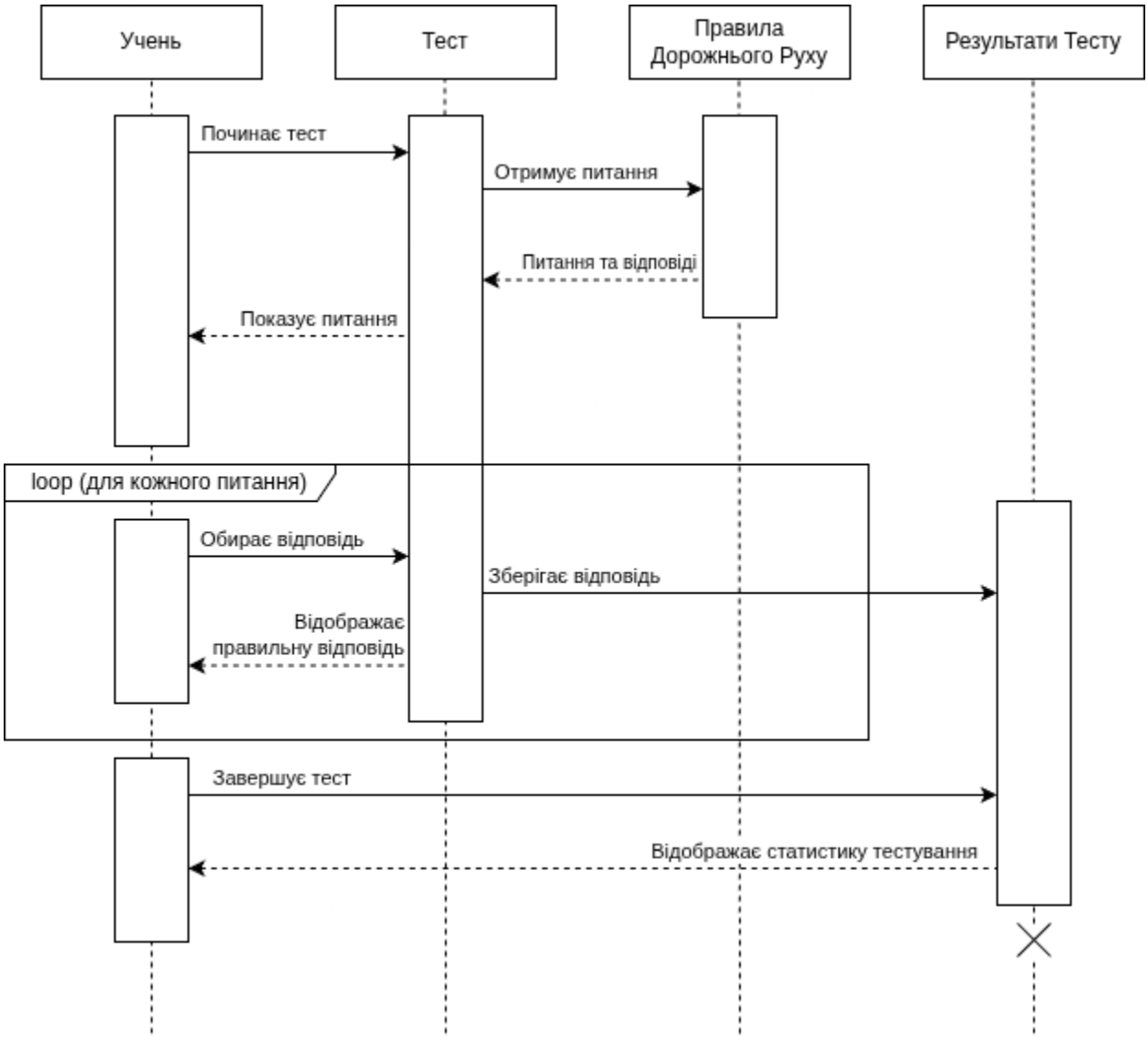

Рис 1.6. Діаграма послідовності

Послідовність взаємодій:

- **Початок тесту** - учень ініціює процес тестування, відправляючи запит на початок тесту
- **Отримує питання** - система надає користувачу питання для відповіді
- **Цикл відповідей** - реалізується повторюваний процес:
  - Учень обирає відповідь на питання
  - Система перевіряє правильність відповіді через компонент ПДР
  - Система зберігає результат відповіді
  - Процес повторюється для наступного питання
- **Завершення тесту** - після відповіді на всі питання учень завершує тест
- **Отримання статистики** - система генерує та відображає статистику тестування користувачу

Обидві діаграми разом надають уявлення про функціональність системи та її динамічну поведінку, що є основою для подальшого проектування архітектури та реалізації програмного забезпечення.

1.5 Висновок до розділу 1

Після дослідження предметної області до системи навчання та контролю знань з правил дорожнього руху, було зроблено висновок, що однією з головних проблем традиційного навчання є недостатня мотивація. Отже нова система з елементами гейміфікації стане чудовим рішенням для покращення рівня засвоєння знань через молоді.

Опрацьований аналіз існуючих рішень продемонстрував чітке поділ ринку на два сегменти: це дуже гейміфіковані освітні платформи, які не спеціалізуються на ПДР (Duolingo), та спеціалізовані системи для вивчення правил дорожнього руху, без гейміфікації (Green Way, Vodiy.ua, ПДР 2025). Це надає унікальну можливість розробити інноваційний продукт, де поєднають

узагальнену специфіку правил дорожнього руху і сучасні способи інтерактивного та мотиваційного навчання.

Визначені функціональні та нефункціональні вимоги до системи забезпечують проектування надійного, безпечного та продуктивного веб-застосунку, що зможе обслуговувати одночасно велику кількість користувачів при збереженні високої якості навчального процесу. Особливу увагу в проєкті віддали роботі системи на різних пристроях і браузерах, що значно розширить аудиторію потенційних користувачів.

Моделювання предметної області за допомогою UML-діаграм дозволяло чітко структурувати функціональність системи, визначити основні взаємодії між її компонентами. Діаграма прецедентів описувала ключові ролі користувачів і їх можливості в системі, а діаграма послідовності деталізувала спосіб проходження тестування як основної функції додатку.

Отже, дослідження підтверджує актуальність і доцільність розробки гейміфікованої системи навчання ПДР, у якій можна буде суттєво підвищити ефективність підготовки майбутніх водіїв та сприяти покращенню дорожньої безпеки в Україні.

2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Одним з основних етапів у процесі створення програмного забезпечення є побудова логічної моделі даних, що відображає ключові елементи предметної області. Ця модель дозволяє глибоко проаналізувати структуру інформації, яку обробляє система, встановити взаємозв’язки між основними об'єктами, а також сформувати чітке бачення майбутньої реалізації на рівні бази даних.

Для візуалізації логічної структури даних традиційно використовується ER-діаграма (від англ. Entity-Relationship Diagram), яка дає змогу відобразити сутності, їх атрибути, а також типи зв’язків між ними. Така діаграма є універсальним засобом для представлення даних і зрозуміла як технічним спеціалістам, так і бізнес-аналітикам.

У межах даного проєкту було сформовано ER-діаграму, яка охоплює основні компоненти інформаційної системи, орієнтованої на навчання та перевірку знань правил дорожнього руху. Її побудову було здійснено з урахуванням вимог реляційної моделі даних і принципів нормалізації до третьої нормальної форми. Це забезпечує відсутність надлишкових залежностей і знижує ризик виникнення аномалій під час оновлення чи видалення інформації.

До ключових сутностей системи належать:

Користувачі (**users**) - містить інформацію про зареєстрованих користувачів:

- user_id - унікальний ідентифікатор користувача
- username - ім'я користувача для входу в систему
- email - електронна адреса користувача
- password_hash - захешований пароль для безпеки
- is_admin - прапорець, що визначає права адміністратора

Статистика користувачів (**user_stats**) - сутність для зберігання геймфікованих показників користувача:

- user_id (PK, FK) - зв'язок з таблицею користувачів
- xp - очки досвіду користувача
- daystreak - кількість днів підряд навчання
- last_completion_date - дата останнього завершення завдання

Питання (**questions**) - сутність, що містить банк питань для тестування:

- question_id (PK) - унікальний ідентифікатор питання
- question_title - текст питання
- image_url - посилання на зображення (якщо необхідно)
- answers_json - варіанти відповідей у JSON форматі
- explanation - пояснення правильної відповіді

Теми (**topic**) - сутність для категоризації питань за тематичними розділами:

- topic_id (PK) - унікальний ідентифікатор теми
- title - назва теми
- order_index - порядок відображення тем

Тести (**test**) - сутність для зберігання інформації про згенеровані тести:

- test_id (PK) - унікальний ідентифікатор тесту
- topic_id (FK) - зв'язок з темою тесту
- order_index - порядок питань у тесті
- is_exam - прапорець, що визначає чи є тест екзаменаційним

Допоміжні сутності

Результати тестів користувачів (**user_tests**) - сутність для зберігання результатів проходження тестів:

- user_id (FK) - посилання на користувача
- test_id (FK) - посилання на тест
- topic_id (FK) - посилання на тему
- is_completed - статус завершення тесту
- timestamp - час проходження тесту

Відповіді користувачів (**user_answers**) - зберігання відповідей користувача:

- user_answer_id (PK) - унікальний ідентифікатор відповіді
- user_id (FK) - посилання на користувача
- question_id (FK) - посилання на питання
- is_correct - правильність відповіді
- timestamp - час надання відповіді

Зв'язок тестів і питань (**test_questions**) - проміжна сутність для реалізації зв'язку багато-до-багатьох між тестами та питаннями:

- question_id (FK) - посилання на питання
- test_id (FK) - посилання на тест
- topic_id (FK) - посилання на тему

ER-діаграма цієї структури представлена на рисунку 2.1. Вона ілюструє взаємодію між компонентами системи та дає змогу легко переходити до етапу фізичного проєктування, а саме написання SQL-запитів для створення таблиць.

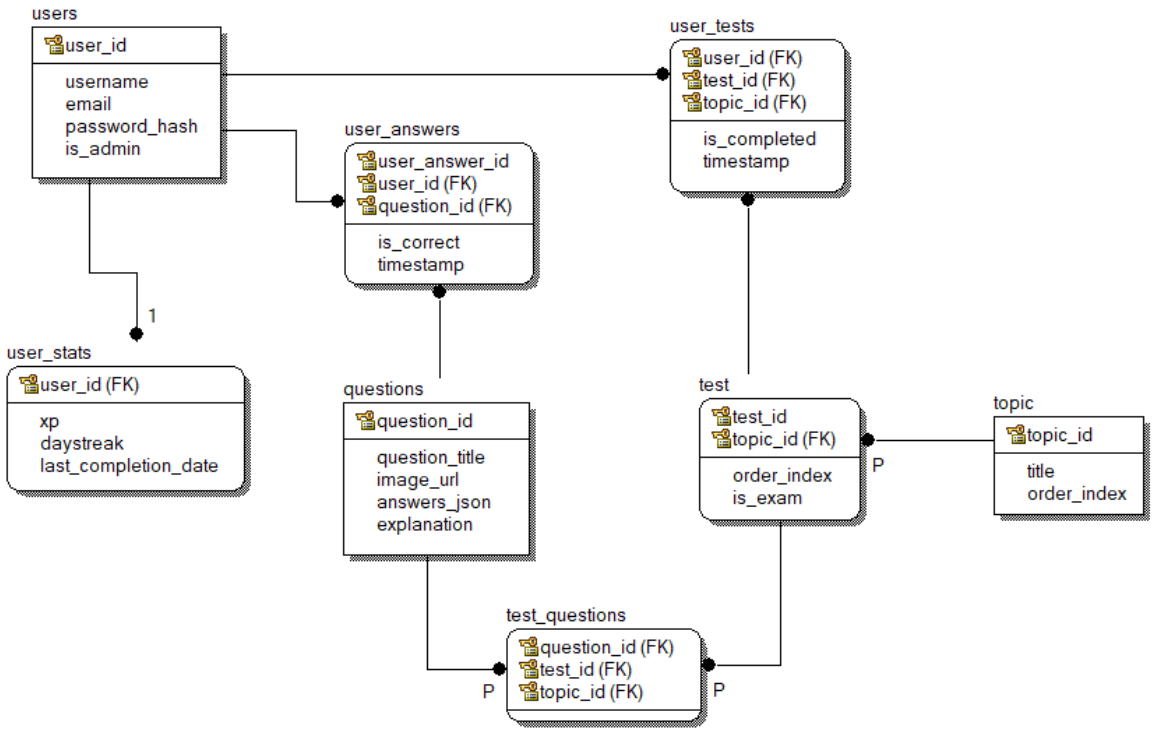

Рис 2.1. ER-діаграма бази даних

2.2 Діаграма класів та кооперацій

У процесі розробки архітектури програмної системи навчання ПДР важливим є створення діаграм класів та кооперацій. Ці діаграми становлять основу для подальшої технічної реалізації системи, оскільки забезпечують чітке розуміння структури компонентів, їх взаємодії та розподілу відповідальностей у межах застосунку.

