Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Проєктування та розробка програмної системи автоматизованого тестування з модулем візуальної аналітики результатів на базі JavaFX

Виконав: студент 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Буняк В.С.
(підпис) (прізвище та ініціали)

Керівник _____ Мудрик І.Я.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Буняку Віталію Сергійовичу

1. Тема роботи Проєктування та розоробка програмної системи автоматизованого тестування з модулем візуальної аналітики результатів на базі JavaFX

Керівник роботи Мудрик Іван Ярославович, док. Філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від « 06 » квітня 2026 року № _____
2. Термін подання студентом роботи
3. Вихідні дані до роботи Предметна область, завдання, вимоги та специфікація, програмне рішення
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз вимог до придметної системи
2. Проєктування та розробка прогрмної системи
3. Тестування впровадження та підтримка
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги. 5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку..
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С.Ю		
Нормоконтроль	Стоянов Ю.М.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та затвердження теми		
2.	Аналіз предметної області та технічного завдання		
3.	Вибір методології розробки		
4.	Формування вимог до системи		
5.	Визначення архітектури, проєктування голосвних класів		
6.	Конструювання основного функціоналу ПЗ		
7.	Тестування та верифікація вимог		
8.	Безпека життєдіяльності, основи охорони праці		
9.	Підготовка презентації та доповіді		
10.	Нормоконтроль		
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи		

Студент _____ Буняк В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І.Я.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2026. Сторінок 70, таблиць 2 ,рисунків 20 , презентація 13

Тема: Проєктування та розробка програмної системи автоматизованого тестування з модулем візуальної аналітики результатів на базі JavaFX

Дана робота присвячена проєктуванню та розробці відкритої програмної системи автоматизованого тестування «PassIt» із модулем візуальної аналітики на базі платформи JavaFX. Об’єктом дослідження є процеси автоматизації контролю знань та візуалізації аналітичних даних. Метою роботи є створення надійного інструментарію для проведення тестування, який забезпечує інтелектуальну обробку відповідей та надання засобів статистичного аналізу результатів у реальному часі.

У ході виконання роботи було спроєктовано та реалізовано багаторівневу архітектуру додатка на основі патерна MVC. Особливу увагу приділено оптимізації взаємодії з базою даних MySQL шляхом впровадження HikariCP, що суттєво підвищило продуктивність системи при інтенсивних запитах. Практична цінність розробки полягає у створенні аналітичного модуля для супервайзера, який використовує методи лінійної регресії для прогнозування активності користувачів та надає наочні інструменти візуалізації для оцінки успішності та популярності категорій тестів. Реалізований алгоритм нормалізації текстових даних забезпечує коректну автоматичну перевірку відкритих запитань, мінімізуючи вплив помилок регістру та зайвих пробілів.

Ключові слова: JavaFX, MySQL, HikariCP, автоматизоване тестування, візуальна аналітика, пул з'єднань, лінійна регресія, MVC.

ABSTRACT

Bachelor's certification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 “Software Engineering”. TNTU, 2026. Pages 70, tables 2 , figures 20, presentation 13

Theme: Design and Development of an Automated Testing Software System with a Visual Results Analytics Module Based on JavaFX

This work is devoted to the design and development of the "PassIt" automated testing software system with a visual analytics module based on the JavaFX platform. The object of the study is the processes of automating knowledge control and visualization of analytical data in educational environments. The aim of the work is to create a reliable toolkit for conducting testing that provides intelligent processing of responses and provides the administrator with tools for real-time statistical analysis of results.

During the research, a multi-tier application architecture based on the MVC pattern was designed and implemented, allowing for the separation of data processing logic from the user interface. Particular attention is paid to optimizing interaction with the MySQL database through the implementation of the HikariCP connection pool, which significantly increased system performance during intensive queries. The practical value of the development lies in the creation of an analytical module for the supervisor, which utilizes linear regression methods to forecast user activity and provides intuitive visualization tools to assess performance and the popularity of test categories. The implemented text data normalization algorithm ensures correct automated evaluation of open-ended questions, minimizing the impact of case sensitivity and redundant spaces.

Keywords: JavaFX, MySQL, HikariCP, automated testing, visual analytics, connection pool, linear regression, MVC.

ЗМІСТ

АНОТАЦІЯ 4
ABSTRACT 5
ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 9
1.1 Аналіз предметної області та концепція відкритої платформи 9
1.2 Аналіз існуючих рішень та обґрунтування розробки 9
1.3 Специфікація функціональних вимог 10
1.4 Аналіз нефункціональних вимог та технічних обмежень 12
1.5 Моделювання діаграми варіантів використання 12
1.6 Висновки до розділу 1 16
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ 18
2.1 Архітектурна побудова системи на базі патерна MVC 18
2.2 Проєктування логічної та фізичної структури бази даних 19
2.3 Програмна реалізація, структура об'єктної моделі та контроль версій 20
2.4 Об'єктно-орієнтована модель та діаграма класів 22
2.5 Алгоритмічне забезпечення модуля прогнозування на основі лінійної регресії 24
2.6 Розробка інтерфейсу користувача та опис функціональних можливостей системи 27
2.7 Опис вкладки «Статистика» 43
2.8 Висновки до розділу 2 46
РОЗДІЛ 3. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА 47
3.1 Методика та результати тестування програмних модулів 47
3.2 Впровадження системи та інструкція користувача 48
3.3 Системні вимоги та регламент підтримки 49
3.4 Висновки до розділу 3 51
3.5 Навантажувальне тестування 51
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 53
4.1 Надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природними причинами 53
4.2 Навчання працюючих та інструктажі з охорони праці 56
ВИСНОВКИ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62
ДОДАТКИ 65

ДОДАТОК А. Повна діаграма класів згенерована Intellij Idea Ultimate
ДОДАТОК Б. Тези для публікації на науково-технічну конференцію
ДОДАТОК В. Лістинг коду для реалізації алгоритму відстані Лівенштейна
ДОДАТОК Г. Лістинг коду для будування графіків
ДОДАТОК Д. Диск із кваліфікаційною роботою бакалавра

ВСТУП

У сучасну епоху цифрової трансформації та активного обміну інформацією зростає потреба у доступних і водночас потужних інструментах самоперевірки та незалежної оцінки знань. На відміну від закритих академічних платформ, існує гострий дефіцит відкритих систем, де будь-який користувач може не лише проходити тестування, а й виступати автором власного контенту.

Актуальність роботи полягає у створенні програмного забезпечення, яке поєднує низький поріг входження для широкої аудиторії з інструментарієм візуальної аналітики. Більшість існуючих публічних сервісів мають ігрову спрямованість або обмежений функціонал аналізу, що робить розробку системи «PassIt» із вбудованими модулями математичного прогнозування та динамічної звітності досить затребуваною.

Об’єктом дослідження є процеси автоматизації контролю знань, збору та візуалізації аналітичних даних у відкритих інформаційних середовищах.

Предметом дослідження є архітектурні рішення, методи лінійної регресії для аналізу активності та програмні засоби розробки кросплатформних додатків на базі технологій JavaFX та реляційних баз даних.

Мета роботи — проєктування та програмна реалізація системи «PassIt», що забезпечує відкритий доступ до створення та проходження тестів, реалізацію рольової моделі з обмеженням контенту та надання супервайзерам засобів інтелектуальної візуалізації результатів у реальному часі.

Практичне значення отриманих результатів полягає у створенні готового до використання програмного продукту, який дозволяє користувачам легко ділитися знаннями, а супервайзерам отримувати глибокий інженерний аналіз успішності респондентів та динаміки попиту на створений матеріал.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У даному розділі проведено дослідження предметної області та обґрунтовано концепцію відкритої платформи тестування «PassIt». Здійснено порівняльний аналіз існуючих рішень на основі якого сформовано функціональні та нефункціональні вимоги до системи. Розглянуто рольову модель користувачів, а також змодельовано діаграму варіантів використання, яка відображає основні сценарії взаємодії акторів із розробленим програмним продуктом

1.1 Аналіз предметної області та концепція відкритої платформи

Сучасні тенденції споживання контенту свідчать про зростання попиту на інструменти самоперевірки. На відміну від жорстких корпоративних чи академічних систем, відкрита платформа тестування має забезпечувати низький поріг входження, де будь-який зареєстрований користувач може виступати як у ролі пересічного користувача, так і в ролі автора.

Основна проблема існуючих відкритих рішень полягає у відсутності балансу або вони занадто примітивні для професійної оцінки результатів, або занадто складні для звичайного користувача. Система «PassIt» розробляється як гнучке середовище, де публічність тестів поєднується з глибинною аналітикою, доступною авторам для оцінки якості їхнього контенту.

1.2 Аналіз існуючих рішень та обґрунтування розробки

Для вибору оптимального шляху реалізації було проведено порівняльний аналіз популярних систем:

- Moodle: Потужна система управління навчанням. Мінусом є складність налаштування для локальних завдань та високі вимоги до серверних ресурсів.

- Quizizz/Kahoot/Quizzlet: Орієнтовані на ігрову механіку. Мають обмежені можливості для десктопного аналізу великих масивів даних.
- Google Forms: Легкий інструмент для збору відповідей. Основним недоліком є обмеженість у візуальній аналітиці в реальному часі та неможливість повноцінної роботи у закритих локальних мережах без доступу до інтернету.

Таблиця 1.1 Порівняльна характеристика систем тестування

Критерій	Moodle	Google Forms	Kahoot	PassIt
Тип платформи	Веб-сервіс	Хмарний сервіс	Веб-сервіс	Десктоп
Вбудована аналітика	Середня (звіти)	Базова (діаграми)	Низька	Середня (тренд, регресія)
Автономність	Потребує сервер	Потребує інтернет	Потребує інтернет	Потребує інтернет
Складність розгортання	Висока	Низька	Низька	Низька

1.3 Специфікація функціональних вимог

Функціональність системи базується на забезпеченні вільного доступу до створення та проходження тестів, що вимагає чіткого розмежування прав та можливостей користувачів. В основі лежить модель саморегульованої спільноти, де якість контенту контролюється через рольові обмеження. Побудована архітектура платформи PassIt спрямована на створення демократичного цифрового простору, де кожен користувач отримує інструментарій не лише для використання готових знань, а й для генерації власних тестів для проходження іншими

користувача.

Рольова модель, що включає звичайних користувачів та супервайзерів, забезпечує баланс між свободою творчості та необхідністю системного нагляду. У той час як користувачі фокусуються на персональному розвитку, проходженні тестів та створенні авторського контенту в конструкторі, супервайзери здійснюють високорівневий моніторинг стабільності платформи та актуальності її тематичного наповнення.

Звичайний користувач отримує базовий набір інструментів, що включає:

- Реєстрацію персонального профілю та керування ним.
- Пошук тестів за назвою або категорією.
- Проходження тестів із автоматичним розрахунком результатів у реальному часі.
- Перегляд особистої аналітики та статистики профілю.
- Створення обмеженої кількості власних тестів до 4 одиниць. Дане обмеження впроваджено для стимулювання авторів до створення максимально релевантних та якісних опитувань.
- Можливість інтеграції власної ШІ моделі для аналізу тестів.

Супервайзер є особливою роллю, яка відкриває доступ до додаткових функцій у системы:

- Всі ті самі можливості що і у звичайного користувача у системі.
- Зняття кількісного ліміту на створення тестів.
- Доступ до закритої панелі аналітики, де збираються дані про кожного респондента, що проходив тест автора.
- Додавання нових категорій.
- Видалення тестів та користувачів з системи.
- Можливість перегляду математичних прогнозів щодо популярності контенту.

1.4 Аналіз нефункціональних вимог та технічних обмежень

Окрім прямих функцій користувачів системи, розроблювана програмна система має відповідати основним технічним стандартам, які забезпечують комфортну роботу користувача та стабільність даних у системі.

Вимоги до продуктивності та архітектури:

- Ефективність обробки запитів: Оскільки система передбачає роботу багатьох користувачів із єдиною базою даних де може бути багато запитів одночасно, використання пулу з’єднань через бібліотеку HikariCP є критичною вимогою.
- Кросплатформенність: Використання середовища Java забезпечує запуск додатка на операційних системах Windows, macOS та Linux без зміни вихідного коду.
- Автономність та мережева стійкість: Програма повинна коректно обробляти ситуації з короткочасним розривом інтернет-з'єднання, зберігаючи стан проходження тесту в локальній пам'яті до відновлення зв'язку з MySQL для продовження роботи, такий підхід дозволяє зберігати дані користувача без необхідності повторного проходження тестування.

Вимоги до безпеки:

- Цілісність даних: Користувач не повинен мати можливості змінити результат тесту після його завершення або втрутитися в логіку підрахунку балів на стороні клієнта.
- Захист доступу: Доступ до панелі аналітики та статистики Супервайзера має бути захищений перевіркою прав на рівні сесії користувача.

1.5 Моделювання діаграми варіантів використання

Для візуалізації взаємодії акторів із системою розроблено діаграму варіантів використання (див. рис. 1.1). Вона відображає основні сценарії, які реалізовані в «PassIt».

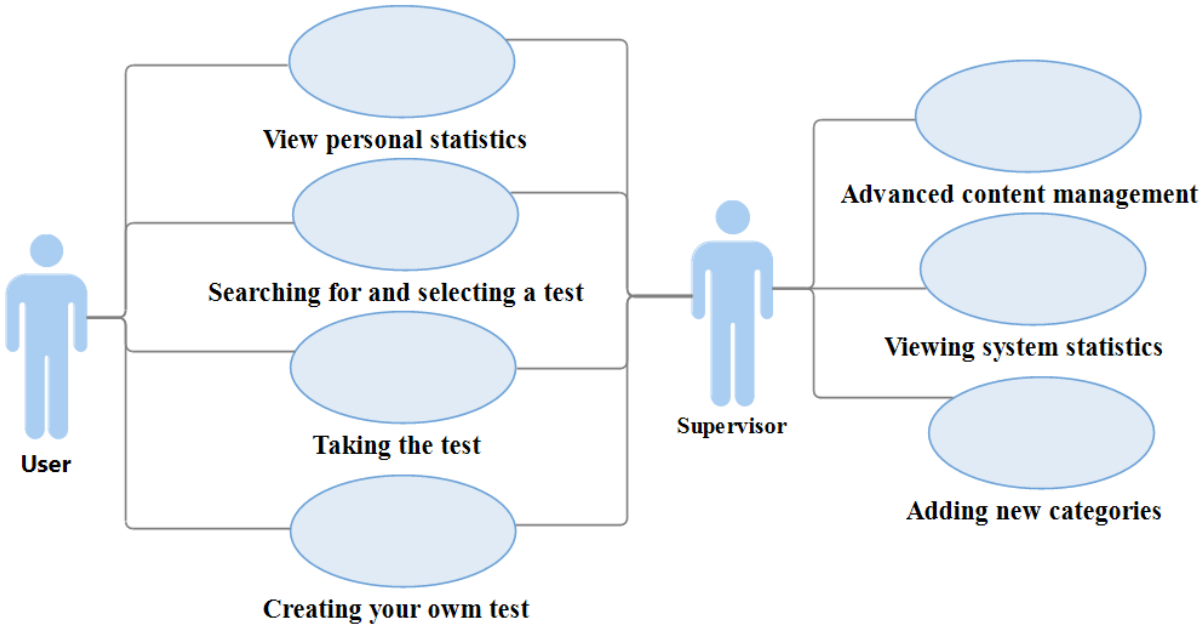

Рисунок 1.1 Діаграма варіантів використання

Нижче наведено розширений та деталізований опис варіантів взаємодій в межах платформи для ключового актора системи — «Користувач». Цей опис розкриває не лише технічні кроки, а й логіку функціонування відкритої системи, де користувач виступає активним учасником розробленого програмного продукту.

1. Варіант використання: «Пошук та вибір тесту»
   - Актор: Користувач.
   - Передумова: Користувач авторизований у системі.

   Основний потік подій:

   1) Користувач переходить у розділ «Пошук».
   2) Користувач вводить ключове слово в текстове поле або вибирає категорію зі списку.
   3) Система формує SQL-запит до таблиці Tests із фільтрацією за

заданими параметрами.