Діаграма класів є одним з найважливіших інструментів об'єктно-орієнтованого проектування. Вона відображає основні класи системи, їх атрибути та методи, а також демонструє відношення між класами. Саме через діаграму класів визначається, які об'єкти функціонуватимуть у системі, як вони взаємодіятимуть між собою та яку логіку і дані кожен з них містить.

На рисунку 2.2 зображена діаграма класів для проєктованої системи.

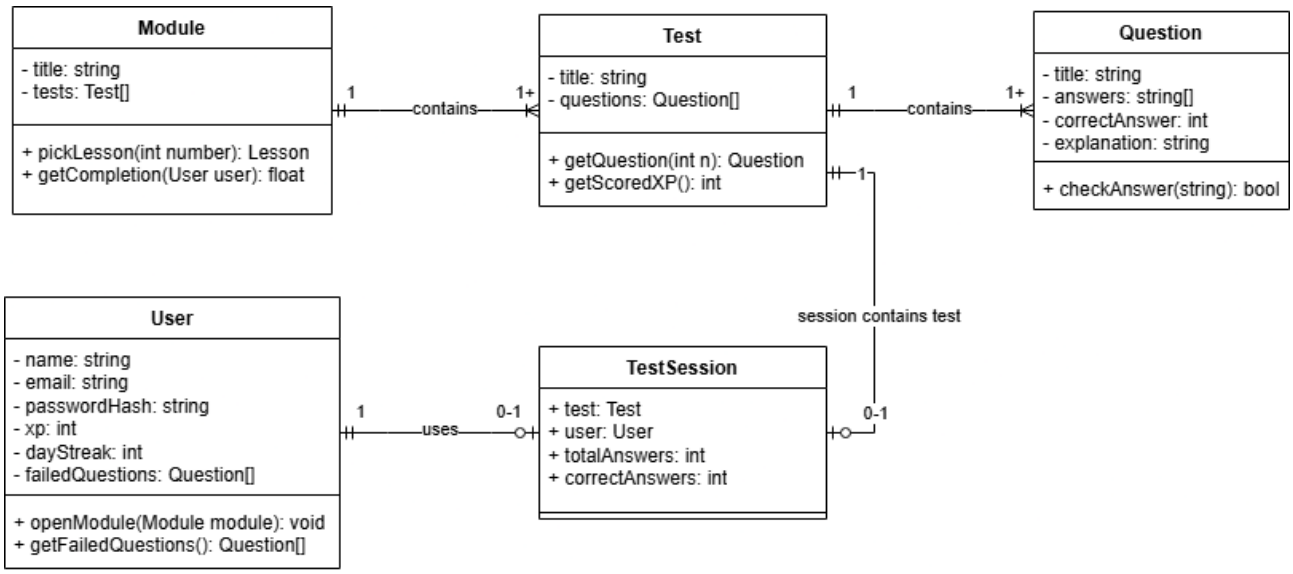

Рис 2.2. Діаграма класів

User (Користувач) - клас з даними користувача, що містить:

- **Атрибути**: *name*, *email*, *passwordHash*, *xp*, *dayStreak* для зберігання особистої інформації та геймфікованих показників
- **Методи**: *openModule*(), *getFailedQuestions*() для навігації по модулях та аналізу помилок

Module (Модуль) - клас для структурування навчального контенту:

- **Атрибути**: *title*, *tests* для організації тематичних розділів
- **Методи**: *pickLessonInt*(), *getCompletion*() для управління уроками та відстеження прогресу
- Зв'**язки**: композиція з *Test* (модуль містить тести) та агрегація з *Lesson*

Test (**Тест**) - основний функціональний клас для проведення тестування:

- **Атрибути**: *title, questions* для зберігання тестової інформації
- **Методи**: *getQuestion*(), *getScore*() для управління процесом тестування
- Зв'**язки**: містить множину *Question* та використовується в *TestSession*

Question (**Питання**) - клас для зберігання питань та перевірки відповідей:

- **Атрибути**: *title, answers_string, correctAnswer_int, explanation*
- **Методи**: *checkAnswer*() для валідації відповідей користувачів

TestSession (**Сесія тестування**) - клас для управління процесом проходження тестів:

- **Атрибути**: *test, user, totalAnswers, correctAnswers* для відстеження поточної сесії
- Зв'**язки**: пов'язує користувача з конкретним тестом під час його проходження

Діаграма кооперацій (або колаборацій) є інструментом для моделювання взаємодії між об'єктами в межах конкретних сценаріїв використання. Вона акцентує увагу не тільки на структурних зв'язках, а й на процесах співпраці між об'єктами під час виконання певних операцій.

На рисунку 2.3 представлено діаграму з трьома сценаріями взаємодії користувача з системою.

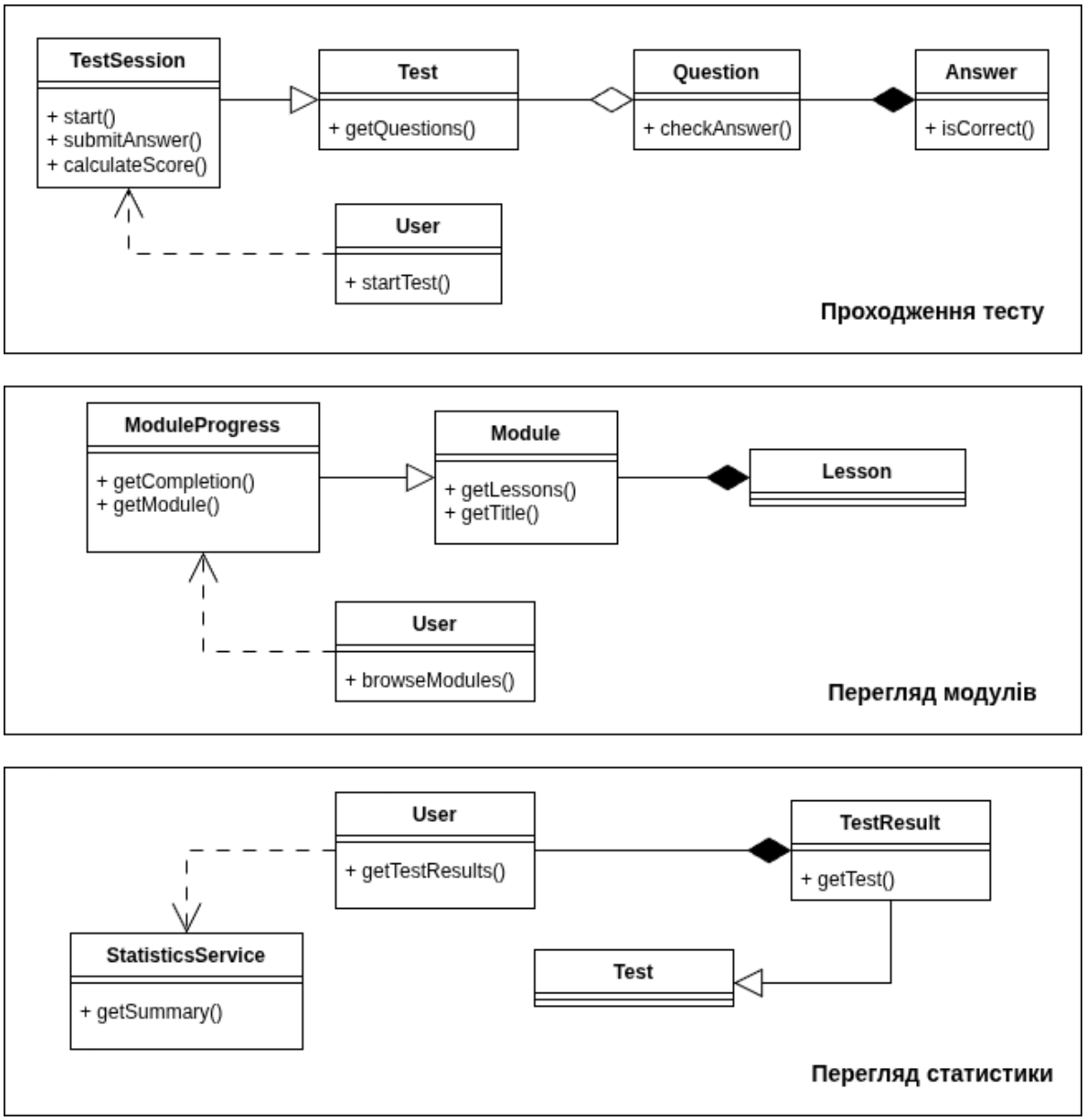

Рис 2.3. Діаграма кооперацій

Сценарій "Проходження тесту"

- **User** ініціює початок тесту через метод *startTest*()
- **TestSession** викликає метод *start*() для початку сесії
- **Test** надає питання через *getQuestions*()
- **Question** перевіряє правильність відповіді через *checkAnswer*()
- **Answer** повертає результат перевірки через *isCorrect*()
- **TestSession** обчислює фінальний результат за допомогою *calculateScore*()

Сценарій "Перегляд модулів"

- **User** запитує список модулів через *browseModules*()
- **ModuleProgress** отримує інформацію про завершення через *getCompletion*()
- **Module** надає дані про уроки за допомогою *getLessons*() та назву через *getTitle*()
- **Lesson** відображає детальну інформацію про урок

Сценарій "Перегляд статистики"

- **User** запитує результати тестів через *getTestResults*()
- **StatisticsService** формує узагальнену статистику за допомогою *getSummary*()
- **TestResult** надає детальні дані про конкретний тест через *getTest*()
- **Test** надає додаткову інформацію про тестування

Ці діаграми кооперацій показують як різні компоненти системи взаємодіють для реалізації ключових функцій: проходження тестів, навігації по навчальним модулям та аналізу статистики успішності.

Використання діаграм класів і кооперацій являє собою критичний аспект проектування архітектури програмних систем. У разі істотних структурних і поведінкових аспектів системи ці діаграми дозволяють уявити потенційні проблеми архітектури ще до початку кодування.

2.3 Діаграма пакетів

Діаграма пакетів (Package Diagram) є одним з типів структурних UML-діаграм, призначених для відображення організації системи на високому рівні абстракції. Вона показує, як система розділена на логічні групи або модулі (пакети), демонструє залежності між цими пакетами та їх внутрішню структуру. Діаграма пакетів особливо корисна для великих і складних систем, оскільки дозволяє зрозуміти загальну архітектуру без заглиблення в деталі реалізації окремих компонентів.[5]

На діаграмі пакетів представленій на рисунку 2.4 зображена класична архітектура веб-застосунку, що розділена на два основні рівні: клієнтський (Client) та серверний (Server). Така архітектура забезпечує чітке розділення відповідальностей та дозволяє незалежно розвивати різні частини системи.