4) Система відображає список доступних тестів у вікні додатка.
5) Користувач вибирає конкретний необхідний тест для перегляду та проходження.

У результаті користувач системи отримує доступ до стартового екрана вибраного тесту.

2. Варіант використання: «Проходження тестування»

- Актор: Користувач.
- Передумова: Вибрано конкретний тест із каталогу.

Основний потік подій:

1) Система завантажує масив питань, пов'язаних з ідентифікатором тесту test_id.
2) Користувач обирає варіанти відповіді та натискає «Далі» для переходу до наступного питання.
3) Після останнього питання користувач натискає «Завершити тест» щоб перейти до результатів.
4) Система порівнює обрані id відповідей із правильними в базі даних та зберігає пройдений тест в історії.

Якщо користувач повністю закриває програму до завершення тестування, поточний прогрес не зберігається, проте якщо під час проходження виникають складнощі із з’єднанням до бази даних то прогрес зберігається до відновлення зєднання з БД.

У результатів користувач додає запис нового тесту в таблицю test_attempts та отримує відображення результату.

3. Варіант використання: «Створення власного тесту»

- Актор: Користувач.
- Передумова: Наявність менше ніж 4-х створених тестів у профілі.

Основний потік подій:

1) Користувач натискає кнопку «Створити новий тест».
2) Система перевіряє кількість записів в таблиці Tests у базі даних.

3) При успішній валідації відкривається форма конструктора для створення тесту.
4) Користувач додає питання та варіанти відповідей, позначаючи правильні.
5) Користувач натискає «Опублікувати».

У результаті новий тест додається в базу даних та появляється у списку тестів платформи.

4. Варіант використання: «Перегляд особистої статистики»

- Актор: Користувач.
- Передумова: Наявність історії проходжень у базі даних.

Основний потік подій:

1) Користувач переходить у вкладку «Мої Тести».
2) Користувач обирає необхідний тест.
3) У списку тестів користувач бачить назву тесту, дату та отриманий відсоток успішності.

У результаті користувач отримав результат та аналіз пройденого ним тесту

Додатково нижче наведено розширений аналіз варіантів взаємодії для привілейованого актора системи — «Супервайзер». Цей опис розкриває специфіку управління відкритою платформою.

1. Варіант використання: «Розширене керування контентом»

- Актор: Супервайзер.
- Передумова: Користувач має статус Супервайзера.

Основний потік подій:

1) Автор ініціює створення нового тесту.
2) Система ідентифікує роль «Супервайзер» і ігнорує базовий ліміт у 4 тести.
3) Автор додає необмежену кількість питань.
4) Система зберігає структуру тесту в БД MySQL.

У результаті тест буде опубліковано для проходження без системних обмежень.

2. Варіант використання: «Перегляд статистики системи»
   - Актор: Супервайзер.
   - Передумова: Користувач має статус Супервайзера.

   Основний потік подій:

   1) Супервайзер переходить в випадаючому меню у розділ статистики Supervisor -> Статистика.
   2) Система генерує та відображає супервайзеру на екрані розраховані статистичні дані.

У результаті супервайзер отримує статистичні дані у вигляді діаграм для перегляду.

3. Варіант використання: «Додавання нових категорій»
   - Актор: Супервайзер.
   - Передумова: Користувач має статус Супервайзера та категорії ще не існує.

   Основний потік подій:

   1) Супервайзер переходить у вкладку Supervisor та натискає кнопку додавання нової категорії.
   2) Супервайзер вводить назву категорії.
   3) Система перевіряє чи категорія існує.
   4) Якщо категорія не існує то супервайзер бачить сповіщення про успішне додавання.

У результаті супервайзер додає нову доступну для використання категорію в систему

1.6 Висновки до розділу 1

У першому розділі було проведено аналіз предметної області та обґрунтовано необхідність розробки автоматизованої системи тестування «PassIt». У ході дослідження було розглянуто особливості організації процесу тестування знань, а також проаналізовано існуючі програмні рішення, їхні переваги та недоліки.

Встановлено, що значна частина наявних систем має обмежені можливості щодо управління тестами та аналізу результатів, а також не завжди забезпечує достатній рівень зручності для користувачів.

На основі проведеного аналізу було визначено основні функціональні та нефункціональні вимоги до системи, що дозволило сформувати чітке бачення майбутнього програмного продукту. Особливу увагу приділено вимогам до зручності використання, безпеки даних, продуктивності та надійності системи.

Крім того, було побудовано діаграму варіантів використання, яка відображає взаємодію користувачів із системою та розмежовує функціональні можливості звичайного користувача і супервайзера.

Отримані результати дослідження стали основою для подальшого проєктування системи «PassIt», вибору архітектурних рішень та реалізації її основних функціональних можливостей у наступних розділах дипломної роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

У даному розділі описано безпосередню реалізацію програмної системи «PassIt» від архітектурних рішень до конкретних програмних модулів. Розглянуто побудову системи на базі патерна MVC, проєктування бази даних та об'єктної моделі, а також алгоритмічне забезпечення модуля прогнозування на основі лінійної регресії. Окрему увагу приділено опису реалізованого інтерфейсу користувача та вкладки статистики, яка демонструє практичну роботу аналітичного модуля системи.

2.1 Архітектурна побудова системи на базі патерна MVC

Для забезпечення високої якості програмного коду, гнучкості інтерфейсу та можливості подальшого розширення системи, було обрано архітектурний патерн Model-View-Controller (MVC). Вибір даної архітектури обумовлений необхідністю чіткого розділення бізнес-логіки додатка, графічного представлення та механізмів управління даними. В межах розробленої системи компоненти MVC реалізовані наступним чином.

Model (Модель): відповідає за структуру даних та логіку взаємодії з базою даних MySQL. Сюди відносяться класи-сутності (наприклад, User, Test, Question), а також сервісні класи, що використовують HikariCP для виконання SQL-запитів. Модель не має прямого доступу до візуальних елементів.

View (Представлення): реалізоване за допомогою технології JavaFX та мови розмітки FXML. Це дозволяє описувати інтерфейс у вигляді декларативних файлів, що спрощує його модифікацію без втручання в програмну логіку. Кожне вікно системи (авторизація, конструктор тестів, розділ аналітики) є окремим файлом представлення.

Controller (Контролер): виступає посередником між користувачем та даними. Контролери обробляють події (натискання кнопок, введення тексту), викликають

відповідні методи моделей та оновлюють візуальний стан вікон.

Така багаторівнева структура дозволяє паралельно розробляти візуальну частину та основний функціонал, а також спрощує тестування окремих модулів системи.

2.2 Проєктування логічної та фізичної структури бази даних

Для забезпечення стабільної роботи платформи та реалізації відкритої моделі тестування було розроблено реляційну базу даних на базі СУБД MySQL. Структура бази даних нормалізована до третьої нормальної форми, що мінімізує надлишковість даних та гарантує цілісність інформації. Основними компонентами бази даних є наступні таблиці:

- Accounts: Центральна таблиця користувачів. Зберігає облікові дані (логін, хеш пароля) та атрибут ролі. Саме на рівні цієї таблиці система перевіряє права доступу користувача.
- Categories: Довідник тематичних категорій. Використання окремої таблиці дозволяє легко розширювати перелік тематик та здійснювати швидку фільтрацію контенту.
- Tests: Зберігає загальну інформацію про тести. Тут реалізовано зв’язок «багато до одного» до авторів.
- Questions: Містить перелік запитань для кожного створеного тесту. Поле типу питання дозволяє системі визначати логіку відображення варіантів у JavaFX.э
- Answers: Таблиця варіантів відповідей, де кожне питання пов’язане з декількома можливими варіантами через зовнішній ключ question_id. Поле is_correct визначає правильны відповіді для автоматичної перевірки.
- TestAttempts: Важливий елемент для розробленого аналітичного модуля. Тут фіксується фактична спроба кожного проходження

користувачем тесту. Ці дані є основою для побудови графіків активності та лінійної регресії.

- UserAnswers: Логи відповідей користувача на кожне конкретне питання в межах спроби. Це дозволяє супервайзеру аналізувати, які саме питання у тесті викликають найбільші труднощі у користувачів під час проходження.
- UserNotifications & UserPreferences: Допоміжні таблиці для покращення UX. Перша відповідає за сповіщення про нові тести, а друга за персоналізацію інтерфейсу додатка по інтересуючих категоріях.

Для оптимізації швидкодії системи було впроваджено індексацію за зовнішніми ключами, що дозволяє швидко отримувати дані для побудови візуальних звітів. Використання пулу з'єднань HikariCP забезпечує ефективне управління потоками підключень до цих таблиць, запобігаючи перевантаженню бази даних при масових запитах.

2.3 Програмна реалізація, структура об'єктної моделі та контроль версій

Проєкт розроблено на базі системи автоматизації збірки Maven, що дозволяє ефективно керувати залежностями Програмний код зосереджено у пакеті vt.passit, який має чітку модульну структуру, що відповідає сучасним стандартам розробки ПЗ. Опис компонентної структури пакетів:

vt.passit.Animations: Містить класи для керування візуальними ефектами. Наприклад, клас TestCardAnimator відповідає за плавну взаємодію з картками тестів, що покращує користувацький досвід та робить інтерфейс динамічним.

vt.passit.Controllers: Пакет, у якому зосереджено логіку обробки подій. Контролери є сполучною ланкою між FXML-розміткою та бізнес-логікою, відповідаючи за ініціалізацію вікон та реакцію на дії користувача.

vt.passit.Interfaces: Містить абстракції та інтерфейси, що визначають шляхи для взаємодії між різними модулями системи. Це забезпечує гнучкість коду та

легкість його тестування.

vt.passit.Modules: Основне ядро системи. Тут реалізовано алгоритми перевірки тестів, взаємодію з базою даних через пул з'єднань та математичні методи обробки статистики для супервайзерів.

vt.passit.Views: Відповідає за програмне керування вікнами. Клас PassItMain є точкою входу в розроблений додаток, ініціалізуючи головну сцену JavaFX для користувача.

Перерахування (Enums): Для стандартизації даних у головному пакеті визначено QuestionType (типи питань) та Role (ролі користувачів: звичайний та супервайзер). Це гарантує типізацію та виключає помилки при передачі станів між модулями.

Управління вихідним кодом та відстеження етапів розробки програмного продукту здійснювалося за допомогою децентралізованої системи контролю версій Git. Проєкт розгорнуто у віддаленому хостингу репозиторіїв GitHub.

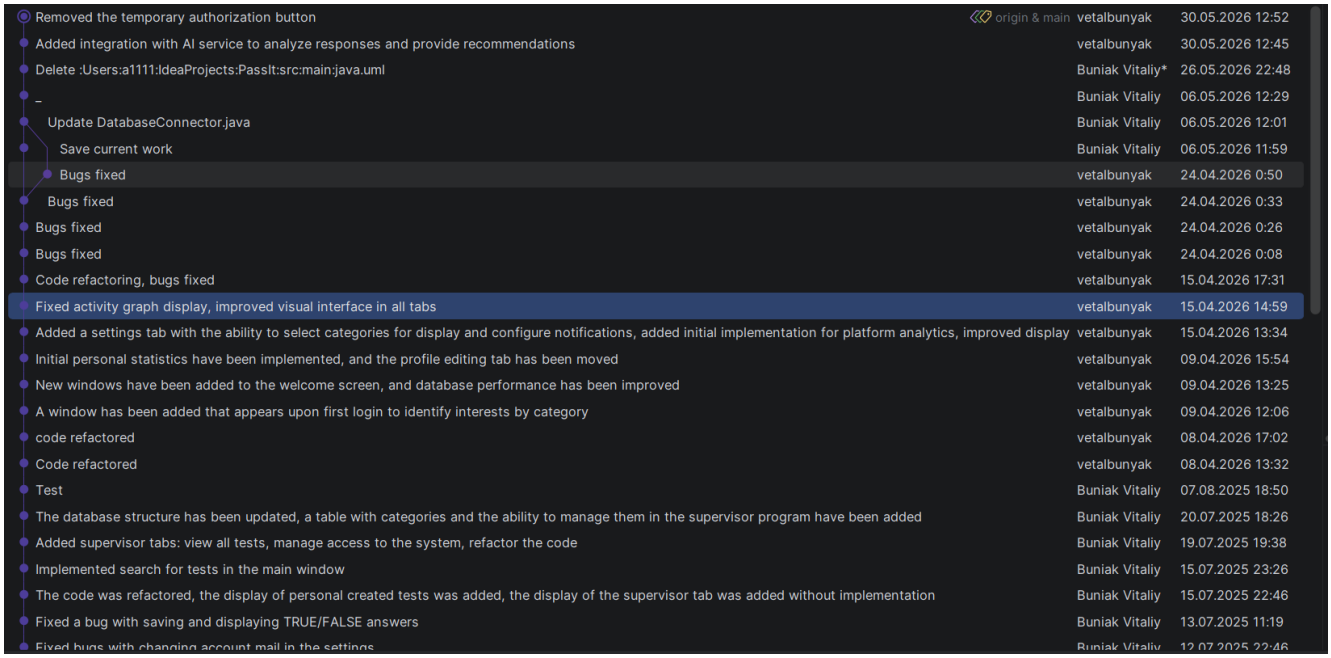

Рисунок 2.1 Дерево коммітів

Інтеграція системи Git у робочий процес забезпечила вирішення таких завдань:

1. Фіксація проміжних станів— кожна логічно завершена зміна, архітектурне оновлення чи виправлення дефектів реєструвалися в історії розробки з відповідними текстовими описами, що дозволяє за потреби виконати відкат до стабільних ревізій коду.

2. Безпека та ізоляція коду — для розмежування експериментального та стабільного функціоналу використовувалася модель розгалуження. Основна стабільна кодова база підтримується в гілці main, тоді як розробка нових модулів (наприклад, інтеграція з ШІ чи налаштування пулу з'єднань БД) виконувалася в ізольованих гілках з наступним злиттям.

3. Синхронізація робочого середовища — взаємодія з віддаленим сервером конфігурувалася за допомогою захищеного протоколу SSH та протоколу HTTPS, що гарантує безпечний обмін даними між локальною робочою станцією розробника та хмарним сховищем. Такий підхід до організації коду та використання сучасних інструментів контролю версій гарантує масштабованість системи, спрощує її подальше супроводження та унеможливлює втрату критично важливих даних під час рефакторингу.

2.4 Об'єктно-орієнтована модель та діаграма класів

Проєктування програмної системи «PassIt» базується на принципах об'єктно-орієнтованого програмування (ООП) та архітектурному шаблоні MVC (Model-View-Controller). Це дозволило розділити бізнес-логіку, інтерфейс користувача та керування даними у незалежні модулі. Статична структура системи, її ключові класи та їх взаємозв'язки відображені на діаграмі класів UML (рис. 2.1).

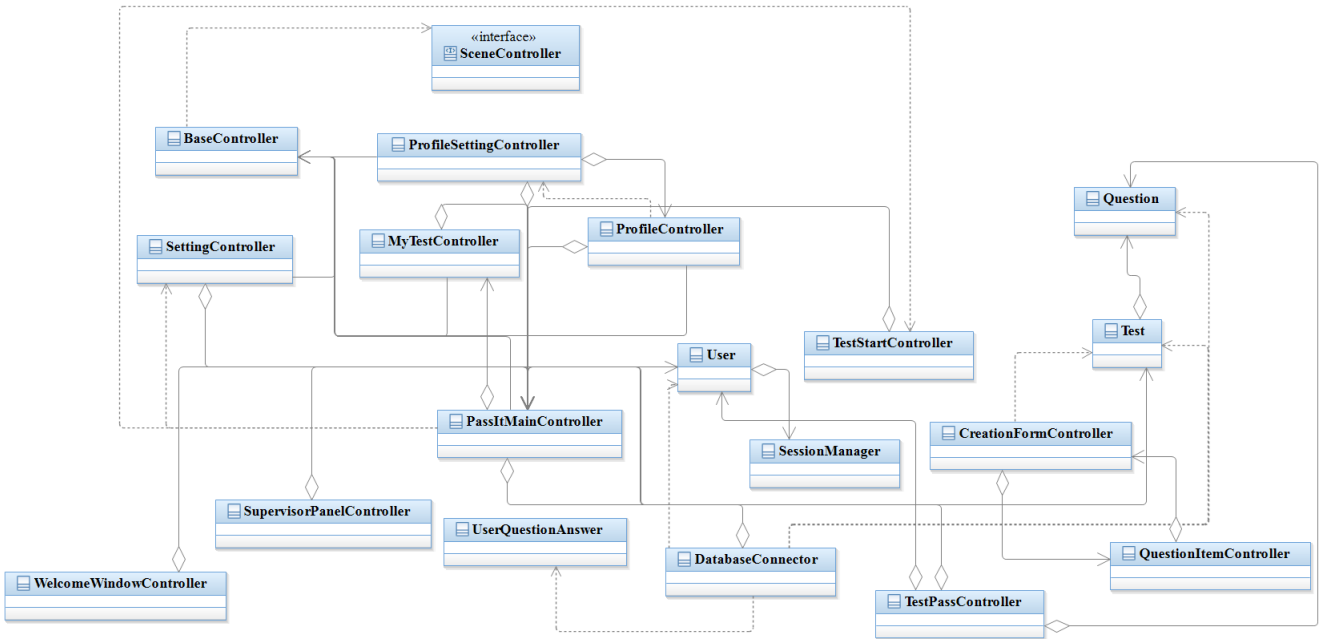

Рисунок 2.2 Діаграма класів системи

На основі розробленої моделі можна виділити чотири фундаментальні рівні архітектури.

Рівень представлення та архітектура контроллерів: Управління графічним інтерфейсом JavaFX реалізовано через ієрархію контролерів, де базовий клас BaseController виконує роль шаблону. Він містить методи для динамічної зміни сцен, обробки глобальних подій та управління модальними вікнами. Спеціалізовані контролери наслідують цей функціонал, що мінімізує дублювання коду.

Об'єктна модель предметної області: Модель даних спроєктована для підтримки високої вкладеності об'єктів. Клас Test виступає агрегатором для списку об'єктів Question, кожен з яких, у свою чергу, містить колекцію AnswerOption. Така деревоподібна структура дозволяє системі легко обробляти тести різних типів.

Рівень сервісів та інтелектуальної обробки: Для забезпечення стабільності та чистоти коду, важка обчислювальна логіка винесена в автономні сервісні модулі. Класи DatabaseConnector & SessionManager: Реалізують патерни «Singleton» та «State», забезпечуючи стабільне підключення до MySQL через пул HikariCP та зберігаючи стан авторизації користувача. AnswerEvaluator виступає інтелектуальним ядром системи. В ньому інкапсульована логіка валідації

відповідей. Це дозволяє легко розширювати систему новими типами перевірок (наприклад, додавання підтримки регулярних виразів або нових метрик схожості тексту) без зміни логіки інтерфейсу. HashData забезпечує безпеку персональних даних, реалізуючи алгоритми хешування паролів перед їх збереженням у базі даних.

Компонентно-орієнтований UI-шар: Особливістю архітектури є використання паттерна «Factory». Класи TestCell, CategoryCell та інші є самостійними мікро-контролерами. Це дозволяє динамічно генерувати складні елементи інтерфейсу залежно від даних, що надходять із БД. За візуальну динаміку відповідає TestCardAnimator, який відокремлює логіку анімацій від бізнес-логіки додатка.

Така структурована архітектура забезпечує високу стійкість системи до помилок та полегшує процес тестування і підтримки програмного продукту на етапі впровадження.

2.5 Алгоритмічне забезпечення модуля прогнозування на основі лінійної регресії

Для супервайзера розробленої системи важливо не просто бачити кількість пройдених тестів сьогодні, а розуміти, що чекає на проєкт завтра. Саме тому в програму вбудовано функцію прогнозування, яка працює на основі лінійної регресії. Суть цього методу полягає у пошуку зв’язку між часом (змінна x) та активністю людей (змінна y). Програма бере дані з бази за минулі дні та намагається провести крізь них ідеальну пряму лінію, яка найкраще описує загальний напрямок подій. Якщо ця лінія спрямована вгору, це математично доводить, що популярність тесту зростає. Нижче наведена формула моделі простої лінійної регресії, формула 2.1.

$$y = \beta_0 + \beta_1 x + \varepsilon \quad (2.1)$$

де:

y - прогнозована активність;

x - часовий показник;

β_0 - вільний член, що визначає початковий рівень активності;

β_1 - коефіцієнт нахилу, що вказує на швидкість зростання або спадання популярності;

ε - випадкова помилка

Для знаходження оптимальних параметрів β_0 та β_1 у класі SupervisorPanelController (метод updateActivityChart) реалізовано метод найменших квадратів. Метод найменших квадратів дозволяє системі знайти таку лінію тренду, яка проходить максимально близько до всіх точок фактичної активності. Програмна реалізація у контролері SupervisorPanelController автоматично обчислює коефіцієнт нахилу (динаміку популярності) та вільний член (базовий рівень), що дозволяє візуалізувати прогноз на майбутні періоди навіть при нерівномірному проходженні тестів користувачами. Розрахунок здійснюється за наступним математичним алгоритмом:

1) Накопичення сум: Програма ітерує по колекції sortedActivity, обчислюючи суми: $\sum x, \sum y, \sum x^2, \sum xy$
2) Обчислення коефіцієнта нахилу:

$$\beta_1 = \frac{n\sum(x_i y_i) - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2} \quad (2.2)$$

3) Обчислення точки перетину:

$$\beta_0 = \frac{\sum y_i - \beta_1 \sum x_i}{n} \quad (2.3)$$

У коді це реалізовано через змінні slope та intercept. Особливістю реалізації є перевірка умови n > 1 (кількість днів), що запобігає діленню на нуль при недостатній кількості статистичних даних.

Після проведення математичних розрахунків система генерує дві серії даних для компонента LineChart:

- Actual Activity: реальні дані про кількість проходжень за кожен день.
- Trend Line: лінія прогнозу, побудована за отриманими коефіцієнтами.

Лінійна регресія: Представлений лістинг демонструє математичне ядро системи. Програма перевіряє наявність даних та обчислює знаменник (denom) для уникнення помилки ділення на нуль. На основі статистичних сум розраховуються коефіцієнт нахилу тренду (slope) та зміщення прямої (intercept). Цикл по днях тижня будує прогнозний вектор . Нижче наведено програмну реалізацію методів.

Лістинг коду:

```
    // Розрахунок коефіцієнтів лінійної регресії (метод найменших
квадратів)
    if (hasData && n > 1) {
        double denom = n * sumX2 - sumX * sumX;
        if (denom != 0) {
            double slope = (n * sumXY - sumX * sumY) / denom;
            double intercept = (sumY - slope * sumX) / n;

            int i = 0;
            for (String date : filled.keySet()) {
                // Додавання точок прогнозу із захистом від від'ємних
значень

trendSeries.getData().add(new XYChart.Data<>(date, Math.max(0, slope * i +
intercept)));
                i++;
            }
        }
    }

    // Динамічне масштабування осі Y під максимальне значення активності
    double maxVal = series.getData().stream()
            .mapToDouble(d -> d.getYValue().doubleValue())
            .max().orElse(0);

    if (maxVal == 0) {
        yAxis.setUpperBound(5);
        yAxis.setTickUnit(1);
    } else {
        yAxis.setUpperBound(Math.ceil(maxVal) + 1);
        yAxis.setTickUnit(Math.max(1, Math.ceil((maxVal + 1) / 6)));
    }
```

Використання цього алгоритму дає Супервайзеру змогу виявляти позитивні тренди ($\beta_1 > 0$), що свідчить про успішність тесту та вчасно помічати спад інтересу ($\beta_1 < 0$), що вказує на необхідність оновлення запитань або зміни тематики контенту.

2.6 Розробка інтерфейсу користувача та опис функціональних можливостей системи

У даному підрозділі наведено опис практичної реалізації розробленої платформи тестування. Візуальна частина системи побудована з використанням бібліотеки JavaFX, що дозволяє створити сучасний, адаптивний та інтуїтивно зрозумілий інтерфейс. Взаємодія користувача з програмою реалізована через послідовну зміну екранних форм, кожна з яких відповідає за конкретний етап роботи: ідентифікацію, вибір контенту або безпосередньо процес тестування.

Після успішного проходження етапу авторизації, при першому вході в систему користувачеві відображається вікно персоналізації профілю (рис. 2.3)

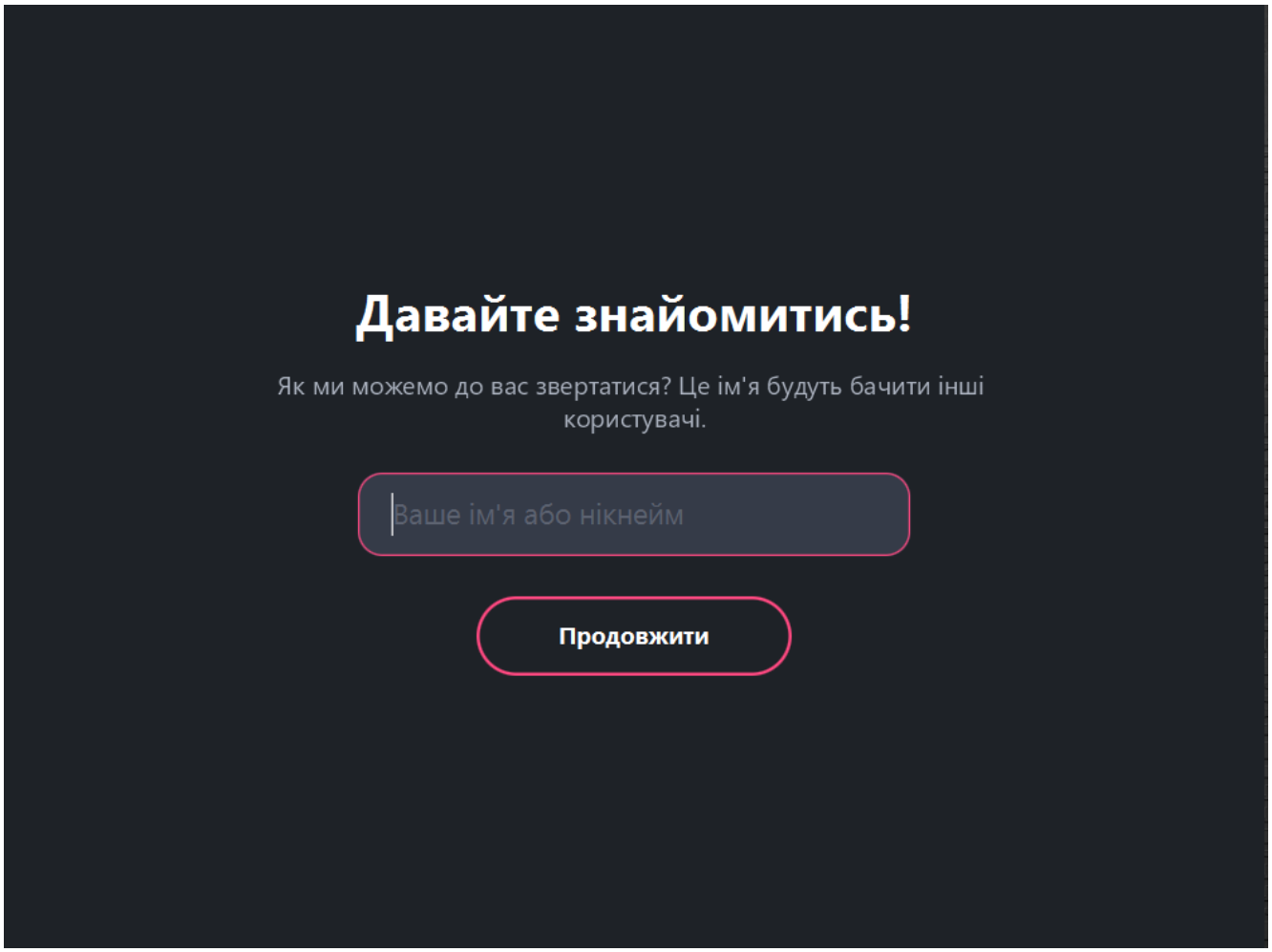

Рисунок 2.3 Вікно первинного налаштування профілю користувача

Це вікно виконує функцію ідентифікації та збору аналітичних даних для подальшої персоналізації контенту. Користувач вказує своє ім'я або нікнейм, який надалі буде використовуватись у системі. Дане вікно з’являється лише один раз за весь цикл використання акаунта. Після натискання кнопки «Продовжити» внесені дані зберігаються в базі даних і відкривається наступне вікно онбордингу. Після введення імені система пропонує користувачу визначити пріоритетні напрямки тестування. Це вікно (рис. 2.4) є ключовим для роботи алгоритму рекомендацій.

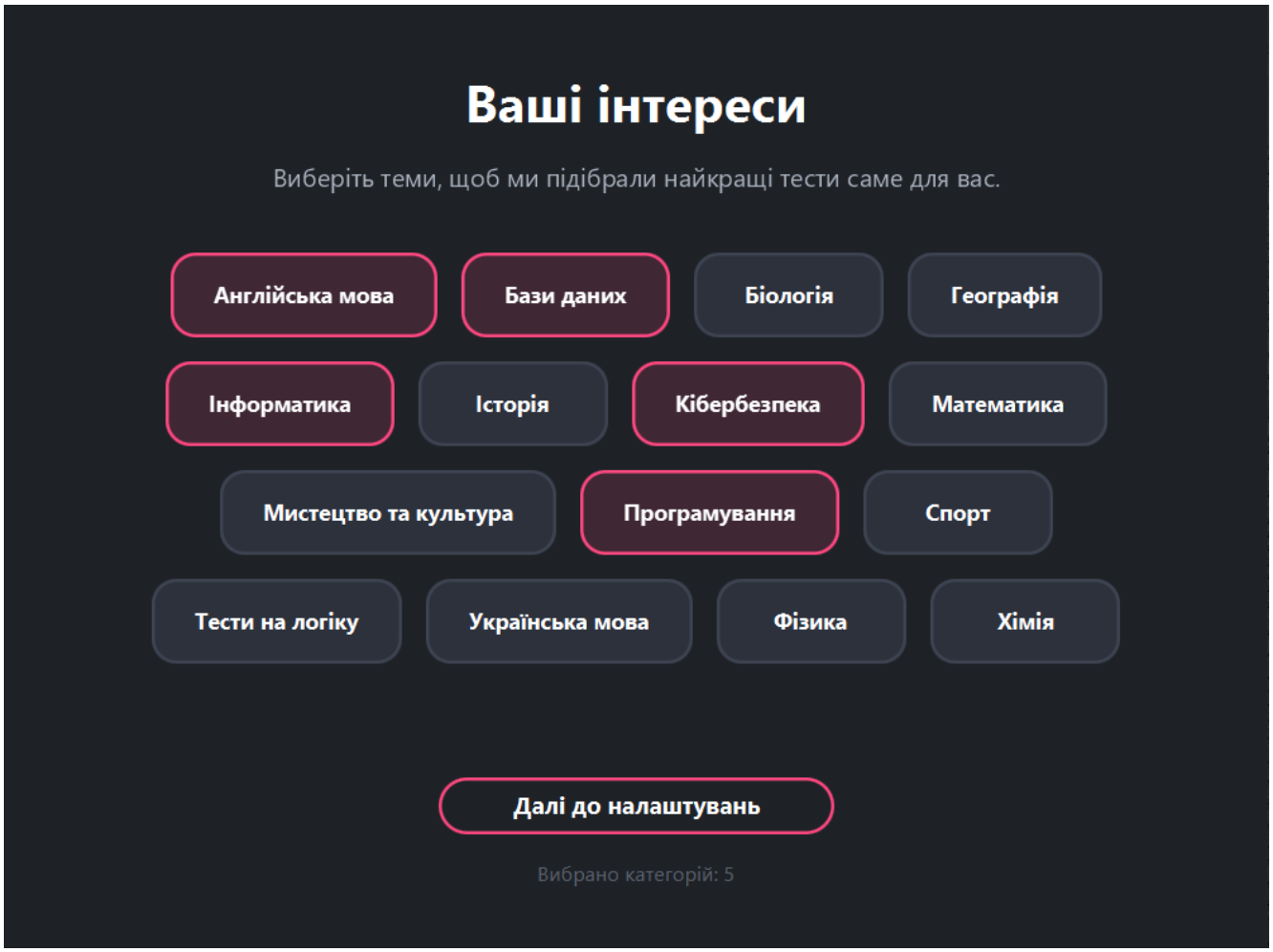

Рисунок 2.4 – Інтерфейс вибору категорій інтересів користувача

Це вікно призначене для персоналізації контенту та формування стрічки рекомендованих тестів на основі вподобань користувача. Користувач відмічає цікаві йому тематики (наприклад, "Програмування", "Бази даних", "Кібербезпека") за допомогою інтерактивних тегів та в подальшому на головному вікні буде бачити лише популярні тести саме по обраних категоріях, в майбутньому користувач може змінити категорії які його цікавлять в налаштуваннях профілю. Після натискання кнопки «Далі», обрані інтереси зберігаються в базі даних і відкривається фінальне вікно онбордингу.