Клієнтська частина (Client)

Клієнтська частина системи організована за принципом Model-View архітектури та містить наступні пакети:

UI Components (Компоненти інтерфейсу користувача) - пакет який містить всі візуальні компоненти інтерфейсу користувача:

- **ProgressPage** - сторінка відображення прогресу навчання користувача, статистики та досягнень
- **TestPage** - сторінка проходження тестів з інтерактивними питаннями та варіантами відповідей
- **AuthPage** - сторінка аутентифікації та авторизації користувачів

View Models (Моделі представлення) - логіка управління станом та бізнес-логікою клієнтської частини:

- **ProgressModel** - управляє даними про прогрес користувача, обчислює статистику та відстежує досягнення

[5] Visual Paradigm. What is Package Diagram? [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/

- **TestModel** - керує процесом тестування, валідацією відповідей та підрахунком результатів
- **AuthModel** - відповідає за процеси входу, реєстрації та управління сесією користувача

REST API Client - компонент для взаємодії з серверною частиною, що забезпечує:

- Відправку HTTP-запитів до серверних контролерів
- Обробку відповідей та помилок
- Серіалізацію та десеріалізацію даних
- Управління токенами аутентифікації

Серверна частина (Server)

Серверна частина включає такі пакети:

Controllers (Контролери) - містить контролери, що обробляють HTTP-запити та координують роботу системи:

- **Progress Controller** - обробляє запити, пов'язані з відображенням прогресу навчання, статистикою та досягненнями користувачів
- **Test Controller** - керує логікою тестування, генерацією питань, перевіркою відповідей та збереженням результатів
- **Auth Controller** - відповідає за аутентифікацію, авторизацію, реєстрацію користувачів та управління сесіями

Database Controller та Database

- **Database Controller** - забезпечує абстракцію для роботи з базою даних
- **Database** - фізичне сховище даних системи, що містить всю інформацію про користувачів, тести, питання та результати

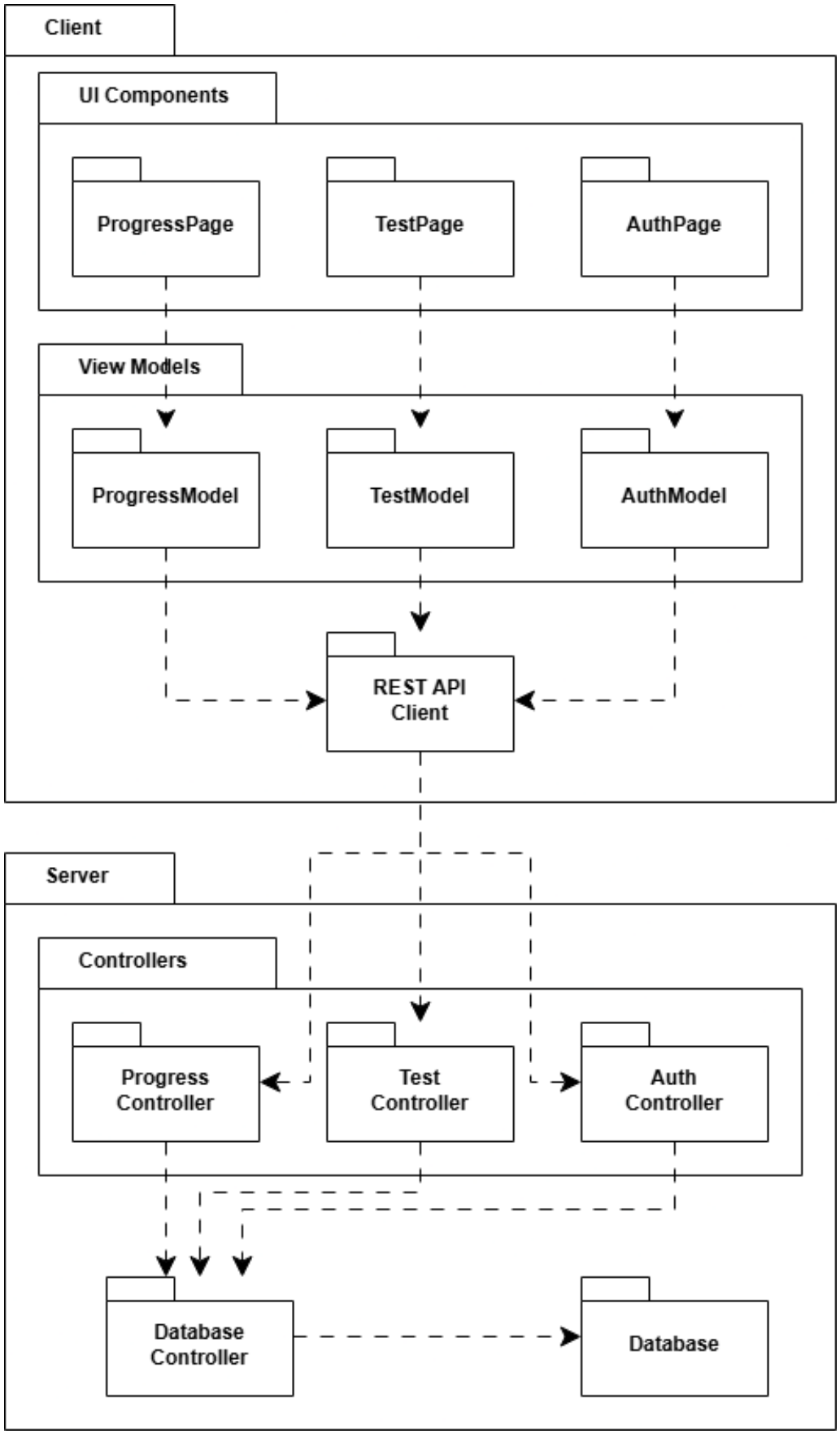

Рис 2.4. Діаграма пакетів

2.4 Діаграма компонентів

Діаграма компонентів (Component Diagram) є UML-діаграмою, яка показує організацію та залежності між програмними компонентами системи. Вона відображає фізичні компоненти системи, їх інтерфейси та зв'язки між ними на рівні реалізації. Діаграма компонентів особливо корисна для розуміння архітектури системи з точки зору розгортання та інтеграції різних модулів.

Діаграма компонентів зображена на рисунку 2.5 показує архітектуру веб-застосунку з складається з трьох рівнів, а саме: **рівень презентації** (**User Interface**), **бізнес-логіки** (**REST API**) та **рівня даних** (**Database** та **Utils**).

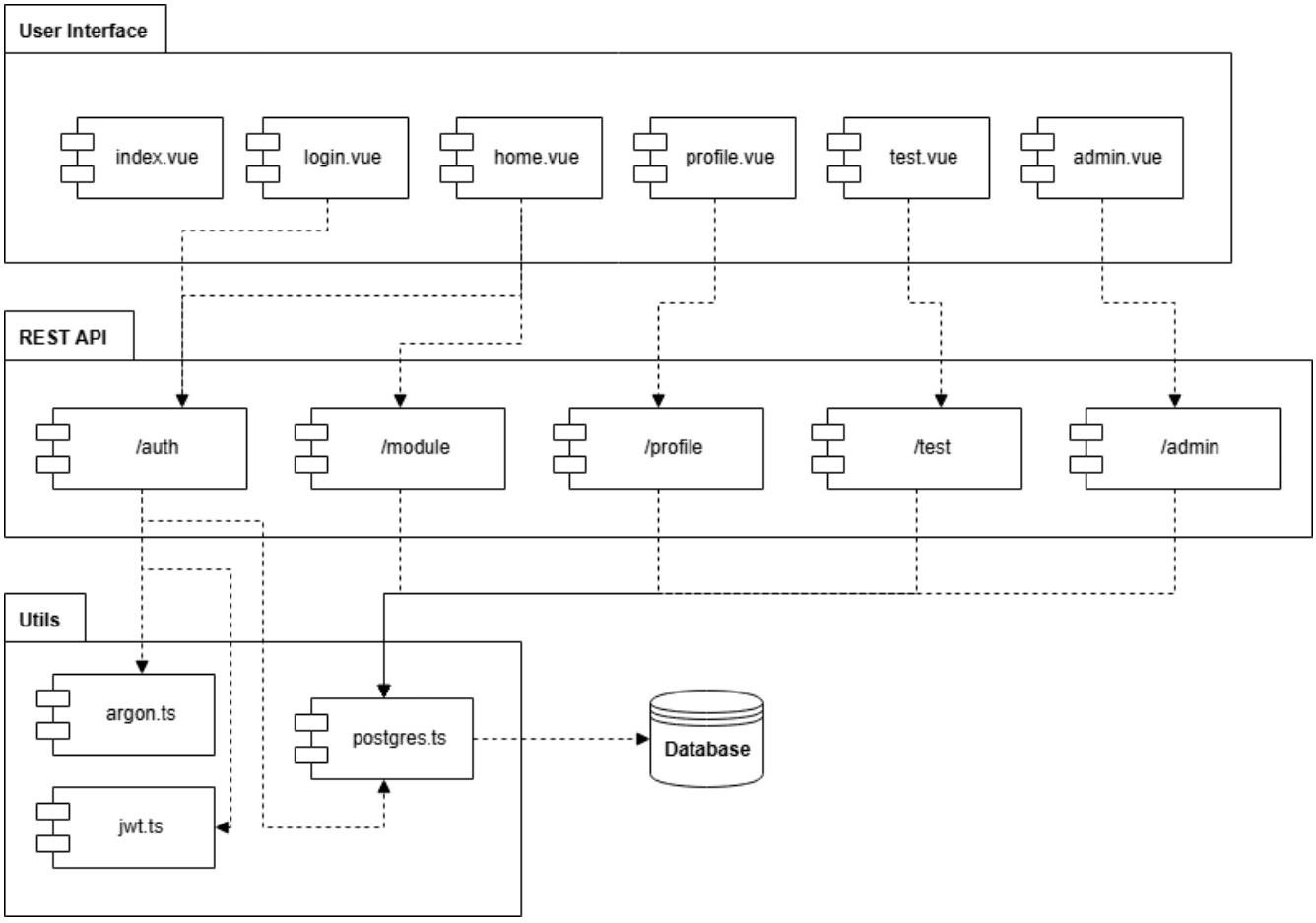

Рис 2.5. Діаграма компонентів

Рівень презентації (User Interface)

Рівень користувацького інтерфейсу містить компоненти, що відповідають за взаємодію з користувачем:

- **index.vue** - компонент застосунку, який являє собою домашню сторінку застосунку.
- **login.vue** - компонент автентифікації користувачів, що надає форми для входу в систему та реєстрації нових акаунтів. Забезпечує валідацію даних користувача та інтеграцію з системою авторизації.
- **home.vue** - головна сторінка застосунку після успішної автентифікації, що відображає дашборд користувача з основною навігацією по модулях тестів.
- **profile.vue** - компонент профілю користувача, що дозволяє переглядати та редагувати персональну інформацію, налаштування акаунту та персональну статистику навчання.
- **test.vue** - інтерактивний компонент для проходження тестів з ПДР, що забезпечує відображення питань, обробку відповідей користувача та показ результатів.
- **admin.vue** - адміністративний компонент для користувачів з правами адміністратора, що надає інструменти для управління контентом, користувачами та системними налаштуваннями.

Рівень бізнес-логіки (REST API)

Середній рівень архітектури представлений REST API компонентами, що обробляють бізнес-логіку та забезпечують комунікацію між клієнтом і сервером:

- **/auth** - компонент автентифікації та авторизації, що обробляє запити на вхід, реєстрацію, верифікацію токенів та управління сесіями користувачів.
- **/module** - компонент управління навчальними модулями, що забезпечує операції для модулів, відстеження прогресу вивчення та організацію навчального контенту.
- **/profile** - компонент профілю, що дозволяє отримувати .

- **/test** - компонент тестування, що завантажує тести, обробляє відповіді користувачів, підраховує результати та зберігає статистику проходження.
- **/admin** - адміністративний API компонент, що надає функціональність для управління системою.

Рівень даних та утиліт (Utils)

Нижній рівень архітектури містить допоміжні компоненти та систему зберігання даних:

- **argon.ts** - утилітний компонент для роботи з криптографічним алгоритмом Argon2, що забезпечує безпечне хешування паролів користувачів. Цей компонент є критично важливим для забезпечення безпеки системи.
- **jwt.ts** - компонент для роботи з JSON Web Tokens, що відповідає за генерацію, валідацію та верифікацію JWT токенів для автентифікації користувачів та підтримки сесій.
- **postgres.ts** - компонент для взаємодії з базою даних PostgreSQL, що забезпечує підключення до бази даних, виконання запитів та управління транзакціями.

Database - фізична база даних PostgreSQL, що зберігає всі дані системи: інформацію про користувачів, тести, питання, результати та статистику.

2.5 Висновок по розділу 2

У межах другого розділу було здійснено повноцінне проєктування архітектури майбутньої системи навчання та контролю знань з правил дорожнього руху. Було створено логічну модель даних у вигляді ER-діаграми, яка охоплює основні сутності системи та відображає взаємозв’язки між ними. Це дало змогу на ранньому етапі врахувати всі структурні особливості інформаційної бази та підготувати ґрунт для ефективного збереження та обробки даних.