Завершальним етапом онбордингу є налаштування каналів комунікації та сповіщень (рис. 2.5).

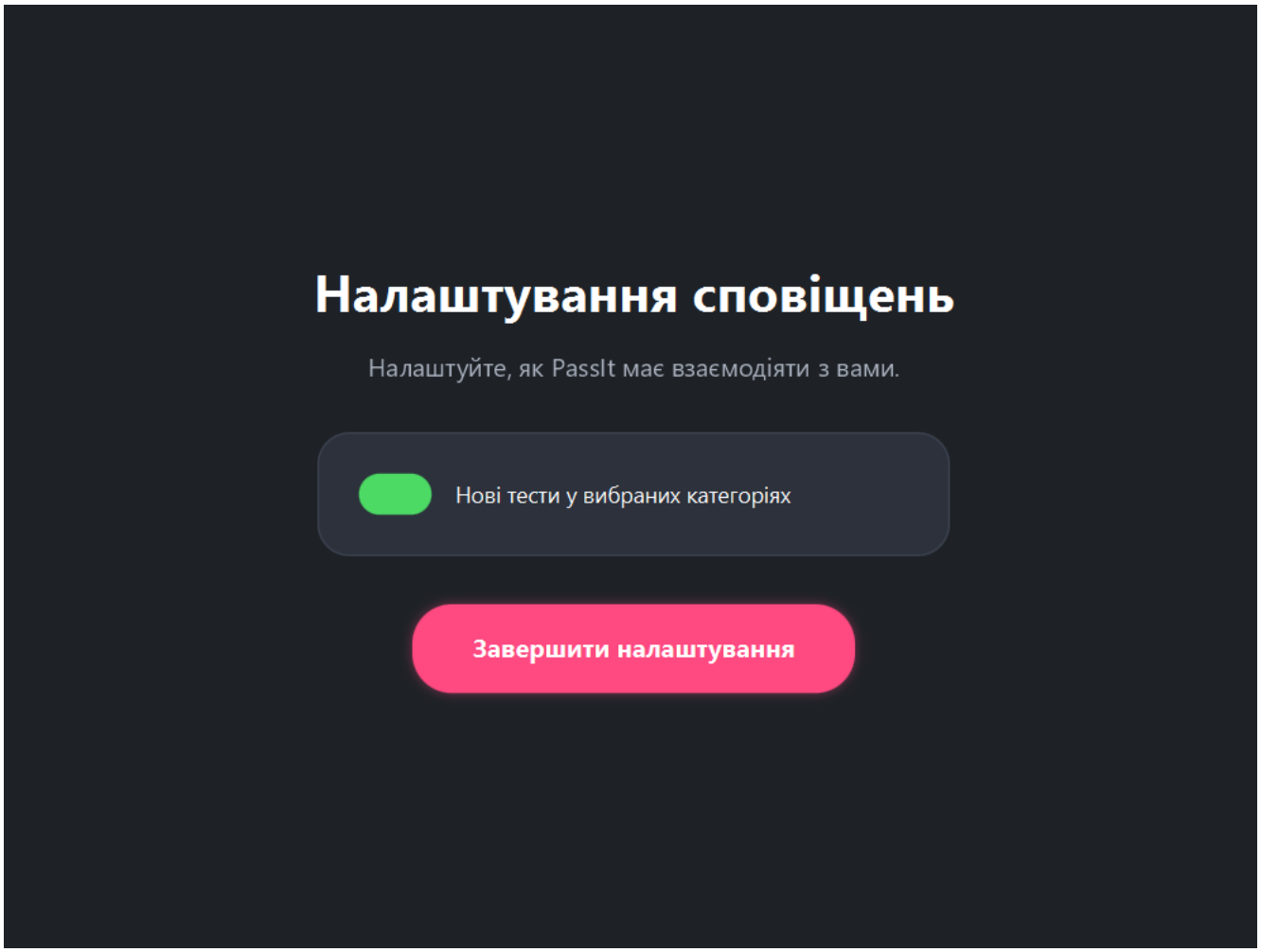

Рисунок 2.5 – Вікно налаштування сповіщень користувача

Це вікно завершує процес входу нового користувача в систему та дозволяє клієнту увімкнути сповіщення про нові тести у обраних категоріях.

Після завершення всіх етапів онбордингу користувач потрапляє до головного вікна (рис. 2.6), яке є центральним вікном для доступу до навчального контенту. Це вікно спроєктовано з акцентом на контент та швидкий пошук необхідних тестів.

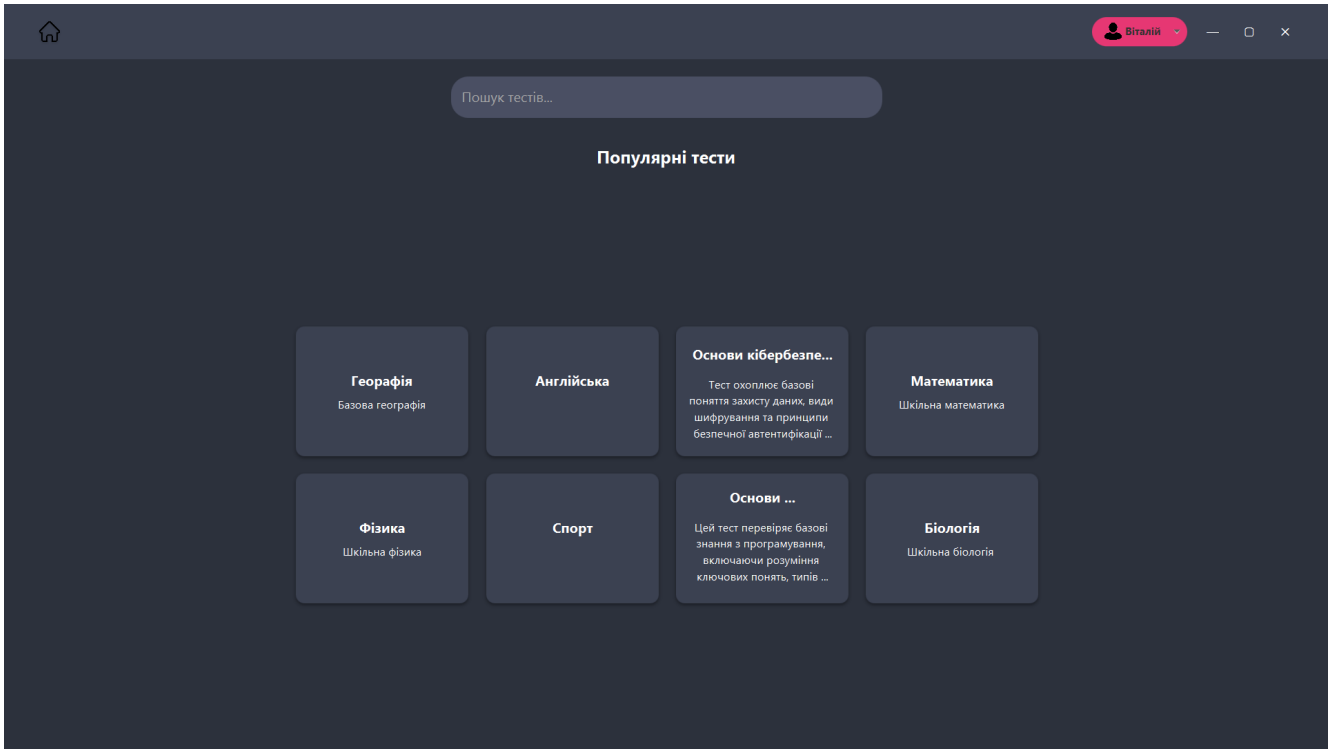

Рисунок 2.6 – Головне вікно користувача

Функціональні особливості головного екрану:

- Панель пошуку: У верхній частині розміщено інтерактивне поле пошуку, яке дозволяє користувачеві миттєво фільтрувати доступні тести за назвою або ключовими словами.
- Динамічна сітка тестів: Тести представлені у вигляді карток. Кожна картка містить назву тесту та короткий опис, картки у списку сортуються по популярності тесту згідно загальної кількості проходжень, максимально таких тестів які можуть відобразитись одночасно - 8.
- Профіль користувача: У правому верхньому куті відображається ім'я користувача та аватар. Це меню забезпечує швидкий доступ до взаємодії з аккаунтом.
- Врахування інтересів: Блок «Популярні тести» формується з урахуванням раніше обраних категорій.

При виборі конкретної картки на головному екрані, користувачеві відкривається вікно з розширеною інформацією про обраний користувачем тест (рис. 2.7).

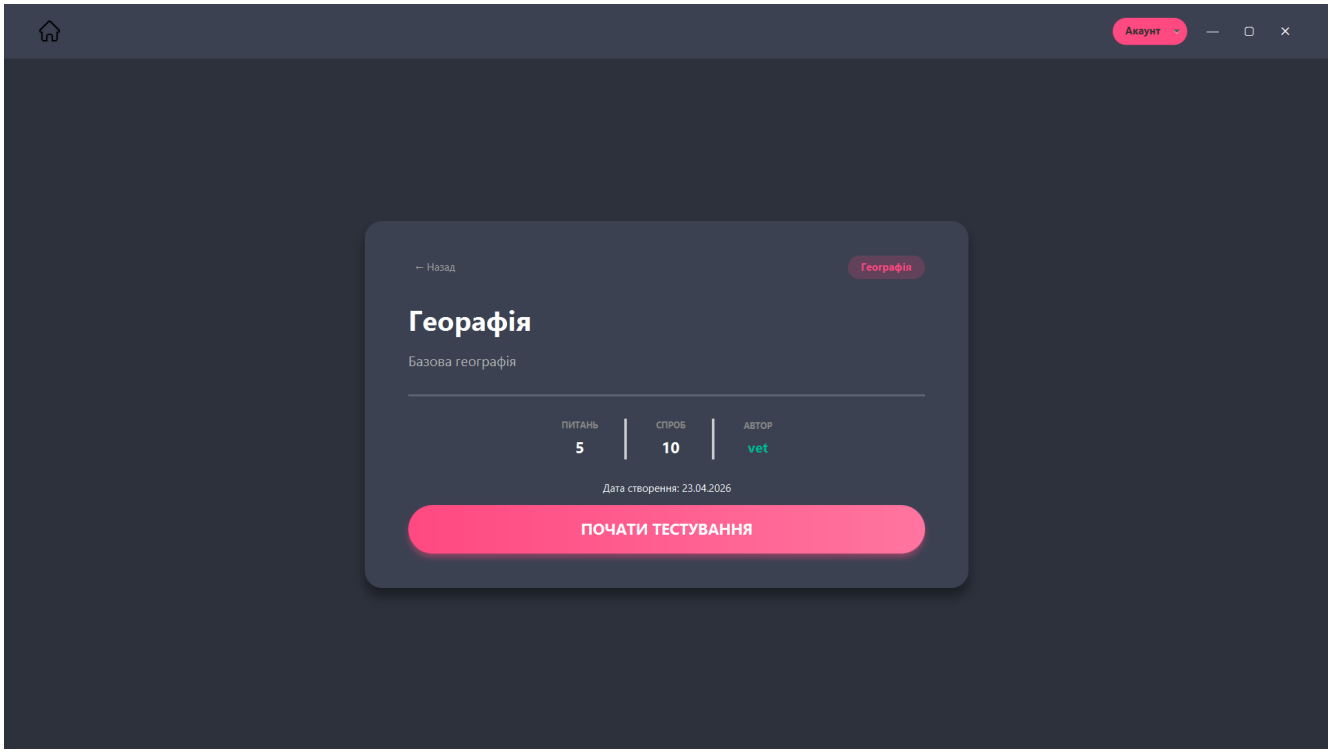

Рисунок 2.7 – Вікно перегляду деталей тесту перед початком

Функціональні можливості вікна:

- Інформаційна довідка: Вікно показує ключові дані про відкритий тест, а саме назву, короткий опис тесту, кількість питань, загальну кількість спроб проходження тесту, зроблених усіма користувачами, та ім’я автора тесту.
- Дата актуальності: Відображення дати створення тесту дозволяє користувачеві оцінити актуальність запропонованого матеріалу для проходження.
- Навігація: Реалізовано можливість повернення до загального списку або безпосереднього переходу до виконання тесту.

Дані для цього вікна завантажуються динамічно з бази даних за ідентифікатором обраного тесту.

Процес проходження тестування реалізовано у вигляді інтерактивної сесії користувача, де користувач взаємодіє з питаннями в режимі реального часу (рис. 2.8).

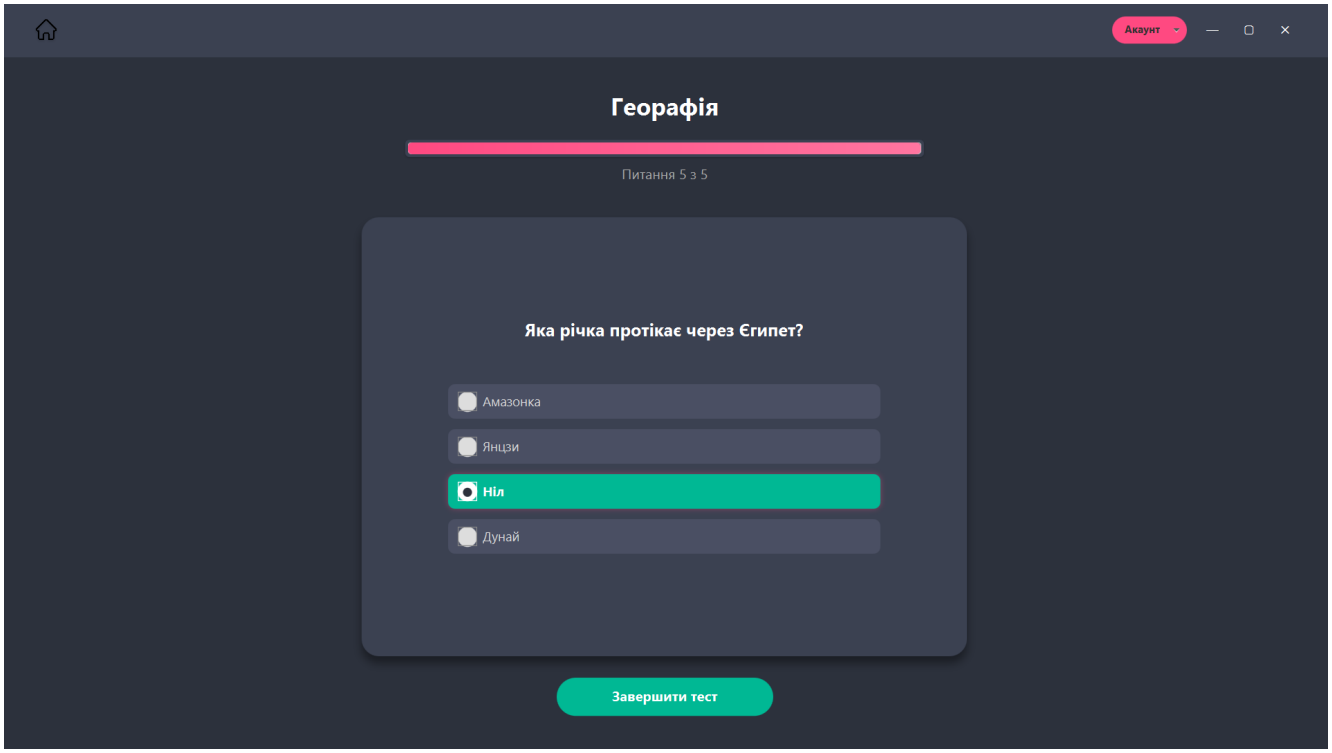

Рисунок 2.8 – Вікно активної сесії тестування

У верхній частині вікна розташовано ProgressBar, який візуалізує частку пройдених питань. Під ним міститься текстовий лічильник, що допомагає користувачеві орієнтуватися в обсязі роботи. Центральна частина вікна відведена під текст питання та варіанти відповідей. Контейнер для відповідей автоматично підлаштовується під тип питання. Всього є 4 типи можливих питань:

- Питання з однією відповіддю, дозволяє обрати лише один варіант відповіді серед усіх можливих.
- Питання з декількома відповідями, дозволяє обрати відповіді із декількох можливих варіантів.
- Питання “Правда чи неправда”, дозволяє обрати одну відповідь із двох можливих “Правда” або “Брехня”.
- Питання з відкритою відповіддю, дозволяє написати відкриту відповідь

на питання, правильність відповіді розраховується за алгоритмом відстані Лівенштейна.

На останньому питанні кнопка навігації автоматично змінює функціонал на «Завершити тест». При натисканні активується алгоритм фінального збору відповідей та проводиться транзакція в базу даних для фіксації спроби та розрахунку результату.

Особливу увагу на цьому етапі варто привернути на питання з відкритою відповіддю, обробляти їх строгим порівнянням не зовсім правильно тому в даній реалізації було використано алгоритм “Відстань Лівенштейна” для нечіткого порівняння рядків. Оскільки користувачі можуть припускатися механічних помилок, одруківок або використовувати різний регістр, використання прямого порівняння рядків є неефективним.

Для оцінювання близькості відповіді користувача до еталона використовується відстань Левенштейна. Це метрика, яка визначає мінімальну кількість операцій вставки, видалення або заміни одного символу, необхідних для перетворення одного рядка в інший.

Першочергово введена користувачем відповідь та еталонний рядок проходять стадію очищення: видаляються зайві пробіли на початку й у кінці, а всі символи зводяться до нижнього регістру. Це нівелює можливо випадкові помилки. Якщо рядки не є ідентичними, програма застосовує алгоритм Лівенштейна. Він математично визначає кількість мінімальних правок а саме замін, вставок або видалень символів, необхідних для того, щоб відповідь користувача повністю збіглася з правильним варіантом.

Отримана кількість правок конвертується у дробове значення від 0.0 до 1.0. Наприклад, якщо у слові з 10 літер допущена одна помилка, схожість становитиме 0.9 (або 90%). Після цього система порівнює отриманий коефіцієнт із встановленим порогом проходження. Такий підхід дозволяє платформі бути лояльною до користувача: замість того, щоб ставити «0» за правильну по суті відповідь із механічною помилкою, система розпізнає відповідь користувача і зараховує бал.

Реалізація цього модуля підвищує об'єктивність оцінювання, наближаючи автоматизовану перевірку до людського сприйняття тексту, що є критично важливим для освітніх платформ.

Нижче наведено лістинг коду для програмної реалізації алгоритму Лівенштейна (рис. 2.9).

```
public static double calculateSimilarity(String s1, String s2) {  1 usage  vetalbunyak
    String str1 = s1.trim().toLowerCase();
    String str2 = s2.trim().toLowerCase();

    if (str1.equals(str2)) return 1.0;

    int distance = computeLevenshteinDistance(str1, str2);
    return 1.0 - ((double) distance / Math.max(str1.length(), str2.length()));
}

private static int computeLevenshteinDistance(CharSequence lhs, CharSequence rhs) {  1 usage  vetalbunyak
    int len0 = lhs.length() + 1;
    int len1 = rhs.length() + 1;
    int[] cost = new int[len0];
    int[] newcost = new int[len0];
    for (int i = 0; i < len0; i++) cost[i] = i;
    for (int j = 1; j < len1; j++) {
        newcost[0] = j;
        for (int i = 1; i < len0; i++) {
            int match = (lhs.charAt(i - 1) == rhs.charAt(j - 1)) ? 0 : 1;
            int cost_replace = cost[i - 1] + match;
            int cost_insert = cost[i] + 1;
            int cost_delete = newcost[i - 1] + 1;
            newcost[i] = Math.min(Math.min(cost_insert, cost_delete), cost_replace);
        }
        int[] swap = cost; cost = newcost; newcost = swap;
    }
    return cost[len0 - 1];
}
```

Рисунок 2.9 – Реалізація алгоритму відстані Лівенштейна

Наступним важливим елементом розробленої системи є персональний кабінет користувача, а саме розділ з пройденими та створеними користувачем тестами (рис. 2.10).

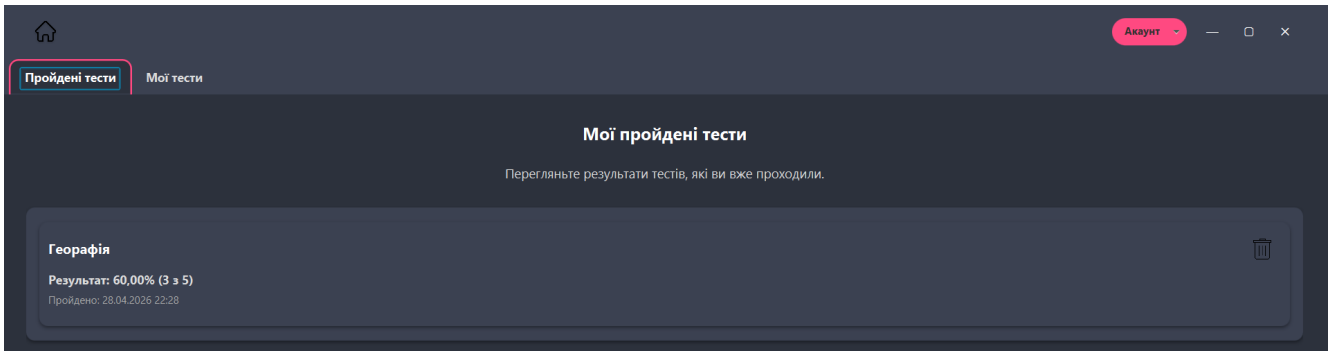

Рисунок 2.10 – Інтерфейс перегляду історії пройдених тестів

Це вікно на даному етапі розробки відображає дві вкладки “Пройдені тести” та “Мої тести”.

Вкладка “Пройдені тести” відображає історію усіх пройдених користувачем тестів та надає такий функціонал:

- Моніторинг успішності: Відображає список усіх завершених сесій тестування. Для кожного запису вказується назва тесту, дата проходження та отриманий результат у відсотках і балах.
- Інтерактивність: Користувач має можливість видалити застарілі результати з бази даних за допомогою кнопки з іконкою кошика, що дозволяє підтримувати актуальність статистики.

Вкладка “Мої тести” відображає усі раніше створені тести даним користувачем і дозволяє створити новий тест у системі. Також як і згадувалося раніше, звичайний користувач може створити лише 4 тести, це обмеження встановлено для того щоб уникнути спаму тестами та користувач раціонально використовував встановлені обмеження.

При натисканні на будь-який пройдений тест користувач може побачити результат проходження цього тесту. Це вікно є фінальним етапом у ланцюжку взаємодії користувача з тестом, де система надає розгорнутий фідбек щодо рівня засвоєння матеріалу (рис. 2.11).

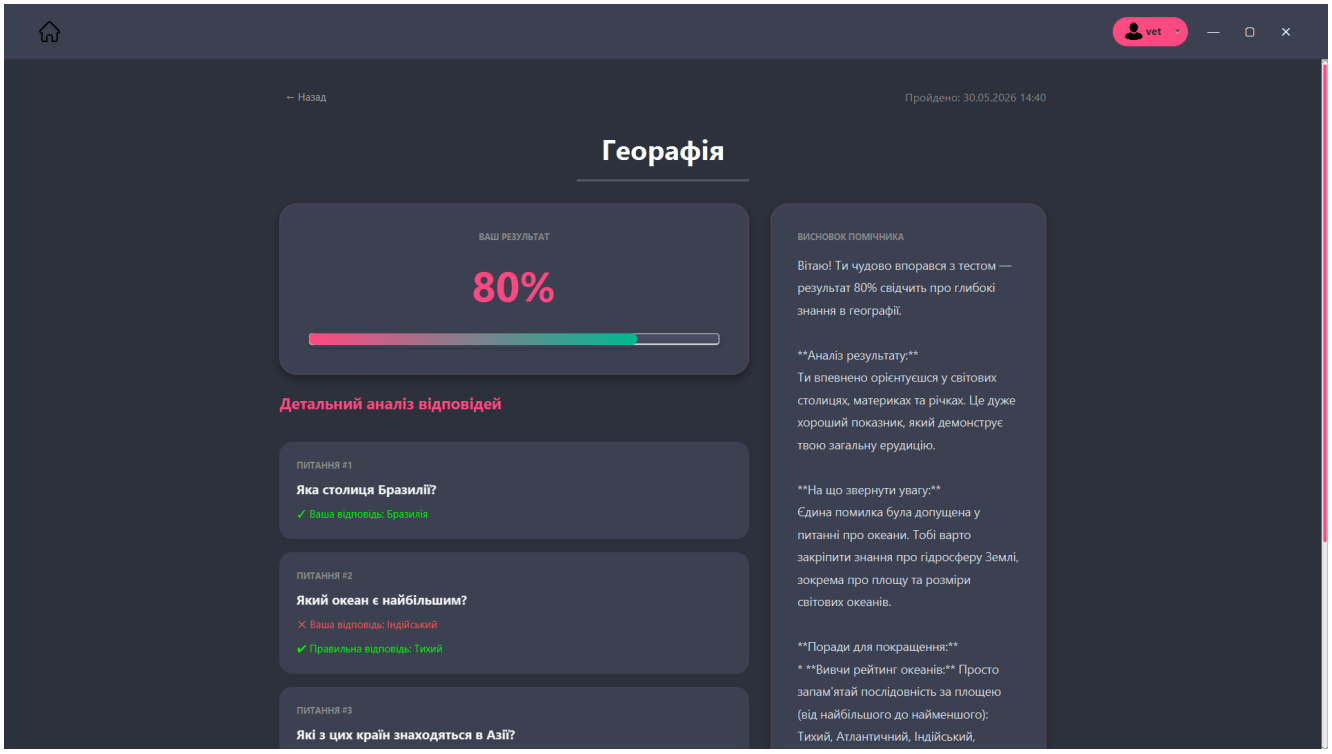

Рисунок 2.11 – Вікно детального аналізу результатів

Функціональні та візуальні елементи вікна:

- Підсумкова статистика: У центрі розміщено блок із загальним результатом у відсотках. Для кращого візуального сприйняття використано градієнтний індикатор прогресу, який кольором та заповненням показує успішність проходження.
- Блок «Детальний аналіз відповідей»: Система генерує вертикальний список карток для кожного питання тесту. Кожна картка містить номер та текст питання, маркування коректності відповіді користувача, підказку у вигляді правильної відповіді у разі помилки користувача у відповіді на дане питання.

Цей екран динамічно формується на основі даних з таблиці UserAnswers та TestAttempts. Програма ітерується по списку питань, порівнюючи збережені ID відповідей користувача з правильними ID з бази даних. Саме тут застосовується описаний раніше алгоритм Левенштейна для відкритих питань — якщо дистанція в межах норми, система підсвічує текстове поле зеленим, навіть якщо є незначні помилки, та зараховує бал у загальну статистику.

Особливістю архітектури розробленого додатка є гнучка інтеграція з моделями штучного інтелекту для генерації індивідуального зворотного зв'язку. На відміну від жорстко закодованих алгоритмів оцінювання, які лише підраховують бали, у систему впроваджено сервіс AiService. Користувач може вказати власний API-ключ у налаштуваннях профілю, що повністю знімає з розробника витрати на утримання хмарних ШІ-серверів та забезпечує конфіденційність запитів. Після завершення тесту система автоматично збирає результати спроби (тексти питань, вибрані відповіді, еталонні значення) та відправляє структурований промпт до ШІ-асистента. Згенерований аналіз помилок та персональні рекомендації щодо навчання динамічно виводяться на екран деталей спроби.

Однією з ключових функціональних особливостей платформи є вбудований інструментарій для створення авторського контенту (рис. 2.12). Це вікно реалізує концепцію відкритої платформи, де кожен користувач може виступати в ролі автора тестів.

Рисунок 2.12 – Інтерфейс конструктора створення нового тесту

Для створення тесту автор має вказати:

- Дані тесту: Автор задає назву та розширений опис тесту. Це дозволяє майбутнім користувачам зрозуміти тематику та рівень складності ще до початку проходження тесту.
- Категоризація: Випадаючий список «Оберіть категорію» забезпечує прив’язку тесту до конкретної предметної області. Це досить важливо для коректної роботи алгоритму рекомендацій, який було описано в етапі онбордингу.
- Питання: Система дозволяє обирати тип питання для кожного окремого створеного елемента (наприклад, SINGLE_CHOICE, MULTIPLE_CHOICE, TRUE_FALSE або OPEN_ENDED).

Кожне нове питання — це екземпляр окремого FXML-компонента, який завантажується в пам'ять при натисканні кнопки додавання. Це дозволяє зберігати інтерфейс легким та гнучним навіть при створенні великих тестів на десятки питань.

Після натискання кнопки «Опублікувати тест» система ініціює складну транзакцію в базі даних: дані обробляються з інтерфейсу, валідуються та зберігаються у зв'язаних таблицях.

Ще одним важливим модулем клієнтської частини застосунку є вікно налаштувань профілю (рис. 2.13), яке дозволяє користувачеві під час використання застосунку змінити параметри, задані під час первинного онбордингу.