За допомогою UML-діаграм було спроєктовано як структуру системи (діаграма класів, діаграма компонентів), так і взаємодію між її частинами (діаграма кооперацій, діаграма послідовності). Також було побудовано діаграму пакетів, що дозволила відобразити логічний поділ на клієнтську і серверну частину та залежності між модулями.

Особливу увагу приділено побудові модульної архітектури, яка забезпечує легку підтримку, масштабування та розширення функціональності у майбутньому. Важливим результатом є також розділення рівнів взаємодії (UI, REST API, база даних), що відповідає принципам сучасного веб-розробництва. Отже, проєктування створило надійний фундамент для реалізації функціональних компонентів системи.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Одним з важливих етапів розробки програмного продукту є вибір системи управління базами даних (СУБД), для забезпечення збереження, доступу та захисту інформації.

У межах даного проєкту було обрано **PostgreSQL** як основну систему управління базами даних. Це безкоштовна та сучасна об’єктно-реляційна СУБД з відкритим вихідним кодом яка є дуже популярною для подібних проєктів.

До переваг PostgreSQL можна віднести:

- Повна підтримка стандарту SQL разом із вкладеними запитами, тригерами, процедурами;
- Розширені типи даних такі як JSON, масиви, геолокації а також можливість створення власних типів[6];
- Активна спільнота з великою кількістю ресурсів та документації;

Порівняємо PostgreSQL з іншими СУБД, а саме **MySQL**, **Microsoft SQL Server**, **MongoDB** та **SQLite**.

MySQL

Переваги:

- Достатньо швидка при виконанні SELECT запитів;
- Дуже поширена серед хостинг-провайдерів;

Недоліки:

- Менш сувора реалізація стандарту SQL;
- Гірша підтримка типів даних та транзакцій порівняно з PostgreSQL;

[6] PostgreSQL: Documentation [Електронний ресурс]. Режим доступу - https://www.postgresql.org/docs/

Microsoft SQL Server

Переваги:

- Інтеграція з системами розробленими Microsoft, такі як .NET, Azure, Excel, тощо;
- Потужні інструменти адміністрування;

Недоліки:

- Повна версія потребує платної ліцензії;
- Орієнтована на ОС Windows, що значно звужує вимоги до сервера (недавно з’явилась підтримка Linux, але не на повному рівні);
- Не підходить для проєктів з відкритим вихідним кодом;

MongoDB

Переваги:

- Добре підходить для неструктурованих даних;
- Горизонтальне масштабування підтримується “з коробки”;

Недоліки:

- Не містить підтримки реляційності – зв’язків між таблицями;
- Великі запити більш складні для реалізації через не чітку структуру;

SQLite

Переваги:

- Дуже легка, вбудована СУБД, яка не потребує сервера, оскільки всі дані зберігаються у файлі;
- Ідеальна для програм які потребують збереження даних безпосередньо на пристрої користувача;
- Повністю відкрита та безкоштовна;

Недоліки:

- Обмежений багатокористувацький доступ – не підходить для одночасної обробки великої кількості запитів;
- Не має підтримки розподілення навантаження на декілька серверів.

3.2 Вибір інструментарію для створення прикладного програмного забезпечення

Після того, як ми визначилися з базою даних, настав час обрати технології для розробки самого веб-застосунку. Сьогодні вибір справді великий — є десятки фреймворків і інструментів, кожен із яких має свої плюси й мінуси. Важливо підібрати такий стек, який не лише дозволить швидко почати розробку, але й буде зручним у підтримці, масштабованим і не стане головним болем у майбутньому.

Під час аналізу були враховані різні аспекти: рівень підтримки спільнотою, активність розробки, зручність інтеграції з іншими інструментами, продуктивність, крива навчання та наявність документації. На основі цих факторів був обраний стек, який включає **Nuxt.js** для фронтенд-частини, **TypeScript** для основної логіки та **PostgreSQL** у якості системи керування базами даних. Усі ці компоненти мають сильну репутацію, активно підтримуються й чудово поєднуються один з одним.

Серед основних аргументів на користь Nuxt.js можна виділити такі[7]:

- **Можливість серверного рендерингу** (**SSR**). Це забезпечує швидший час першого завантаження сторінки, що особливо важливо для мобільних пристроїв та повільніших мереж, а також покращує індексацію сайту пошуковими системами.
- **Автоматична маршрутизація**. Nuxt генерує маршрути на основі структури файлів, що значно пришвидшує процес розробки.

[7] Introduction – Get Started with Nuxt [Електронний ресурс]. – Режим доступу: https://nuxt.com/docs/getting-started/introduction

- **Підтримка TypeScript із коробки**. Це дозволяє писати типобезпечний код із покращеною підтримкою інструментів для розробників (наприклад, автодоповнення в IDE).
- **Вбудований API** (**server API routes**). У Nuxt 3 реалізовано підтримку серверних маршрутів, що дозволяє обробляти запити до бекенду прямо в межах того самого проєкту.
- **Гнучка структура і модульна архітектура**. Існує безліч готових модулів, які легко підключаються: автентифікація, робота з базами даних, аналітика, UI-бібліотеки, тощо.

Порівняння з альтернативними рішеннями

Next.js

Це аналог Nuxt.js у світі React. Він також підтримує SSR, має модульну структуру та розвинену екосистему. Серед переваг Next — велика спільнота, швидкий розвиток та активна підтримка.

Проте:

- React має вищий поріг входження, особливо для тих, хто не має досвіду з JSX.
- Кількість шаблонного коду більша, що ускладнює підтримку невеликого проєкту.
- Через особливості React продуктивність на фронтенді може бути трохи нижчою в порівнянні з Vue.

SvelteKit

Інноваційний фреймворк, що пропонує дуже високу продуктивність та мінімальне використання ресурсів. Його основна перевага — компіляція компонентів у чистий JavaScript без runtime-бібліотек.

Проте:

- Проєкт ще досить молодий і нестабільний.
- Невелика кількість модулів і менша спільнота.

- Деякі інструменти й підходи ще не до кінця стандартизовані.

Angular

Потужний фреймворк, який містить усе необхідне для створення масштабних проєктів. Має вбудовані рішення для форм, роутінгу, валідації, сервісів та інше. Проте:

- Занадто важкий для невеликого або середнього проєкту.
- Складна структура, багато шаблонного коду.
- Висока крива навчання для нових розробників.

З огляду на ці фактори, Nuxt.js виглядав найбільш збалансованим вибором — з оптимальним поєднанням простоти, потужності та гнучкості.

Вибір бекенд-технологій

Для бекенд-частини використовувався вбудований API-фреймворк у Nuxt.js, який дозволяє створювати серверні маршрути на Node.js з використанням TypeScript. Це значно спрощує розгортання, зменшує кількість окремих сервісів у проєкті та дозволяє писати як фронтенд, так і бекенд у межах одного середовища.

Використання TypeScript

TypeScript було обрано через його типобезпечність, що зменшує кількість помилок на етапі розробки та забезпечує кращу інтеграцію з інструментами розробника. Також він дозволяє спільно використовувати типи між клієнтом і сервером, що робить код більш узгодженим.

Підключення до PostgreSQL

Для підключення до бази даних використовується бібліотека postgres, яка забезпечує простий інтерфейс для виконання SQL-запитів без використання важких ORM. Це дозволяє мати повний контроль над тим, як саме відбувається

взаємодія з базою даних, уникнути зайвих обгорток і підвищити продуктивність застосунку.

Інші технології, які використовувались

- **Bootstrap 5**: для побудови UI-інтерфейсу — легкий, адаптивний фреймворк, що добре інтегрується з Vue-компонентами.
- **Argon2**: алгоритм хешування паролів — сучасний, безпечний і рекомендований до використання OWASP.
- **jsonwebtoken (JWT)**: бібліотека для генерації та перевірки токенів авторизації — дозволяє реалізувати систему доступу без необхідності зберігати сессії на стороні сервера.

Таким чином, обраний технологічний стек — Nuxt.js + TypeScript + PostgreSQL — є сучасним, гнучким і добре збалансованим рішенням для розробки вебзастосунку.

3.3 Розробка інформаційної бази

Оскільки вже була обрана СУБД, а також спроєктована ER-діаграма, залишилось лише написати SQL-запити для створення необхідних таблиць. На рисунку 3.1 показано код для створення трьох таблиць. Повний SQL код можна знайти у Додатку А.

```sql
CREATE TABLE IF NOT EXISTS users
(
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    is_admin BOOLEAN DEFAULT FALSE
);

CREATE TABLE IF NOT EXISTS user_stats
(
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    xp INT DEFAULT 0,
    daystreak INT DEFAULT 0,
    last_completion_date TIMESTAMP DEFAULT '1970-01-01 00:00:00',
    PRIMARY KEY (user_id)
);

CREATE TABLE IF NOT EXISTS topic
(
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    description TEXT,
    order_index INT NOT NULL DEFAULT 0
);
```

Рис 3.1. SQL-запити для створення таблиць

У результаті ми маємо базу даних яка повністю готова до роботи. За допомогою клієнтської програми pgAdmin, яка використовується для підключення та управління серверами PostgreSQL, ми можемо побачити діаграму (рисунок 3.2) нашої бази даних зі вказанням використаних типів даних.

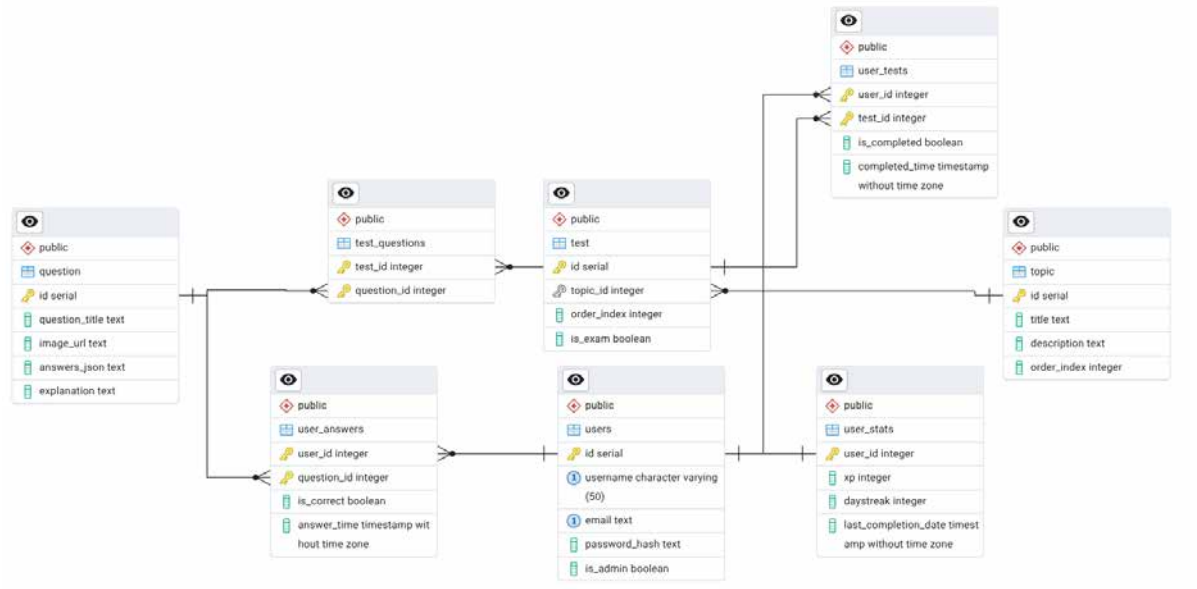

Рис 3.2. Діаграма бази даних у середовищі pgAdmin

3.4 Алгоритмізація та програмування програмних модулів

Одним з заключних етапів розробки програмної системи є саме написання алгоритмів та програмування.

Алгоритм процесу авторизації користувача

Процес авторизації є критично важливим компонентом системи, оскільки забезпечує безпечний доступ користувачів до персоналізованого контенту та відстеження їх прогресу навчання. На рисунку 3.3 представлено блок-схему алгоритму реєстрації користувача в системі.

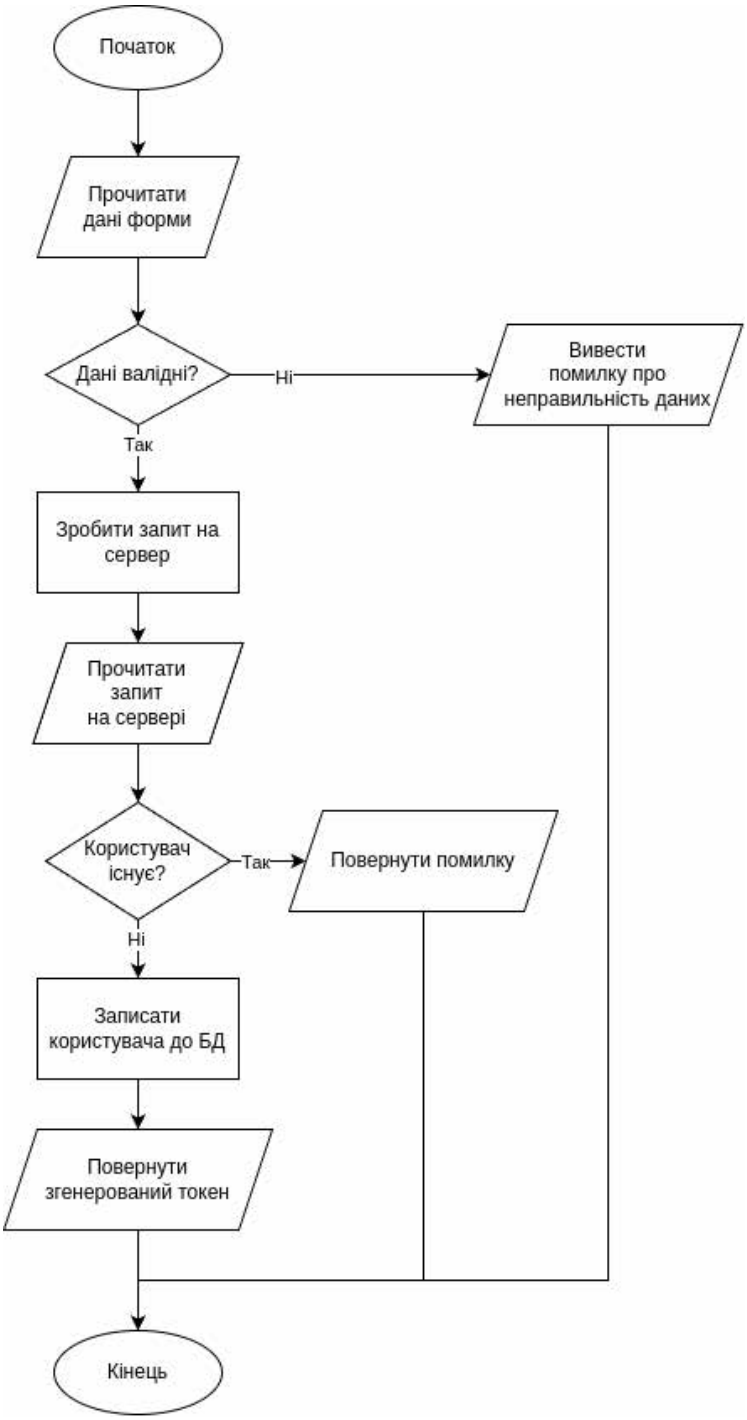

Рис 3.3. Блок-схема реєстрації користувача

Алгоритм авторизації включає наступні основні етапи:

- **Зчитування даних форми** - система отримує введені користувачем дані (email та пароль)
- **Валідація вхідних даних** - перевірка коректності формату email та наявності пароля

- **Обробка помилок валідації** - у разі некоректних даних відображається повідомлення про помилку
- **Створення запиту на сервер** - формування HTTP-запиту з даними авторизації
- **Зчитування запиту на сервері** - сервер отримує та обробляє дані авторизації
- **Перевірка існування користувача** - пошук користувача в базі даних за email
- **Повернення помилки** - якщо користувач не знайдений, повертається відповідна помилка
- **Запис користувача до БД** - оновлення інформації про останній вхід користувача
- **Повернення згенерованого токену** - створення та відправка JWT токену для подальшої авторизації. Для генерації JWT токену використовуються методи jwtSign та jwtVerify, код яких показаний на рисунку 3.4.

```
import jwt from 'jsonwebtoken';

export default function jwtSign(payload: { [key: string]: any }) {
    if (!process.env.JWT_SECRET) {
        throw new Error('JWT_SECRET is not defined');
    }

    return jwt.sign(
        payload,
        process.env.JWT_SECRET as string,
        { expiresIn: '30d' }
    );
}

export async function jwtVerify(token: string) {
    if (!process.env.JWT_SECRET) {
        throw new Error('JWT_SECRET is not defined');
    }

    return new Promise((resolve, reject) => {
        jwt.verify(token, process.env.JWT_SECRET as string, (err, decoded) => {
            if (err) {
                reject(err);
            } else {
                resolve(decoded);
            }
        });
    });
}
```

Рис 3.4. Код для генерації та перевірки JWT токену

Безпека користувацьких даних є не менш важливим фактором при розробці систем авторизації. Для забезпечення найвищого рівня захисту паролів користувачів, використовується алгоритм хешування Argon2.

Argon2 — це сучасний криптографічний алгоритм хешування паролів, який був переможцем конкурсу Password Hashing Competition у 2015 році. Він розроблений спеціально для безпечного зберігання паролів і має наступні ключові переваги[8]:

Основні характеристики Argon2:

- **Стійкість до атак грубої сили** - алгоритм спеціально розроблений для споживання значних обчислювальних ресурсів
- **Захист від атак по пам'яті** - використовує велику кількість оперативної пам'яті, що ускладнює атаки з використанням спеціалізованого обладнання
- **Рекомендації безпеки** - офіційно рекомендований OWASP (Open Web Application Security Project) для хешування паролів

Не менш важливо також забезпечити перевірку вводу даних на клієнтській частині. На рисунку 3.5 показано код який перевіряє правильність заповненої форми по кожному полю.

```
// == Validation ==
const isValidEmail = computed(() => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email.value));
const isValidUsername = computed(() => username.value.length >= 3);
const isValidPassword = computed(() => password.value.length >= 6);
const isValidConfirmPassword = computed(() => confirmPassword.value === password.value);
const isFormValid = computed(() => {
  let isValid = isValidPassword.value && isValidUsername.value;
  if (isRegistering.value) {
    isValid = isValid && isValidEmail.value && isValidConfirmPassword.value;
  }
  return isValid;
});
```

Рис 3.5. Код клієнтської частини для валідації введених даних

[8] The password hash Argon2, winner of PHC [Електронний ресурс]. – Режим доступу: https://github.com/P-H-C/phc-winner-argon2

У результаті було розроблено форму реєстрації яка підтримує валідацію, а також асинхронні запити до сервера. На рисунку 3.6 зображено скріншот користувацького інтерфейсу.

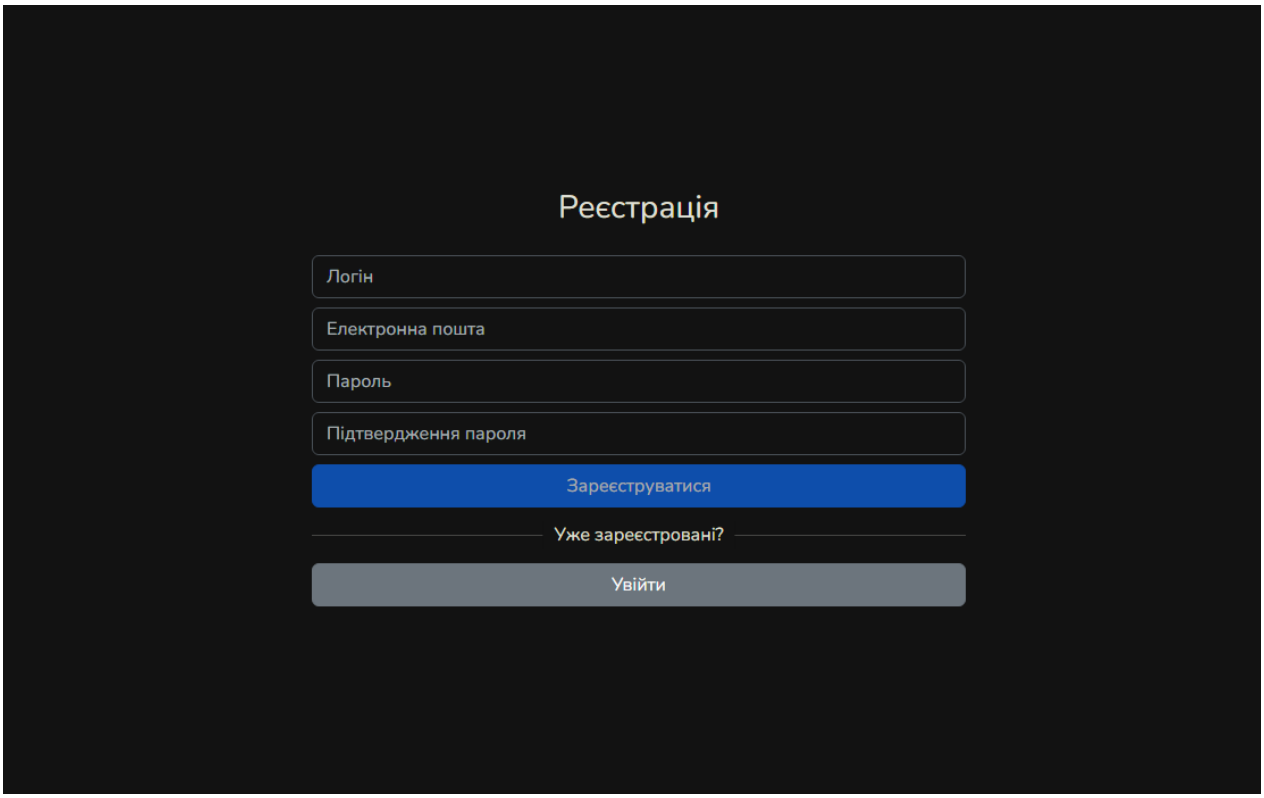

Рис 3.6. Скріншот форми реєстрації

Алгоритм отримання інформації про модуль

Для забезпечення персоналізованого навчального досвіду система повинна надавати користувачам актуальну інформацію про доступні модулі та їх стан завершення.

На рисунку 3.7 зображено блок-схему алгоритму отримання інформації про конкретний навчальний модуль.

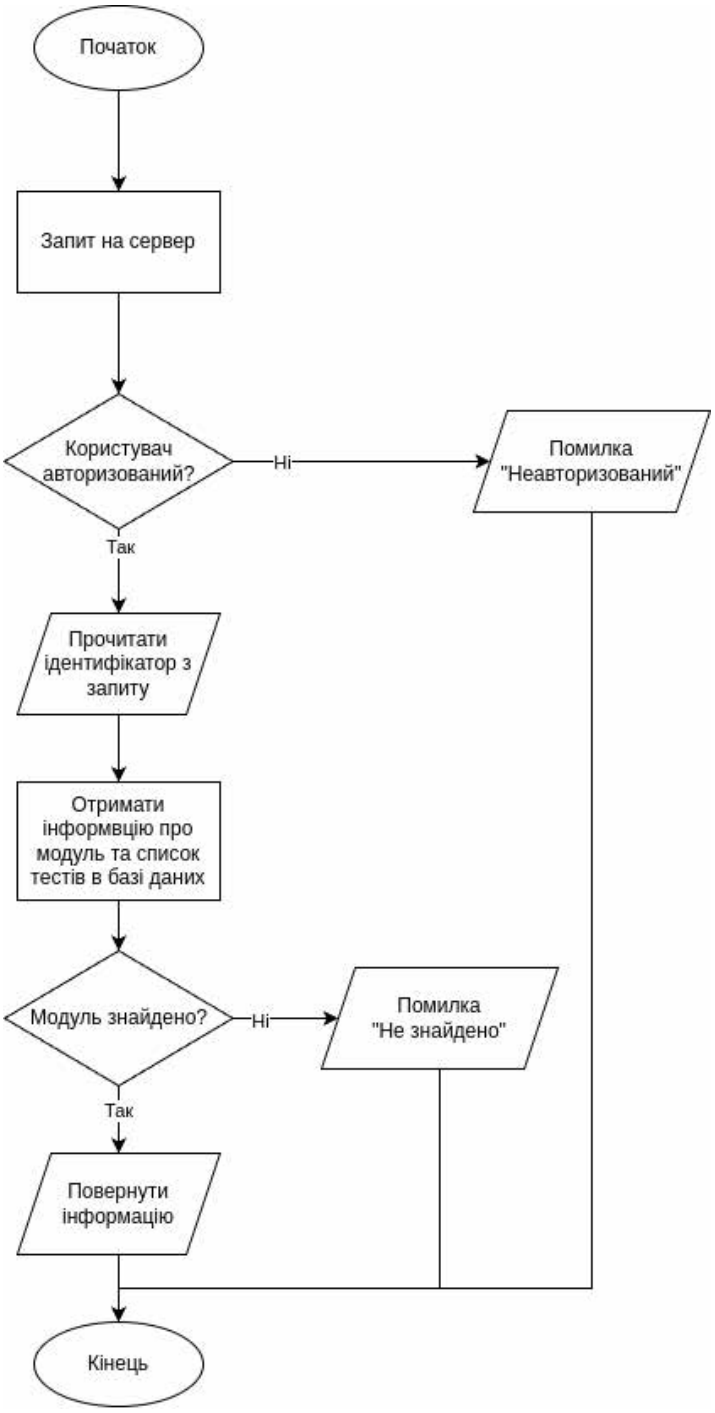

Рис 3.7. Блок схема алгоритму завантаження інформації про модуль

Процес отримання інформації про модуль складається з таких кроків:

1. **Ініціація запиту** - користувач надсилає запит на отримання інформації про модуль
2. **Перевірка авторизації** - система перевіряє валідність токену користувача
3. **Обробка помилки авторизації** - у разі невалідного токену повертається помилка "Неавторизований"

4. **Зчитування ідентифікатора з запиту** - витягнення ID модуля з параметрів запиту
5. **Отримання інформації з бази даних** - виконання SQL-запиту для отримання деталей модуля та списку тестів
6. **Перевірка існування модуля** - валідація того, що модуль з вказаним ID існує в системі
7. **Повернення помилки** - якщо модуль не знайдено, повертається помилка "Не знайдено"
8. **Повернення інформації** - відправка клієнту структурованих даних про модуль, включаючи тести та стан завершення

На рисунку 3.8 показано код який відповідає за отримання даних про модуль та список тестів у ньому. Також цей метод повертає які тести вже були пройдені користувачем у вказаному модулі.

```
async function fetchModule(event: H3Event, id: number, userId: number) {
    const db = usePostgres();

    // Get the module from the database
    const module = await db`
        SELECT topic.id                                    as topic_id,
               topic.title                                 as topic_title,
               topic.description                           as topic_desc,
               topic.order_index                           as topic_index,
               test.id                                     as question_id,
               test.order_index                            as question_index,
               test.is_exam                                as is_exam,
               coalesce(user_tests.is_completed, false) as completed
        FROM topic
                 LEFT JOIN test ON test.topic_id = topic.id
                 LEFT JOIN user_tests ON user_tests.test_id = test.id
        WHERE topic.id = ${id}
          AND user_tests.user_id = ${userId}
        ORDER BY test.order_index
    `;
    if (module.count === 0) {
        return event.respondWith(new Response('Not Found', {status: 404}));
    }

    // Return success response
    return {
        statusCode: 200,
        module: {
            id: module[0].topic_id,
            title: module[0].topic_title,
            description: module[0].topic_desc,
            index: module[0].topic_index,
            tests: module.map((test: any) => ({
                id: test.question_id,
                completed: test.completed,
                isFinal: test.is_exam,
            })) as Test[],
        } as Module,
    }
}
```

Рис 3.8. Код для отримання даних про модуль з бази даних

Алгоритм проходження тестування

Система тестування є ключовим компонентом для перевірки знань користувачів. Алгоритм проходження тесту повинен забезпечувати коректне відображення питань, обробку відповідей та підрахунок результатів.

На рисунку 3.9 представлено блок-схему алгоритму процесу тестування.

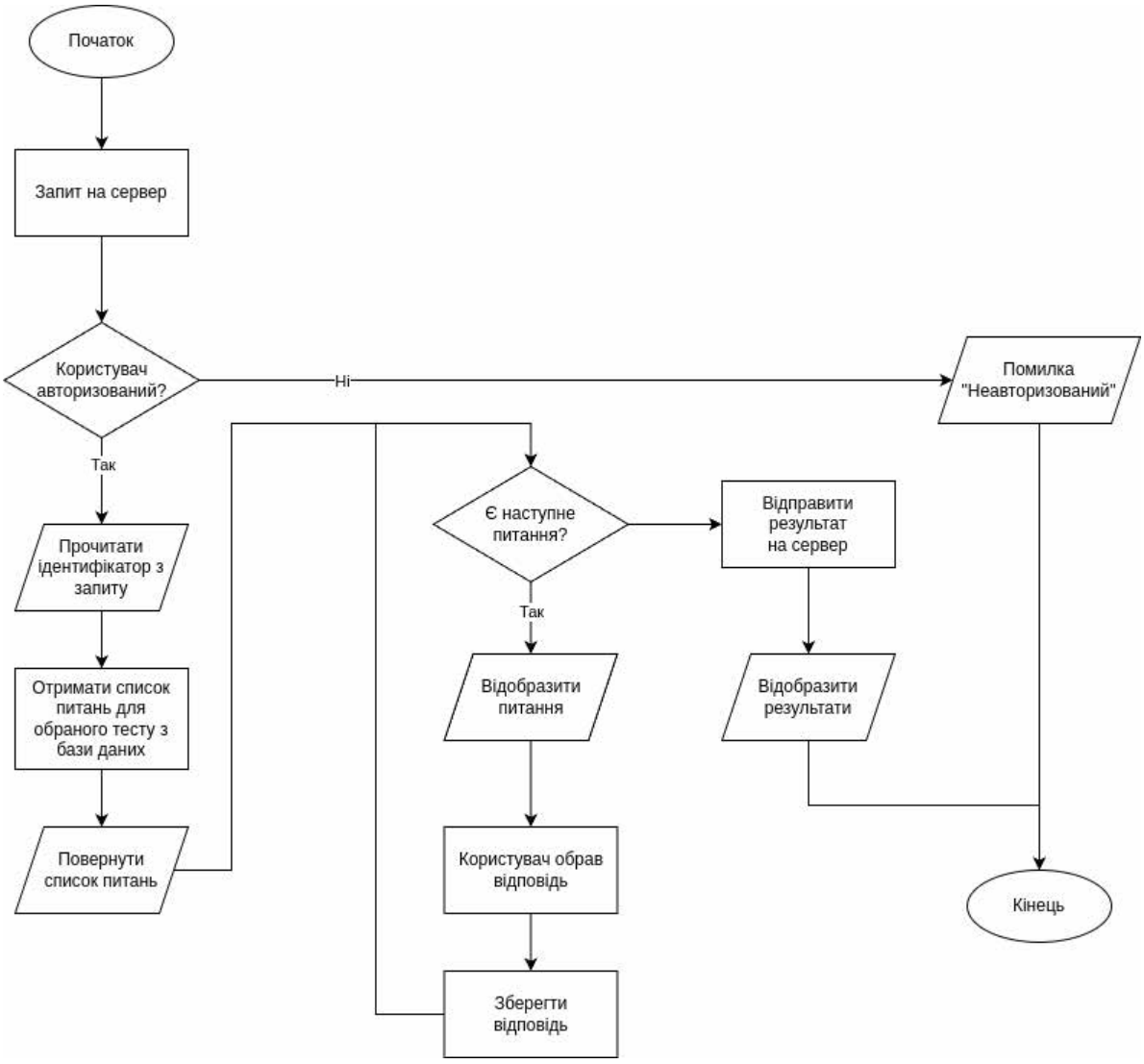

Рис 3.9. Блок схема проходження тестування

Алгоритм тестування включає наступні етапи:

1. **Запит на сервер** - надсилання запиту на отримання питань тесту
2. **Перевірка авторизації** - валідація токену користувача
3. **Обробка помилки авторизації** - повернення помилки у разі неавторизованого доступу
4. **Зчитування ідентифікатора з запиту** - отримання ID тесту з параметрів запиту
5. **Отримання списку питань** - виконання запиту до бази даних для отримання питань тесту

6. **Повернення списку питань** - відправка структурованих даних про питання клієнту
7. **Перевірка наявності питань** - валідація того, що тест містить питання
8. **Відображення питань** - показ питань користувачу на клієнтській частині
9. **Обробка відповідей користувача** - збір та валідація відповідей
10. **Відправлення результатів на сервер** - передача відповідей для обробки
11. **Збереження відповідей** - запис результатів тестування до бази даних

На рисунках 3.10 та 3.11 можна побачити процес отримання даних про тест з бази даних, а також виконання запиту з клієнтської частини з використанням Fetch API.

```
// load questions from API
const questions = ref<Question[]>([]);
const loading = ref(true);

async function load() {
  try {
    const res = await $fetch(`/api/v1/test/get?id=${id}`, {
      headers: {'Authorization': `Bearer ${useAuthStore().token}`}
    });
    console.log(res);
    questions.value = res?.questions;
  } catch (error) {
    // return back to home
    navigateTo('/home');
  } finally {
    loading.value = false;
  }
}

load();
```

Рис 3.10. Виконання запиту до REST API на клієнтській частині

```
export default defineEventHandler(async (event) => {
    // GET request
    if (event.node.req.method !== 'GET') {
        return event.respondWith(new Response('Method Not Allowed', {status: 405}));
    }

    const {err} = await verifyToken(event);
    if (err) {
        return err;
    }

    // get test id from query
    const {id} = getQuery(event);
    if (!id) {
        return event.respondWith(new Response('Bad Request', {status: 400}));
    }

    // check if it's a number
    const testId = Number(id);
    if (isNaN(testId)) {
        return event.respondWith(new Response('Bad Request', {status: 400}));
    }

    // Get the questions from the database
    const db = usePostgres();
    const questions = await db`
        SELECT question.*
        FROM test
                 LEFT JOIN test_questions ON test.id = test_questions.test_id
                 LEFT JOIN question ON test_questions.question_id = question.id
        WHERE test.id = ${testId}
    `;
    if (questions.count === 0) {
        return event.respondWith(new Response('Not Found', {status: 404}));
    }

    // Return success response
    return {
        statusCode: 200,
        questions: questions.map((question: any) => ({
            id: question.id,
            question: question.question_title,
            image_url: question.image_url,
            options: JSON.parse(question.answers_json),
            explanation: question.explanation,
        })) as Question[],
    }
});
```

Рис 3.11. Код отримання даних про тест на серверній частині

Користувацький інтерфейс тестування забезпечує зручне проходження тестів з відображенням питань, варіантів відповідей та прогресу.

На рисунку 3.12 представлено HTML-шаблон компонента тестування, а на рисунку 3.13 скріншот на якому відкрито тест.

```
<template>
  <div class="test-container" v-if="questions && questions.length > 0 && !loading">
    <div class="container">
      <div class="row mb-2">
        <div class="col-1">
          <BButton variant="link-secondary" @click="askToQuit">
            <MdiIcon icon="mdiClose"/>
          </BButton>
        </div>
        <div class="col-11 py-2">
          <ProgressBar :progress="progress"/>
        </div>
      </div>
      <div class="row question-card">
        <div class="col-12 text-center">
          <h3 class="mb-4 question-text">
            {{ question.question }}
          </h3>
          <img :src="question.image_url" alt="Question Image" class="img-fluid mb-4 question-img"
               v-if="question.image_url"/>
        </div>
      </div>
      <div class="row">
        <div class="options">
          <div v-for="(option, index) in question.options" :key="index" class="option">
            <BButton variant="secondary" @click="selectOption(index)" :class="{
                'correct': option.isCorrect && showAnswer,
                'incorrect': selectedOption === index && !option.isCorrect && showAnswer,
                'selected': selectedOption === index && !showAnswer
              }">
              {{ option.text }}
            </BButton>
          </div>
        </div>
      </div>
      <div class="row">
        <div class="text-center">
          <BButton
              variant="primary"
              @click="showAnswer = true"
              :disabled="selectedOption === -1"
              v-if="!showAnswer">
            Перевірити
          </BButton>
```