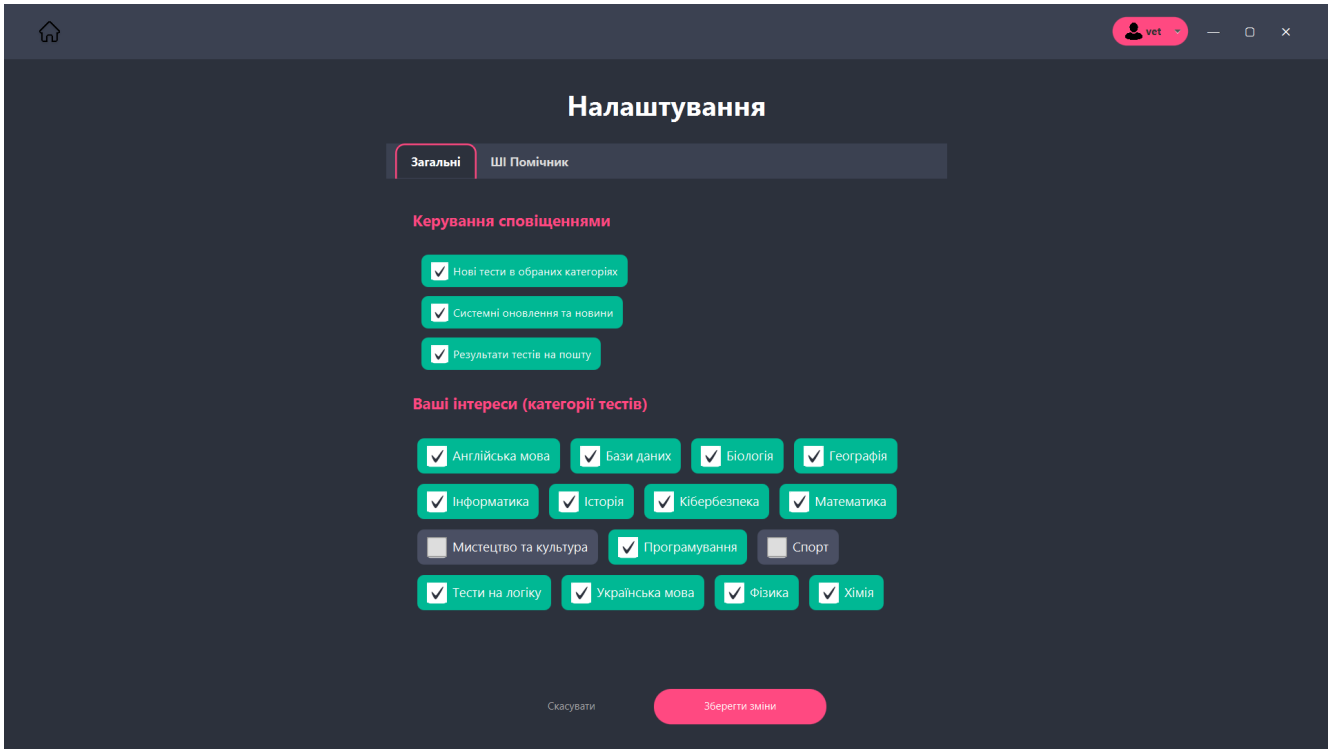

Рисунок 2.13 – Вікно налаштування сповіщень та інтересів

Вікно розділене на два логічні блоки, що охоплюють технічні та контентні аспекти роботи платформи. У верхній частині розміщено блок керування сповіщеннями, де за допомогою системи інтерактивних прапорців користувач самостійно визначає рівень інформаційної взаємодії із системою: від отримання новин про системні оновлення до автоматичної відправки результатів тестування на електронну пошту. Нижче розташована панель тематичних інтересів, яка повністю дублює функціонал вікна ідентифікації, дозволяючи змініои пріоритетні категорії тестів. Це безпосередньо впливає на алгоритм формування стрічки на головному екрані, забезпечуючи динамічну адаптацію контенту під зміну навчальних цілей користувача.

Для керування параметрами інтеграції з нейромережами в налаштуваннях програми передбачено спеціалізований розділ «ШІ Помічник» (рис. 2.14). Його основна функція це безпечне збереження даних для доступу до сторонніх AI-сервісів. Через цей екран користувач може внести свій персональний токен доступу (API-ключ). Інтерфейс розроблено з урахуванням гнучкості: якщо поле ключа залишається порожнім, програма не блокує основні функції тестування, а

просто адаптує інтерфейс результатів, виводячи підказку про можливість увімкнення розширеного аналізу. Введені дані зберігаються в профілі користувача, що дозволяє один раз налаштувати асистента й автоматично отримувати рецензії до всіх наступних тестів.

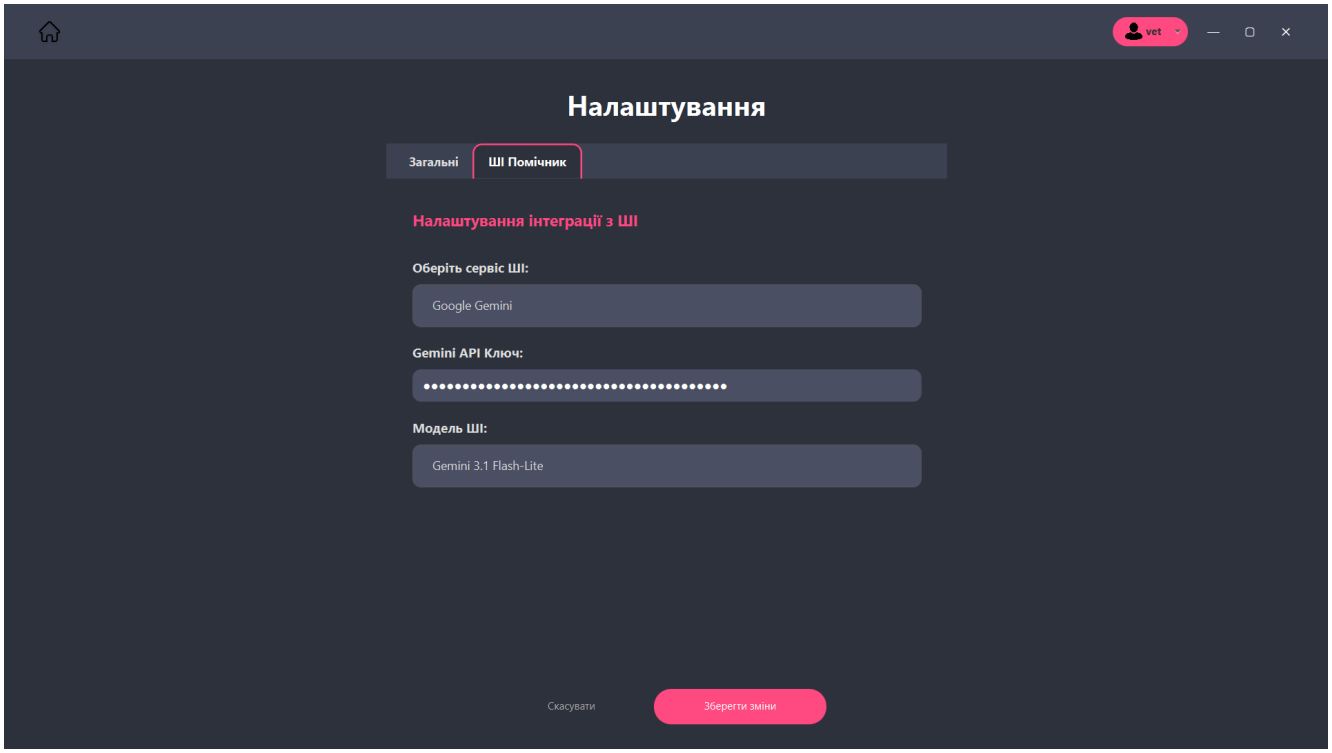

Рисунок 2.14 – Вікно налаштування інтеграції з ШІ

Одним із не менш важливих компонентів архітектури користувацького інтерфейсу є вікно «Мій Профіль та Статистика» (рис. 2.15). Цей модуль виконує функцію агрегації даних про навчальну діяльність користувача та надає інструменти для управління персональною інформацією.

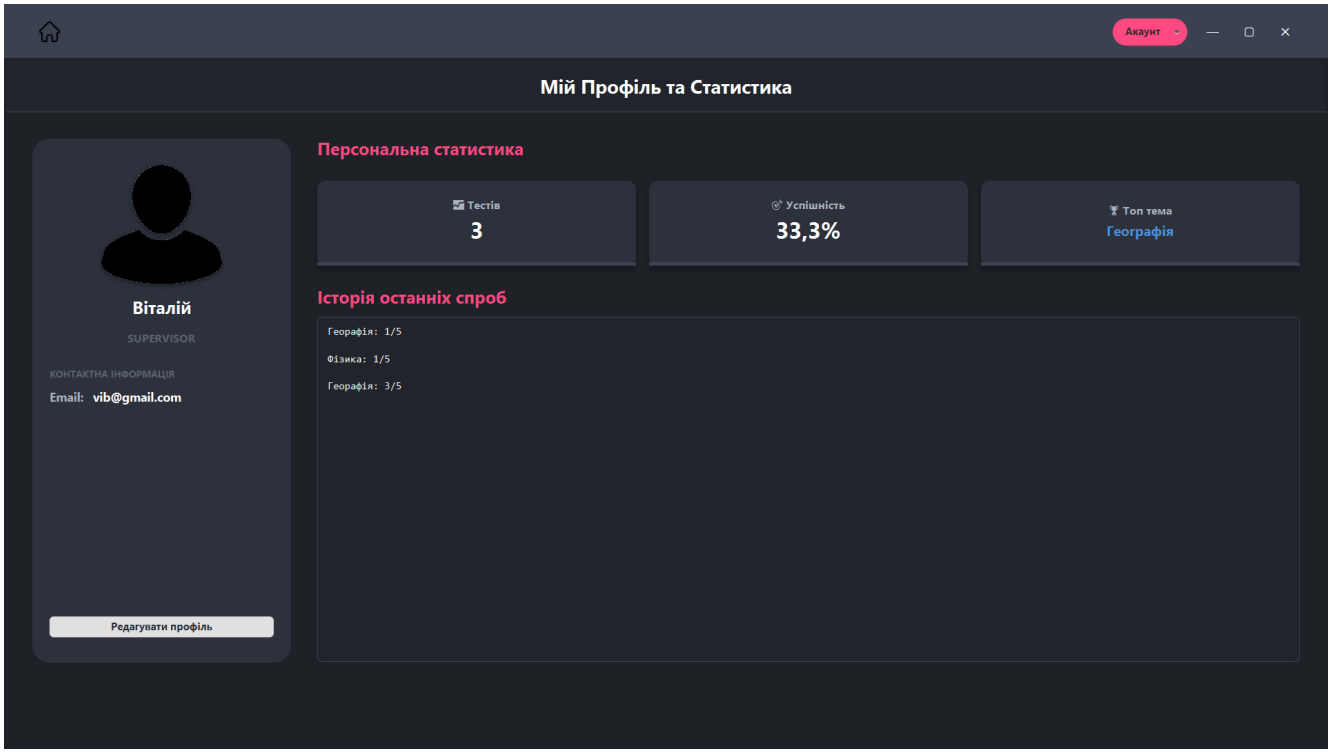

Рисунок 2.15 – Інтерфейс профілю користувача та аналітики успішності

Інтерфейс спроєктовано за принципом дашборду, що розділений на три функціональні зони:

- Панель ідентифікації: У лівій частині вікна розміщено блок візуальної ідентифікації, що включає аватар користувача, його ім'я, роль у системі та контактні дані. Тут також знаходиться кнопка «Редагувати профіль», яка дозволяє оперативно змінити персональні дані у цьому ж вікні.
- Блок ключових показників: Верхня центральна частина містить аналітичні картки, що відображають кількісні та якісні показники навчання такі як кількість пройдених тестів, відсоток успішності та улюблена категорія.
- Історія останніх спроб: Нижня частина вікна відведена під хронологічний список активностей. На відміну від детального звіту, тут подано перелік пройдених тестів, що дозволяє швидко оцінити обсяг виконаної роботи за останній час.

Дані для цього вікна формуються за допомогою агрегуючих SQL-запитів. Система автоматично розраховує середній відсоток успішності на основі всіх

записів у таблиці результатів, пов'язаних із конкретним id користувача.

Останнє важливе вікно це панель супервайзера, нижче наведено вікно для керування категоріями у системі (рис. 2.16).

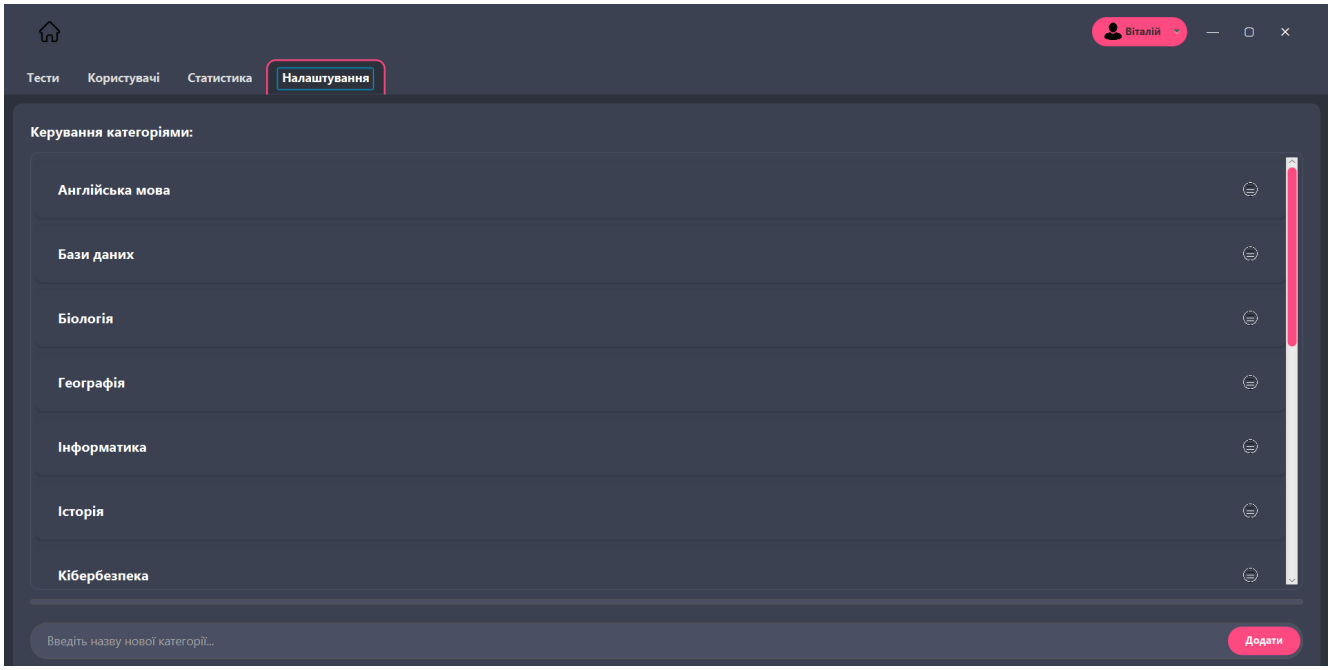

Рисунок 2.16 – Інтерфейс керування категоріями

Дане вікно доступне лише користувачам із роллю “Супервайзер”, який бачить повний перелік категорій, доступних у системі. Це дозволяє підтримувати актуальність структури тестів та уникати дублювання тем. У нижній частині вікна розміщено інструментарій для створення нових тематичних розділів. Після введення назви та натискання кнопки «Додати», нова категорія миттєво з’являється у базі даних і стає доступною для всіх користувачів (як у вікні онбордингу, так і в конструкторі тестів).

2.7 Опис вкладки «Статистика»

Ключовою технічною перевагою розробленої системи є модуль поглибленої аналітики, який доступний користувачам із привілейованою роллю «Супервайзер». Цей інструментарій перетворює платформу PassIt із простого сервісу проходження

тестів на складнішу систему підтримки прийняття рішень, яка дозволяє супервайзерам виявляти приховані закономірності у поведінці користувачів.

Перший екран аналітичного модуля (рис. 2.17) фокусується на операційній діяльності платформи. Одним із найважливіших та графіків у системі є гібридний графік активності розроблений за допомогою математичних моделей, який поєднує історичні дані з прогнозними моделями.