Рис 3.12. HTML-шаблон сторінки проходження тестування

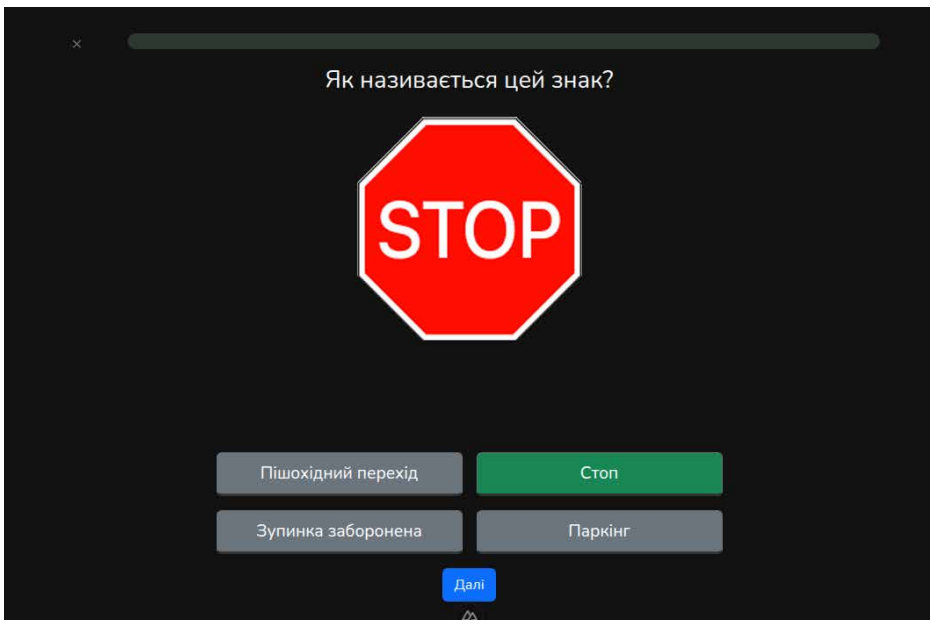

Рис 3.13. Скриншот сторінки тестування

Розроблені програмні модулі забезпечують повнофункціональну систему навчання ПДР з наступними ключовими характеристиками:

Безпека та авторизація:

- JWT-токени для stateless авторизації
- Хешування паролів з використанням Argon2
- Валідація всіх користувацьких вводів

Модульна архітектура:

- Чітке розділення клієнтської та серверної логіки
- RESTful API для взаємодії між компонентами
- Компонентний підхід у інтерфейсі користувача

Ефективність та продуктивність:

- Оптимізовані SQL-запити з JOIN операціями
- Асинхронна обробка запитів
- Реактивні компоненти для динамічного оновлення інтерфейсу

Користувацький досвід:

- Інтуїтивний інтерфейс тестування
- Відстеження прогресу навчання
- Миттєвий зворотний зв'язок при тестуванні

3.5 Висновок по розділу 3

У третьому розділі було безпосередньо реалізовано функціональну частину веб-застосунку. На основі попереднього проєктування створено базу даних PostgreSQL, яка повністю відповідає логічній моделі та забезпечує надійне зберігання інформації про користувачів, тести, результати та прогрес навчання.

Було розроблено серверну логіку авторизації з використанням сучасних методів захисту — хешування паролів через Argon2 та використання JWT-

токенів для аутентифікації. Розроблені алгоритми забезпечують безпечний доступ до системи та збереження персональних даних.

На клієнтській стороні реалізовано адаптивний інтерфейс з використанням Nuxt.js та TypeScript. Було створено компоненти для авторизації, проходження тестів, перегляду статистики та навігації по навчальних модулях. Забезпечено зручну візуалізацію прогресу та миттєвий зворотний зв’язок під час тестування.

Розробка охопила весь цикл роботи системи — від реєстрації користувача до проходження тестів та аналізу результатів. Усі компоненти працюють узгоджено в межах єдиної архітектури, що дозволяє вважати систему повнофункціональною та готовою до впровадження або подальшого розвитку.

4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, спрямованою на виявлення помилок, перевірку відповідності системи встановленим вимогам та забезпечення якості кінцевого продукту. Основна мета тестування — підтвердити, що система працює коректно в різних умовах і сценаріях використання.

Існує кілька основних видів тестування, кожен з яких має свою специфіку та область застосування:

- **Модульне тестування (Unit Testing)** — перевірка окремих компонентів або модулів програми в ізоляції від інших частин системи. Це дозволяє виявити помилки на рівні окремих функцій або методів, забезпечуючи високу точність локалізації проблем. Unit-тести зазвичай пишуться розробниками і автоматизуються для регулярного виконання[9].
- **Інтеграційне тестування** — перевірка взаємодії між різними модулями або компонентами системи. Його мета — виявити помилки у інтерфейсах та взаємодії між компонентами, які можуть не проявлятися під час окремого тестування модулів[5].
- **Системне тестування (System Testing)** — перевірка повністю інтегрованої системи на відповідність функціональним і нефункціональним вимогам. Воно включає тестування продуктивності, безпеки, сумісності та інших аспектів[5].
- **Тестування прийнятності користувачем (User Acceptance Testing, UAT)** — заключний етап тестування, який проводиться кінцевими

[9] Myers, G. J., Sandler, C., & Badgett, T. The Art of Software Testing. Hoboken: John Wiley & Sons, 2011. 256 p.

користувачами або їх представниками для підтвердження того, що система відповідає бізнес-вимогам та готова до впровадження[5].

Веб-застосунки, орієнтовані на взаємодію з користувачем, мають специфічні особливості, які впливають на стратегію тестування:

- **Переважання UI-логіки**: Основна частина функціональності сконцентрована в інтерфейсі користувача, де логіка тісно пов'язана з візуальними компонентами.
- **Інтерактивність**: Більшість функцій реалізується через користувацькі дії (кліки, введення тексту, навігація), що ускладнює автоматизоване тестування.
- **Стан додатку**: Поведінка системи залежить від поточного стану користувацької сесії, що створює складні сценарії для покриття тестами.
- **Браузерна специфіка**: Необхідність перевірки коректної роботи в різних браузерах та на різних пристроях.

У контексті розробки системи навчання правил дорожнього руху було прийнято рішення про застосування прагматичного підходу до тестування, який відповідає специфіці та цілям проєкту.

Основні аргументи проти глибокого unit-тестування:

- **Природа застосунку**: Система є переважно інтерактивним веб-інтерфейсом, де основна логіка полягає у відображенні контенту, обробці користувацьких дій та навігації між сторінками. Такий код важко тестувати в ізоляції без втрати контексту реальної взаємодії.
- **Тісна інтеграція компонентів**: Більшість функціональності залежить від взаємодії між UI-компонентами, станом застосунку та базою даних. Розділення цих компонентів для модульного тестування може призвести до тестування штучних сценаріїв, які не відображають реальне використання.

- **Швидкість розробки**: Для навчального проєкту важливіше продемонструвати основну функціональність та архітектурні рішення, ніж досягти повного покриття тестами.
- **Специфіка бізнес-логіки**: Основна логіка системи полягає у відображенні питань, збереженні відповідей користувачів та обчисленні результатів тестів. Ця логіка краще перевіряється через інтеграційне тестування реальних сценаріїв використання.

Замість формального модульного тестування в проєкті було застосовано мануальне функціональне тестування основних сценаріїв використання:

Тестування аутентифікації:

- Реєстрація нових користувачів з валідними та невалідними даними
- Авторизація існуючих користувачів
- Перевірка захисту приватних сторінок від неавторизованого доступу

Тестування основної функціональності:

- Перегляд доступних тем та модулів
- Проходження тестів різних типів та складності
- Збереження результатів та відображення статистики
- Підрахунок очок досвіду та відстеження прогресу

Тестування інтерфейсу користувача:

- Перевірка адаптивності дизайну на різних розмірах екрану
- Тестування навігації та взаємодії з елементами інтерфейсу
- Перевірка коректності відображення контенту

Тестування продуктивності:

- Перевірка швидкості завантаження сторінок
- Перевірка стабільності при тривалому використанні

4.2 Вимоги до апаратного та програмного забезпечення

Визначення вимог до апаратного та програмного забезпечення є важливим етапом впровадження системи, оскільки дозволяє забезпечити стабільну роботу застосунку та оптимальний користувацький досвід. Вимоги розділяються на дві категорії: вимоги до серверної частини (з точки зору власника системи) та вимоги до клієнтської частини (з точки зору кінцевого користувача).

Вимоги до серверного обладнання та програмного забезпечення

Серверна частина системи навчання правил дорожнього руху має специфічні вимоги, які забезпечують стабільну роботу веб-застосунку та обслуговування одночасних запитів від багатьох користувачів.

Мінімальні апаратні вимоги до сервера:

- Процесор: будь-який x86_64 або ARMv8 сумісний процесор з тактовою частотою не менше 1.5 ГГц
- Оперативна пам'ять: мінімум 2 ГБ (рекомендовано 4 ГБ для стабільної роботи під навантаженням)
- Дисковий простір: мінімум 20 ГБ вільного місця для операційної системи, застосунку та бази даних
- Мережевий інтерфейс: підключення до мережі Інтернет з можливістю відкриття портів для HTTP/HTTPS трафіку
- Пропускна здатність мережі: рекомендовано не менше 100 Мбіт/с для забезпечення швидкого доступу користувачів

Рекомендовані апаратні характеристики:

- Процесор: багатоядерний процесор з тактовою частотою 2.0 ГГц або вище
- Оперативна пам'ять: 8 ГБ або більше для оптимальної продуктивності при високому навантаженні
- Дисковий простір: SSD накопичувач об'ємом 50 ГБ або більше для покращення швидкості доступу до даних

Програмні вимоги до сервера:

- Операційна система: будь-яка сучасна серверна ОС (Linux Ubuntu 20.04+, CentOS 8+, Windows Server 2019+)
- Середовище виконання Node.js версії 22.0.0 або новіше для забезпечення сумісності з сучасними JavaScript функціями
- Менеджер пакетів pnpm версії 10.0.0 або новіше для ефективного управління залежностями проєкту
- Система управління базами даних PostgreSQL версії 14.0 або новіше з налаштованим доступом та необхідними правами користувача
- Веб-сервер або reverse proxy (рекомендовано Nginx) для проксування запитів та забезпечення безпеки

Додаткові програмні вимоги:

- SSL-сертифікат для забезпечення HTTPS з'єднання
- Система моніторингу для відстеження стану сервера та продуктивності застосунку

Вимоги до клієнтського обладнання та програмного забезпечення

Клієнтська частина системи розроблена як адаптивний веб-застосунок, що дозволяє забезпечити широку сумісність з різними пристроями та операційними системами.

Мінімальні апаратні вимоги для настільних комп'ютерів:

- Процесор: Intel Pentium 4 з підтримкою SSE3 або еквівалентний AMD процесор[10]
- Оперативна пам'ять: мінімум 2 ГБ (рекомендовано 4 ГБ для комфортної роботи)
- Відеокарта: будь-яка з підтримкою апаратного прискорення графіки
- Дисковий простір: 100 МБ для кешування даних браузера

[10] Chrome browser system requirements [Електронний ресурс]. – Режим доступу: https://support.google.com/chrome/a/answer/7100626

- Роздільна здатність екрану: мінімум 1024x768 пікселів

Мінімальні апаратні вимоги для мобільних пристроїв:

- Процесор: ARM Cortex-A53 або еквівалентний
- Оперативна пам'ять: мінімум 2 ГБ
- Роздільна здатність екрану: мінімум 320x568 пікселів
- Сенсорний екран з підтримкою мульти-дотику

Програмні вимоги для користувачів:

- Операційна система: будь-яка сучасна ОС з підтримкою веб-браузерів (Windows 10+, macOS 11+, Linux з GUI, Android 8.0+, iOS 13+)
- Веб-браузер з підтримкою сучасних веб-стандартів:
    - Google Chrome версії 90 або новіше
    - Mozilla Firefox версії 88 або новіше
    - Apple Safari версії 14 або новіше
    - Microsoft Edge версії 90 або новіше
    - Opera версії 76 або новіше

Обов'язкові технологічні вимоги до браузера:

- Підтримка HTML5 та CSS3 стандартів
- JavaScript ECMAScript 2018 (ES9) або новіше
- Підтримка Local Storage для збереження налаштувань користувача
- Підтримка Fetch API для асинхронних HTTP запитів

4.3 Склад інсталяційного пакету

Інсталяційний пакет системи навчання правил дорожнього руху складається з комплексу програмних компонентів, які забезпечують повноцінне функціонування веб-застосунку. Архітектура розгортання системи передбачає використання контейнеризованого підходу, що дозволяє забезпечити стабільну роботу в різних середовищах та спростити процес встановлення й налаштування.