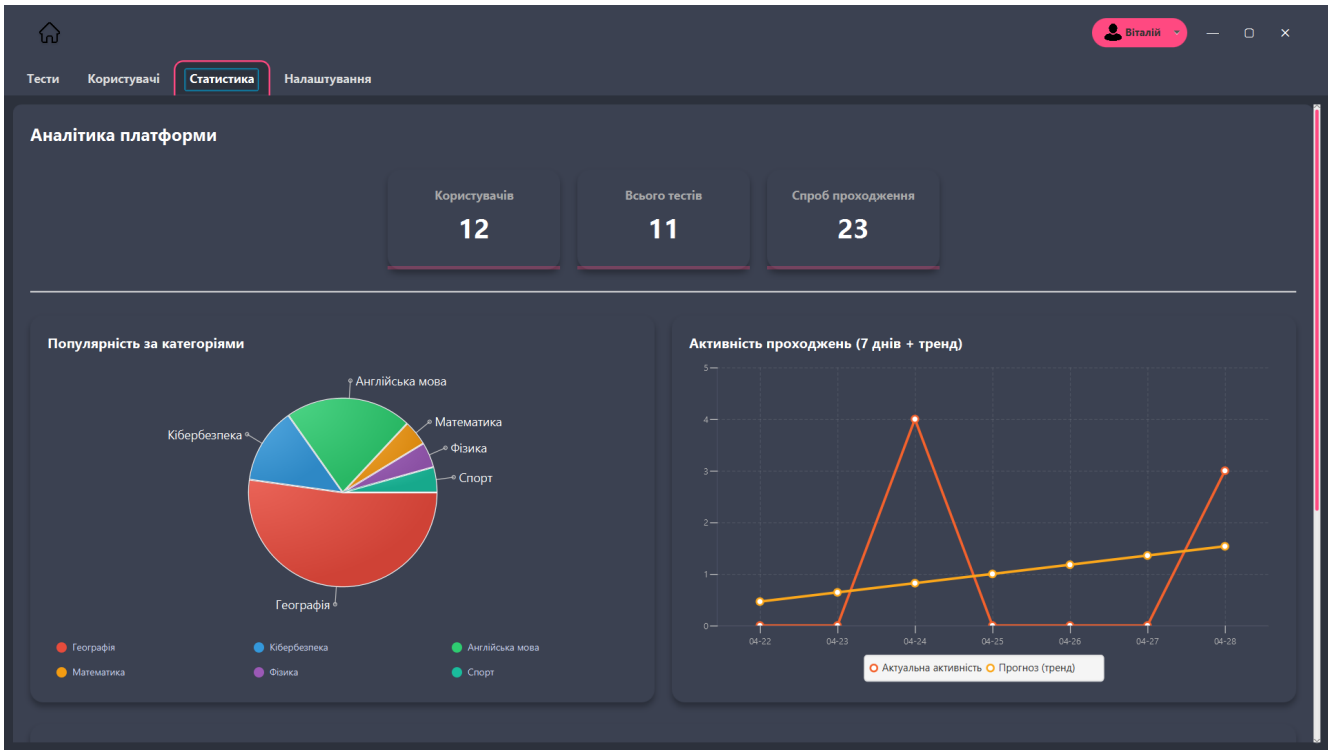

Рисунок 2.17 – Панель операційної аналітики та прогнозування трендів

На графіку «Активність проходжень» помаранчевою лінією відображено реальну кількість спроб проходжень тестів за останні 7 днів. Паралельно з нею система автоматично вираховує та будує лінію тренду (лінія з жовтими маркерами). Використання методу найменших квадратів дозволяє системі екстраполювати дані на майбутні періоди, допомагаючи супервайзеру оцінити темпи росту популярності платформи.

Кругова діаграма «Популярність за категоріями» в реальному часі відображає частку кожної категорії у загальному обсязі всіх тестувань у системі. Це дає змогу виявити «дефіцитні» теми, які потребують наповнення новими

тестами для проходження.

Агреговані метрики: Верхній блок карток забезпечує миттєвий доступ до загальних кількісних характеристик бази даних, що є базою для розрахунку конверсії та залученості користувачів.

Продовження аналітичного модуля (рис. 2.18) представляє гістограму розподілу результатів, що є основним інструментом для оцінки складності тестів та якості підготовки аудиторії.

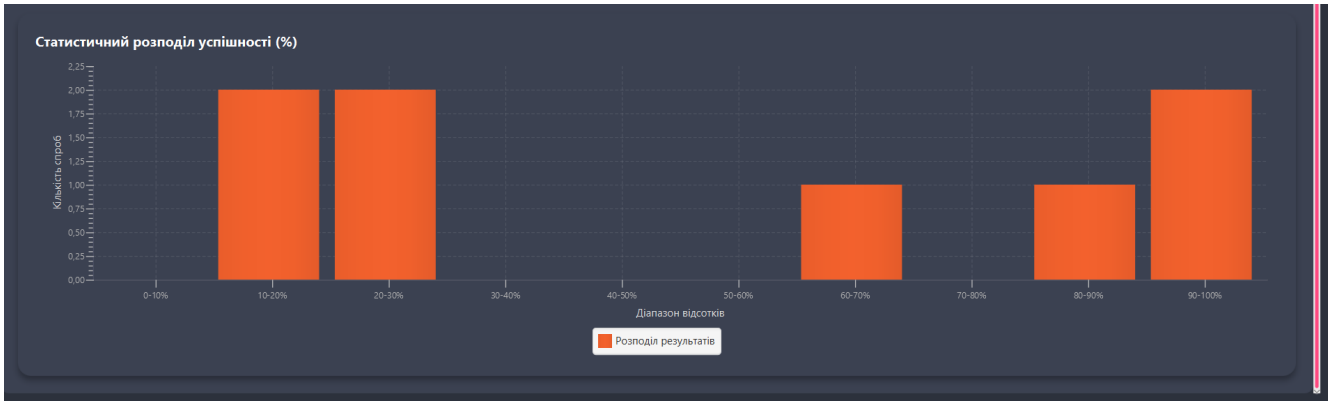

Рисунок 2.18 – Гістограма статистичного розподілу успішності користувачів

Вісь абсцис розділена на десятипроцентні діапазони успішності (від 0-10% до 90-100%), а вісь ординат показує кількість спроб, що потрапили в цей діапазон.

На наведеному прикладі спостерігається такий розподіл: значна частина користувачів має результати або в діапазоні низьких значень (10-30%), або дуже високих (90-100%). Для адміністратора це є сигналом про те, що контент тесту може бути занадто різним — або надто простим для підготовлених, або занадто складним для новачків. Такий підхід дозволяє відсіювати некоректно складені питання.

Аналітичний модуль побудований на базі бібліотеки JavaFX Charts, проте логіка обробки даних винесена на рівень SQL-запитів до бази даних. Для формування гістограми використовується функція групування, що дозволяє серверу видавати вже готові для візуалізації компоненти, не перевантажуючи клієнтську частину досить складними розрахунками. Це забезпечує високу

швидкість обробки даних навіть при наявності великої кількості записів у таблиці результатів

2.8 Висновки до розділу 2

У даному розділі було реалізовано комплексне проєктування архітектури програмної системи, що базується на фундаментальному патерні MVC. Такий підхід забезпечив високий рівень модульності додатка, дозволивши здійснити відокремлення графічного інтерфейсу користувача від внутрішньої бізнес-логіки та процедур обробки даних, що є критично важливим для забезпечення масштабованості та легкості подальшої підтримки продукту.

Кожна сутність була піддана процедурі нормалізації за встановленими стандартами, що дозволило мінімізувати дублювання інформації та гарантувати цілісність даних у межах відкритої платформи.

Особливу увагу приділено побудові об’єктно-орієнтованої моделі системи, яка була візуалізована через розгалужену діаграму класів UML. Це дало змогу наочно продемонструвати статичну структуру додатка, ієрархію контролерів та складні асоціативні зв’язки між моделями даних.

Результати проєктування, викладені у цьому розділі, формують цілісний технічний фундамент, який поєднує в собі класичні архітектурні рішення з методами інтелектуального аналізу даних, що робить платформу «PassIt» конкурентоспроможним та технологічно досконалим програмним продуктом.

РОЗДІЛ 3. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Розділ присвячено перевірці працездатності розробленої системи та умовам її практичного використання. Описано методику тестування програмних модулів, що включає поєднання ручного тестування інтерфейсу та функціональної перевірки програмної логіки, зокрема стабільності взаємодії з базою даних через пул з'єднань HikariCP та точності розрахунків модуля прогнозування. Наведено результати модульного тестування алгоритму Левенштейна, інтеграційного тестування взаємодії з хмарною базою даних на платформі Railway, а також перевірки інтерфейсу на коректність відображення.

3.1. Методика та результати тестування програмних модулів

Ключовим етапом розробки системи стало комплексне тестування, метою якого було підтвердження стабільності роботи всіх функціональних блоків та їх відповідності сформованим вимогам. Процес тестування включав поєднання методів ручного тестування інтерфейсу користувача та функціональної перевірки програмної логіки. Особлива увага приділялася надійності обміну даними між додатком та СУБД MySQL через пул з’єднань HikariCP, а також точності математичних розрахунків у модулі прогнозування. В ході тестування було змодельовано різні сценарії поведінки користувачів, зокрема введення некоректних даних при реєстрації та спроби несанкціонованого доступу до панелі Супервайзера. Система продемонструвала високу стійкість до помилок: механізми валідації успішно блокують пусті запити, а блок обробки виключень дозволяє програмі продовжувати роботу навіть при тимчасовій втраті зв'язку з базою даних. Результати перевірки ключових функцій системи систематизовано та наведено у таблиці (див. табл. 3.1).

Таблиця 3.1 Матриця результатів функціонального тестування

№	Об'єкт тестування	Тест кейс	Очікуваний результат	Статус
1	Модуль авторизації	Введення валідного логіна та пароля	Ініціалізація сесії та вхід у систему	Пройдено
2	Система безпеки	Спроба виклику панелі супервайзера роллю User	Відмова у доступі, захист даних	Пройдено
3	Логіку тестування	Вибір варіантів та завершення тесту	Розрахунок результату, запис у БД	Пройдено
4	Модуль регресії	Обробка статистики активності за тиждень	Відображення лінії тренду на графіку	Пройдено
5	Мережева стійкість	Імітація розриву з'єднання з MySQL	Виведення помилки без збою програми	Пройдено

Для забезпечення якості платформи було застосовано багаторівневий підхід до перевірки коду:

- Модульне тестування: Окрема перевірка алгоритму Лівенштейна в класі AnswerEvaluator. Тестування підтвердило, що система коректно ідентифікує правильні відповіді навіть при наявності 2–3 друкарських помилок, що є критичним для відкритої платформи.
- Інтеграційне тестування: Перевірка взаємодії між Java-додатком та хмарною базою даних Railway. Тестування пулу HikariCP підтвердило стабільність з’єднань при імітації високого мережевого завантаження та автоматичне відновлення сесій після розриву зв’язку.
- Тестування інтерфейсу (UI/UX): Перевірка коректності відображення анімованих переходів TestCardAnimator та адаптивності компонентів JavaFX на екранах з різною роздільною здатністю.

3.2 Впровадження системи та інструкція користувача

Впровадження програмного продукту «PassIt» орієнтоване на максимальну простоту для кінцевого користувача завдяки використанню хмарної

інфраструктури. Нижче наведено регламент взаємодії з основними модулями системи.

Інструкція для користувача:

1) Авторизація: При запуску програми необхідно ввести облікові дані. Якщо ви користуєтеся системою вперше, то потрібно зареєструватись та пройти процедуру онбордингу, вказавши своє ім'я та обравши пріоритетні категорії тестів.
2) Вибір тесту: На головному екрані скористайтеся пошуком або оберіть тест із запропонованих категорій. Натисніть на картку тесту для перегляду деталей (кількість питань, автор, дата створення).
3) Проходження тесту: У вікні тестування обирайте варіанти відповідей або вводьте текст у полі для відкритих питань. Система автоматично враховує можливі неточності завдяки вбудованому алгоритму нечіткого порівняння.
4) Аналіз результатів: Після завершення натисніть «Завершити тест». У вікні результатів ви зможете побачити свій відсоток успішності та переглянути детальний розбір кожної відповіді з підказками щодо помилок.

Інструкція для супервайзера (адміністратора):

1) Моніторинг аналітики: Перейдіть у розділ «Статистика» для перегляду глобальних показників платформи. Використовуйте лінійні графіки з лінією тренду для прогнозування активності та гістограми для аналізу якості тестових завдань.
2) Керування категоріями: У вкладці «Налаштування» ви можете додавати нові предметні категорії або редагувати вже існуючі, що миттєво оновить фільтри для всіх користувачів системи.

3.3 Системні вимоги та регламент підтримки

Для забезпечення стабільної експлуатації та безперебійної роботи

програмної системи «PassIt» середовище розгортання повинно відповідати встановленим технічним параметрам. Оскільки додаток розроблено на мові програмування Java, він є кросплатформенним, проте вимагає наявності встановленого пакету Java Runtime Environment (JRE) версії 17 або новішої.

Ключовою особливістю впровадження даного продукту є використання хмарної інфраструктури платформи Railway, де розгорнуто сервер бази даних MySQL. Таке рішення дозволяє уникнути необхідності встановлення та адміністрування локального сервера бази даних на кожному робочому місці, забезпечуючи глобальну доступність даних та високу надійність їх зберігання.

Апаратні вимоги до робочої станції користувача є помірними: для швидкого відгуку графічного інтерфейсу JavaFX та коректної візуалізації складних аналітичних графіків достатньо процесора з тактовою частотою від 2.0 ГГц та 4 ГБ оперативної пам’яті. Обов’язковою умовою функціонування системи є наявність стабільного інтернет-з’єднання для підтримки зв'язку з віддаленим сервером через пул з’єднань HikariCP.

Регламент технічної підтримки передбачає періодичне оновлення бібліотек через систему автоматизації збірки Maven для підтримання актуального рівня безпеки, а також моніторинг ресурсів на платформі Railway для запобігання перевантаженню бази даних при зростанні кількості активних сесій.

Використання хмарного розгортання в поєднанні з гнучкою архітектурою додатка гарантує цілісність інформації та стабільну роботу системи навіть при зміні технічного стану локального обладнання користувачів. Процес впровадження системи складався з наступних технічних кроків:

- Конфігурація БД: Створення реляційної структури в хмарі Railway та налаштування прав доступу для віддалених підключень.
- Налаштування середовища: Підготовка файлів конфігурації для HikariCP, що містять URL-адресу сервера, логін та пароль у зашифрованому вигляді через ConfigManager.
- Збірка виконуваного файлу: Використання Maven для створення виконавчого JAR-файлу, що містить усі необхідні залежності.

3.4 Висновки до розділу 3

У цьому розділі було проведено цикл тестування розробленої системи та визначено умови її успішного впровадження. Результати функціонального тестування підтвердили високу надійність архітектурних рішень, зокрема стабільність взаємодії з хмарною базою даних та точність роботи математичного модуля прогнозування на основі лінійної регресії.

Було доведено, що система коректно розмежовує права доступу між користувачем та супервайзером, забезпечуючи цілісність даних. Аналіз проведеної перевірки дозволяє стверджувати, що програмний продукт «PassIt» є технічно завершеним, стійким до помилок та повністю відповідає поставленому завданню.

3.5 Навантажувальне тестування

Оскільки архітектура «PassIt» передбачає одночасну роботу багатьох користувачів та віддалене підключення до хмарної СУБД MySQL на платформі Railway, критично важливим етапом перевірки стало навантажувальне тестування. Головною метою цього етапу було визначення межі пропускної здатності пулу з'єднань HikariCP та оцінка стабільності системи при масових запитах.

Тестування проводилося за допомогою спеціалізованого інструментарію Apache JMeter. Для перевірки було розроблено сценарій, який імітував одночасне виконання запитів до таблиць Tests та Questions.

Параметри тестових сценаріїв та отримані результати:

- Сценарій «Стандартне навантаження»: Змодельовано роботу 50 активних користувачів, які надсилали запити протягом 60 секунд. Середній час відгуку системи становив 120 мс, пропускна здатність — 45 запитів/сек. Відсоток помилок дорівнював 0%. Пул HikariCP ефективно розподіляв потоки без затримок.
- Сценарій «Пікове навантаження»: Кількість віртуальних потоків було збільшено до 200 одночасних з'єднань. При цьому середній час

відповіді зріс до 380 мс через мережеві затримки хмарного хостингу Railway, проте система зберегла повну стабільність, а показник помилок не перевищив 0.5%.

Результати навантажувального тестування підтвердили правильність конфігурації пулу з'єднань, який запобігає блокуванню транзакцій та перевантаженню віддаленої бази даних. Програма успішно витримує потоки запитів, що гарантує надійне функціонування платформи в умовах реального навчального процесу.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У даному розділі розглянуто надзвичайні ситуації техногенного, природного та соціально-політичного характеру, що можуть виникати в процесі трудової діяльності, а також заходи з їх профілактики. Окрему увагу приділено організації навчання працівників та проведенню інструктажів з охорони праці відповідно до чинного законодавства України.