На рисунку 4.1 представлена діаграма розгортання системи, яка демонструє організацію програмних компонентів та їх взаємодію в продуктивному середовищі.

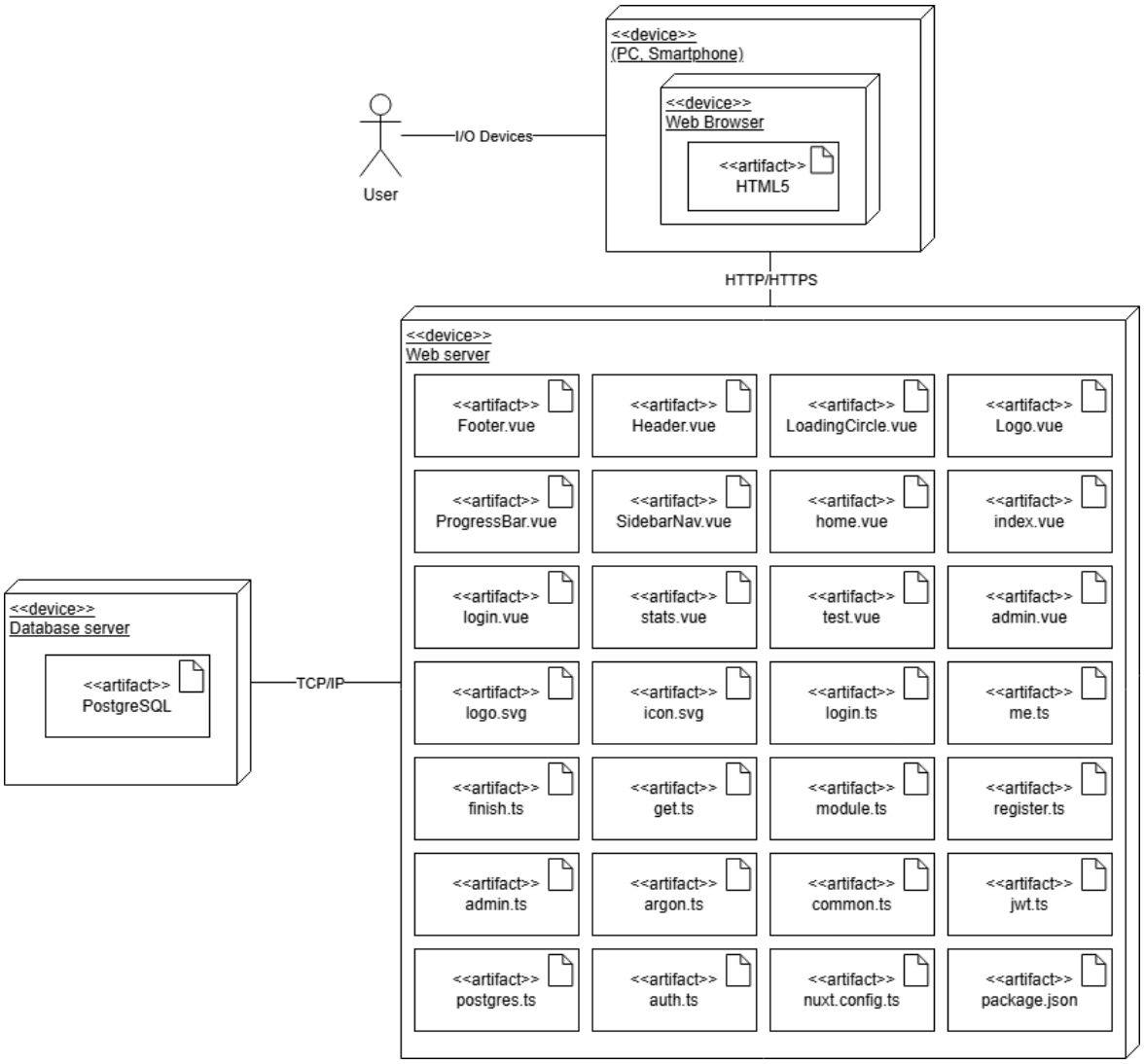

Рис 4.1. Діаграма розгортання

Клієнтська частина

Клієнтська частина системи представлена веб-браузером користувача, який взаємодіє з серверною частиною через протокол HTTP/HTTPS. Основні компоненти клієнтської частини включають:

- **Web Browser** - сучасний веб-браузер з підтримкою HTML5, CSS3 та JavaScript ES2018+

- **HTML5 Interface** - адаптивний користувацький інтерфейс, що забезпечує коректне відображення на різних типах пристроїв

Серверна частина (Web Server)

Серверна частина системи організована у вигляді набору взаємопов'язаних компонентів, які розгортаються в контейнерному середовищі. Основні компоненти включають:

Клієнтські компоненти (.vue файли):

- **Footer.vue** - компонент підвалу сторінки з загальною інформацією та навігацією
- **Header.vue** - компонент заголовка з основною навігацією та інформацією про користувача
- **LoadingCircle.vue** - компонент індикатора завантаження для покращення користувацького досвіду
- **Logo.vue** - компонент логотипу системи
- **ProgressCard.vue** - компонент відображення прогресу навчання користувача
- **Sidebar.vue** - бічна панель навігації з доступом до різних модулів системи
- **Home.vue** - головна сторінка застосунку після авторизації
- **Index.vue** - вхідна сторінка системи

Сторінки застосунку:

- **login.vue** - сторінка авторизації та реєстрації користувачів
- **stats.vue** - сторінка статистики та аналізу результатів навчання
- **test.vue** - інтерактивна сторінка проходження тестів
- **admin.vue** - адміністративна панель для управління системою

API маршрути (.ts файли):

- **login.ts** - обробка запитів авторизації та аутентифікації
- **me.ts** - отримання інформації про поточного користувача
- **get.ts** - загальні API методи для отримання даних

- **module.ts** - API для роботи з навчальними модулями
- **register.ts** - обробка реєстрації нових користувачів
- **finish.ts** - логіка завершення тестів та збереження результатів

Утилітні модулі:

- **argon.ts** - модуль для безпечного хешування паролів з використанням Argon2
- **common.ts** - загальні утиліти та допоміжні функції
- **postgres.ts** - модуль для взаємодії з базою даних PostgreSQL
- **auth.ts** - модуль автентифікації та авторизації
- **nuxt.config.ts** - конфігураційний файл Nuxt.js фреймворку
- **package.json** - опис залежностей та метаданих проєкту

База даних

Database Server (Сервер баз даних)

- **PostgreSQL** - основна система управління базами даних, що зберігає всю інформацію про користувачів, тести, питання та результати навчання

Структура інсталяційного пакету

Інсталяційний пакет системи містить наступні основні групи файлів:

1. Основні конфігураційні файли:

- **package.json** - опис залежностей Node.js та скрипти для збірки
- **nuxt.config.ts** - конфігурація Nuxt.js фреймворку
- **tsconfig.json** - налаштування TypeScript компілятора

2. Вихідний код застосунку:

- Директорія **pages/** - сторінки веб-застосунку
- Директорія **components/** - компоненти Vue.js
- Директорія **server**/ - серверні API маршрути
- Директорія **utils**/ - утилітні модулі та допоміжні функції
- Директорія **public**/ - статичні ресурси (зображення, стилі)
- Директорія **stores**/ – сховища даних (для авторизації)

- Файл **app.vue** – основна сторінка яка виконує routing до необхідної сторінки.

3. База даних:

- SQL скрипти для створення структури бази даних
- Початкові дані для заповнення системи (питання, теми тестів)

Процес розгортання

На сервері з уже встановленими Node.js та PostgreSQL розгортається вихідний код. Необхідно створити конфігураційний файл .env та заповнити поля для підключення до бази даних, а також вказати секретний ключ для генерації JWT ключів. Після конфігурації, сервер запускається командою *pnpm build && pnpm preview*.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було досягнуто поставленої мети — створено гейміфіковану систему навчання та контролю знань з правил дорожнього руху у вигляді веб-застосунку. Проведений аналіз показав недостатню ефективність традиційних засобів навчання ПДР, що стало передумовою для проєктування нового підходу з урахуванням принципів гейміфікації та адаптивності.

Розроблена система реалізує функціональність реєстрації, проходження адаптивних тестів, відстеження навчального прогресу, зберігання статистики та підтримки мотивації користувачів. В основі реалізації лежить сучасний стек технологій: Nuxt.js, TypeScript, PostgreSQL, що забезпечує високу продуктивність, безпеку та гнучкість. Архітектура системи була продумана з урахуванням масштабованості та зручності використання як з комп’ютера, так і з мобільних пристроїв.

Практична цінність роботи полягає у створенні прототипу, який може бути використаний як основа для подальшого розвитку та впровадження в реальну освітню практику, з метою підвищення рівня знань майбутніх водіїв і, як наслідок, зменшення кількості ДТП.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Duolingo – Найкращий спосіб вивчати мову [Електронний ресурс]. - https://uk.duolingo.com/
2. Green Way. Правила дорожнього руху України (ПДР 2025), вчити онлайн [Електронний ресурс]. - https://green-way.com.ua/uk/
3. Vodiy.ua. Тести з ПДР комплект Нові 2025. [Електронний ресурс]. - https://vodiy.ua/
4. Unhelkar B. Software Engineering with UML. Boca Raton : Auerbach Publications, 2020. 426 p.
5. Visual Paradigm. What is Package Diagram? [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/
6. PostgreSQL: Documentation [Електронний ресурс]. Режим доступу - https://www.postgresql.org/docs/
7. Introduction – Get Started with Nuxt [Електронний ресурс]. – Режим доступу: https://nuxt.com/docs/getting-started/introduction
8. The password hash Argon2, winner of PHC [Електронний ресурс]. – Режим доступу: https://github.com/P-H-C/phc-winner-argon2
9. Myers, G. J., Sandler, C., & Badgett, T. The Art of Software Testing. Hoboken: John Wiley & Sons, 2011. 256 p.
10. Chrome browser system requirements [Електронний ресурс]. – Режим доступу: https://support.google.com/chrome/a/answer/7100626

ДОДАТКИ

Додаток А

Код створеної бази даних мовою SQL

```sql
DROP TABLE IF EXISTS user_answers;
DROP TABLE IF EXISTS user_tests;
DROP TABLE IF EXISTS test_questions;
DROP TABLE IF EXISTS question;
DROP TABLE IF EXISTS test;
DROP TABLE IF EXISTS topic;
DROP TABLE IF EXISTS user_stats;
DROP TABLE IF EXISTS users;

CREATE TABLE IF NOT EXISTS users
(
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    is_admin BOOLEAN DEFAULT FALSE
);

CREATE TABLE IF NOT EXISTS user_stats
(
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    xp INT DEFAULT 0,
    daystreak INT DEFAULT 0,
    last_completion_date TIMESTAMP DEFAULT '1970-01-01 00:00:00',
    PRIMARY KEY (user_id)
);

CREATE TABLE IF NOT EXISTS topic
(
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    description TEXT,
    order_index INT NOT NULL DEFAULT 0
);

CREATE TABLE IF NOT EXISTS test
(
    id SERIAL PRIMARY KEY,
    topic_id INT REFERENCES topic(id) ON DELETE CASCADE,
    order_index INT NOT NULL DEFAULT 0,
    is_exam BOOLEAN DEFAULT FALSE
);
```

```
CREATE TABLE IF NOT EXISTS question
(
    id SERIAL PRIMARY KEY,
    question_title TEXT NOT NULL,
    image_url TEXT,
    answers_json TEXT NOT NULL,
    explanation TEXT
);

CREATE TABLE IF NOT EXISTS test_questions
(
    test_id INT REFERENCES test(id) ON DELETE CASCADE,
    question_id INT REFERENCES question(id) ON DELETE CASCADE,
    PRIMARY KEY (test_id, question_id)
);

CREATE TABLE IF NOT EXISTS user_tests
(
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    test_id INT REFERENCES test(id) ON DELETE CASCADE,
    is_completed BOOLEAN DEFAULT FALSE,
    completed_time TIMESTAMP DEFAULT '1970-01-01 00:00:00',
    PRIMARY KEY (user_id, test_id)
);

CREATE TABLE IF NOT EXISTS user_answers
(
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    question_id INT REFERENCES question(id) ON DELETE CASCADE,
    is_correct BOOLEAN DEFAULT FALSE,
    answer_time TIMESTAMP DEFAULT '1970-01-01 00:00:00',
    PRIMARY KEY (user_id, question_id)
);
```