4.1 Надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природними причинами

Розробка та експлуатація програмного забезпечення здійснюється в умовах, які можуть бути схильні до виникнення надзвичайних ситуацій (НС) різного характеру.

Надзвичайні ситуації (НС) – це порушення нормальних умов життя і діяльності людей, що призводить або може призвести до загибелі людей, значних матеріальних втрат та/або суттєвого погіршення стану довкілля.

НС класифікуються за характером походження на чотири основні класи, кожен з яких поділяється на групи та види. НС поділяються на загальнодержавний, регіональний, місцевий та об'єктовий рівні.

Надзвичайні ситуації техногенного характеру**:** Це НС, пов'язані з порушеннями у виробничій, транспортній, енергетичній сферах, а також з аваріями на інженерних мережах. Такими ситуаціями можуть бути:

- Аварії з викидом небезпечних хімічних, радіоактивних, біологічних речовин: Можливі при розміщенні робочого місця поблизу промислових об'єктів.
- Раптове руйнування споруд та будівель: Може статися внаслідок зносу конструкцій, порушення будівельних норм або зовнішніх впливів.
- Аварії на інженерних мережах: Відключення електроенергії, водопостачання, опалення, каналізації, аварії систем зв'язку та

телекомунікацій.

Профілактичні дії:

- Ознайомлення з планами евакуації та цивільного захисту будівлі/організації.
- Забезпечення безперебійного живлення для критично важливого обладнання.
- Дотримання інструкцій адміністрації або рятувальних служб у разі аварії.
- Надання першої допомоги постраждалим, якщо це можливо і безпечно.

Надзвичайні ситуації природного характеру: Це НС, спричинені небезпечними геологічними, метеорологічними, гідрологічними та іншими природними явищами. До таких ситуацій відносяться:

- Небезпечні геологічні явища: Зсуви, обвали, що можуть загрожувати будівлям та життям людей.
- Метеорологічні явища: Сильні буревії, шквали, снігопади, ожеледиця, зливи з ризиком підтоплення, блискавки. Можуть призвести до руйнувань, обривів електромереж.
- Природні пожежі: Лісові та торф'яні пожежі, що можуть спричинити задимлення, погіршення якості повітря.
- Інфекційна захворюваність людей: Епідемії, пандемії, що можуть призвести до значного погіршення умов життєдіяльності та обмежень.

Профілактичні заходи та дії:

- Регулярне відстеження прогнозів погоди та попереджень про НС.
- Відключення електроприладів від мережі під час грози.
- У разі землетрусу – сховатися у безпечному місці або евакуюватися з небезпечного місця.
- Дотримання санітарно-гігієнічних норм для профілактики інфекційних захворювань.

Надзвичайні ситуації соціально-політичного характеру: Це НС, пов'язані з протиправними діями терористичного та антиконституційного спрямування, що

можуть загрожувати громадській безпеці та життю громадян. До них належать:

- Збройні напади, захоплення об'єктів: Терористичні акти, захоплення будівель, що створює пряму загрозу життю та здоров'ю людини.
- Встановлення вибухового пристрою в громадському місці: Загроза вибуху, що вимагає негайної евакуації та дотримання вказівок правоохоронних органів.
- Виявлення застарілих боєприпасів: Знаходження вибухонебезпечних предметів часів минулих воєн, що вимагає негайного повідомлення ДСНС та правоохоронних органів.

Профілактичні заходи:

- Підвищена уважність до підозрілих предметів або осіб.
- Негайно повідомляти правоохоронні органи (102) або ДСНС (101) про будь-які виявлені підозрілі об'єкти чи ситуації.
- У разі загрози терористичного акту – діяти згідно з інструкціями правоохоронних органів, зберігати спокій, допомагати іншим, евакуюватися за вказівками.
- Навчання правилам поведінки при виявленні вибухонебезпечних предметів: не чіпати та повідомити фахівців про виявлення небезпечного предмету.

Надзвичайні ситуації воєнного характеру: В сучасних умовах НС воєнного характеру є реальною і значною загрозою для людини. Їхній вплив на безпеку життєдіяльності є першочерговим. Ці ситуації пов'язані з наслідками застосування зброї різного типу, що може призвести до масштабних руйнувань. Основні джерела небезпек у воєнний час:

1) Застосування зброї:
    - Зброя масового ураження: Ядерна, хімічна, біологічна зброя. Її використання призводить до масового ураження населення на великих територіях.
    - Звичайна зброя: Артилерійські обстріли, ракетні удари,

авіабомбардування.

- Засоби радіоелектронної боротьби: Хоча безпосередньо не руйнують споруди, можуть впливати на функціонування критичних систем зв'язку та навігації.

2) Антисанітарна обстановка та епідемії:

- Під час бойових дій порушується робота комунальних служб, що призводить до погіршення якості води, накопичення відходів.
- Масова загибель людей та тварин, неможливість своєчасного поховання, призводить до розповсюдження трупної отрути та інфекцій.
- Зростання популяції гризунів, комах та інших переносників небезпечних хвороб.
- Недостатнє медичне обслуговування, нестача медичних препаратів, що створює сприятливі умови для виникнення та поширення епідемій.

Війна є джерелом комплексних та взаємопов'язаних небезпек. Завданням безпеки життєдіяльності є максимально ефективний захист життя.

Управління ризиком НС базується на концепції прийнятного ризику, що вимагає виявлення небезпек, оцінки їх ймовірності та наслідків, а також розробки заходів для зниження ризику до припустимого рівня.

4.2 Навчання працюючих та інструктажі з охорони праці

Важливим елементом системи управління охороною праці на підприємстві є організація навчання працівників та проведення інструктажів з питань охорони праці. Основною метою такого навчання є формування у працівників знань і навичок безпечної роботи, попередження виробничого травматизму та професійних захворювань.

Відповідно до Закону України «Про охорону праці» та Типового положення

про порядок проведення навчання і перевірки знань з питань охорони праці (НПАОП 0.00-4.12-05), усі працівники під час прийняття на роботу та в процесі трудової діяльності повинні проходити навчання, інструктажі та перевірку знань з охорони праці.

Залежно від характеру та часу проведення інструктажі поділяються на такі види:

- Вступний інструктаж проводиться з усіма працівниками, які приймаються на роботу незалежно від освіти, стажу роботи та посади. Під час інструктажу працівників ознайомлюють із загальними вимогами охорони праці, правилами пожежної безпеки, електробезпеки та виробничої санітарії.
- Первинний інструктаж на робочому місці проводиться перед початком самостійної роботи працівника. Працівника ознайомлюють із особливостями робочого місця, правилами експлуатації обладнання, можливими небезпечними факторами та засобами індивідуального захисту.
- Повторний інструктаж проводиться для закріплення знань і перевірки дотримання працівниками вимог безпеки праці. Для працівників, діяльність яких пов’язана з використанням комп’ютерної техніки та офісного обладнання, його доцільно проводити не рідше одного разу на шість місяців.
- Позаплановий інструктаж проводиться у разі зміни нормативних документів з охорони праці, впровадження нового обладнання чи технологій, а також після порушення працівником вимог безпеки праці.
- Цільовий інструктаж проводиться при виконанні разових робіт, ліквідації наслідків аварій або під час виконання робіт, які потребують спеціального дозволу чи наряду-допуску.

Для працівників сфери програмної інженерії особливе значення має дотримання вимог безпечної роботи з персональними комп’ютерами. Робоче місце

повинно відповідати ергономічним вимогам, забезпечувати належне освітлення, вентиляцію та безпечне розташування технічного обладнання. Працівники повинні дотримуватися встановлених режимів праці та відпочинку для зниження навантаження на органи зору та опорно-руховий апарат.

Факт проведення інструктажів реєструється у відповідних журналах обліку, де зазначаються дата проведення, вид інструктажу, прізвище особи, яка його проводила, та підпис працівника. Систематичне навчання і контроль знань працівників з питань охорони праці сприяють створенню безпечних умов праці та зменшенню ризику виникнення нещасних випадків на підприємстві.

Особливу роль у забезпеченні безпеки праці відіграє система періодичного навчання та перевірки знань працівників. Перевірка знань з питань охорони праці проводиться у формі тестування, співбесіди або інших способів контролю засвоєння нормативних вимог. Працівники, які не пройшли перевірку знань або отримали незадовільні результати, повинні пройти повторне навчання та повторну перевірку знань у встановлені терміни.

На підприємстві відповідальність за організацію навчання та проведення інструктажів покладається на роботодавця або уповноважену ним особу. Безпосереднє проведення інструктажів здійснюють керівники структурних підрозділів, які пройшли відповідне навчання та перевірку знань з питань охорони праці. Вони контролюють дотримання працівниками вимог безпеки, правильність використання обладнання та своєчасність проходження необхідних інструктажів.

Необхідною складовою навчання є ознайомлення працівників з порядком дій у разі виникнення надзвичайних ситуацій. Працівники повинні знати порядок евакуації з приміщення, правила користування первинними засобами пожежогасіння, способи надання домедичної допомоги потерпілим та алгоритм повідомлення відповідних служб про аварійні ситуації. Регулярне проведення тренувань і навчань з евакуації дозволяє сформувати практичні навички дій у небезпечних ситуаціях.

Таким чином, навчання працівників та проведення інструктажів з охорони праці є невід'ємною складовою системи управління безпекою праці на

підприємстві. Своєчасне та якісне навчання персоналу, контроль знань і постійне вдосконалення заходів з охорони праці забезпечують створення безпечних умов праці, збереження здоров'я працівників та підвищення ефективності виробничої діяльності.

ВИСНОВКИ

У ході виконання дипломної роботи було проведено повний цикл розробки та впровадження програмної системи «PassIt» яка є відкритою багатокористувацькою платформою, призначеною для вільного створення, розповсюдження та проходження тестів у різних предметних галузях. На відміну від закритих корпоративних рішень, розроблений продукт орієнтований на формування відкритої системи знань, де кожен авторизований користувач отримує інструментарій як для перевірки власних навичок, так і для авторської діяльності.

Актуальність проєкту підтверджується зростаючим попитом на сервіси самоосвіти та інтерактивного дозвілля. В процесі дослідження ринку було виявлено необхідність у платформі, яка б поєднувала легкість соціальних мереж із глибокою аналітикою професійних систем тестування. Розроблена система «PassIt» успішно вирішує це завдання, надаючи користувачам інтуїтивний інтерфейс для взаємодії з контентом та потужні алгоритмічні рішення для обробки результатів.

Проєктна частина роботи базувалася на об'єктно-орієнтованому підході та використанні архітектурного шаблону MVC. Це дозволило створити гнучку структуру, де бізнес-логіка повністю відокремлена від візуального представлення на JavaFX. Реалізація компонентно-орієнтованого інтерфейсу забезпечила високу швидкість розробки та можливість динамічного масштабування платформи: додавання нових категорій, типів питань або аналітичних віджетів не потребує перебудови всієї архітектури.

Особливу увагу в роботі приділено якості взаємодії користувача з платформою. Впровадження інтелектуального модуля на основі алгоритму Лівенштейна дозволило вирішити проблему «жорсткої» перевірки відкритих питань, що часто є бар’єром на подібних ресурсах. Завдяки цьому система розпізнає намір користувача навіть при наявності технічних помилок введення, що робить процес проходження тестів більш комфортним.

Вибір хмарної архітектури забезпечив «PassIt» глобальну доступність та стабільність. Використання пулу з’єднань та асинхронної обробки даних гарантує

швидкий відгук інтерфейсу навіть при складних аналітичних запитах. Результати проведеного тестування підтвердили повну відповідність системи заявленим вимогам.

За результатами розробки можна стверджувати, що створена платформа «PassIt» є цілісним та конкурентоспроможним програмним продуктом. Вона відкриває нові можливості для спільноти користувачів у сфері вільного обміну інтелектуальним контентом. У подальшому система може бути доповнена та монетизована. Проведений обсяг робіт повністю відповідає технічному завданню, а реалізовані рішення мають практичну цінність та готові до повноцінного запуску у відкритий доступ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с.

2. UML [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml.org/

3. Java Language Specification [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/jls/se11/html/jls-1.html.

4. Maven Documentation [Електронний ресурс]. – 2025. – Режим доступу до ресурсу: https://maven.apache.org/guides/.

5. Петрик М. Р. Моделювання програмного забезпечення: науково-методичний посібник [Електронний ресурс] / М. Р. Петрик, О. Ю. Петрик. – 2015. – Режим доступу: http://elartu.tntu.edu.ua/handle/123456789/17796

6. Методичні вказівки для написання розділу “Безпека життєдіяльності, основи охорони праці” в кваліфікаційних роботах здобувачів освітнього ступеня ”бакалавр”. [Електронний ресурс] Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/35902/1/Metod._%20vkazivky_%20dlya_%20napysannnya_%20rozd._%20Bezp._%20zhyttyed._.pdf

7. Apache Log4j [Електронний ресурс]. – 2025. – Режим доступу до ресурсу: https://logging.apache.org/log4j/2.x/

8. Пасічник В. В., Резніченко В. А. Організація баз даних та знань : Підручник для студентів вищих навчальних закладів. – К. : Видавнича група BHV, 2022. – 384 с.

9. Шаховська Н. Б., Нога Р. Ю. Системи управління базами даних : Навчальний посібник. – Львів : Видавництво Львівської політехніки, 2021. – 264 с.

10. Сергій Г. Архітектура програмного забезпечення: все що треба знати.

Wezom. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo obespecheniya

11. Bunyak V. PassIt : унікальний репозиторій вихідного коду програмного продукту на базі платформи GitHub. URL: https://github.com/vetalbunyak/PassIt

12. Д. Корба, І. Мудрик. Проєктування та розробка системи моніторингу рухомих об'єктів з використанням технологій Java, Spring та протоколу GTFS. Матеріали XI науково-технічної конференції „Інформаційні моделі, системи та технології". Тернопіль: ТНТУ, 2023. С. 63.

13. O. Bryk, I. Mudryk, M. Holubovskyi, Y. Stoianov. Machine learning models and methods aspects of processing unstructured data. Proceedings of the 1st International Workshop on Bioinformatics and Applied Information Technologies (BAIT 2024), Zboriv, Ukraine, 2024. pp. 64–74.

14. HikariCP — A solid high-performance JDBC connection pool. URL: https://github.com/brettwooldridge/HikariCP

15. Петрик М. Р., Петрик О. Ю. Моделювання програмного забезпечення : навчально-методичний посібник. Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. 180 с.

16. Закон України «Про охорону праці». [Електронний ресурс] Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12

17. Сокуренко В.В. Безпека життєдіяльності та охорона праці : Підручник. Харків: Харків. нац. ун-т внутр. справ. 2021. 308 с

18. ДСТУ 4933:2008 Безпека у надзвичайних ситуаціях. Техногенні надзвичайні ситуації. Терміни та визначення основних понять.

19. ДСТУ 3994-2000 Безпека в надзвичайних ситуаціях. Надзвичайні ситуації природні. Чинники фізичного походження. Терміни та визначення.

20. ДСТУ 3994-2000. Безпека в надзвичайних ситуаціях. Надзвичайні ситуації природні. Чинники фізичного походження. Терміни та визначення. Київ : Держстандарт України, 2000. 16 с.

21. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин.

22. Типове положення про порядок проведення навчання і перевірки знань

з питань охорони праці : НПАОП 0.00-4.12-05. Затв. наказом Державного комітету України з нагляду за охороною праці від 26.01.2005 № 15.

23. Правила пожежної безпеки в Україні. Затв. наказом Міністерства внутрішніх справ України від 30.12.2014 № 1417.

24. Бедрій Я.І. Основи охорони праці : навч. посіб. 4-е вид. перероб. і доп. Тернопіль : Навчальна книга – Богдан, 2018. 240 с

ДОДАТКИ

ДОДАТОК А. Повна діаграма класів згенерована Intellij Idea Ultimate

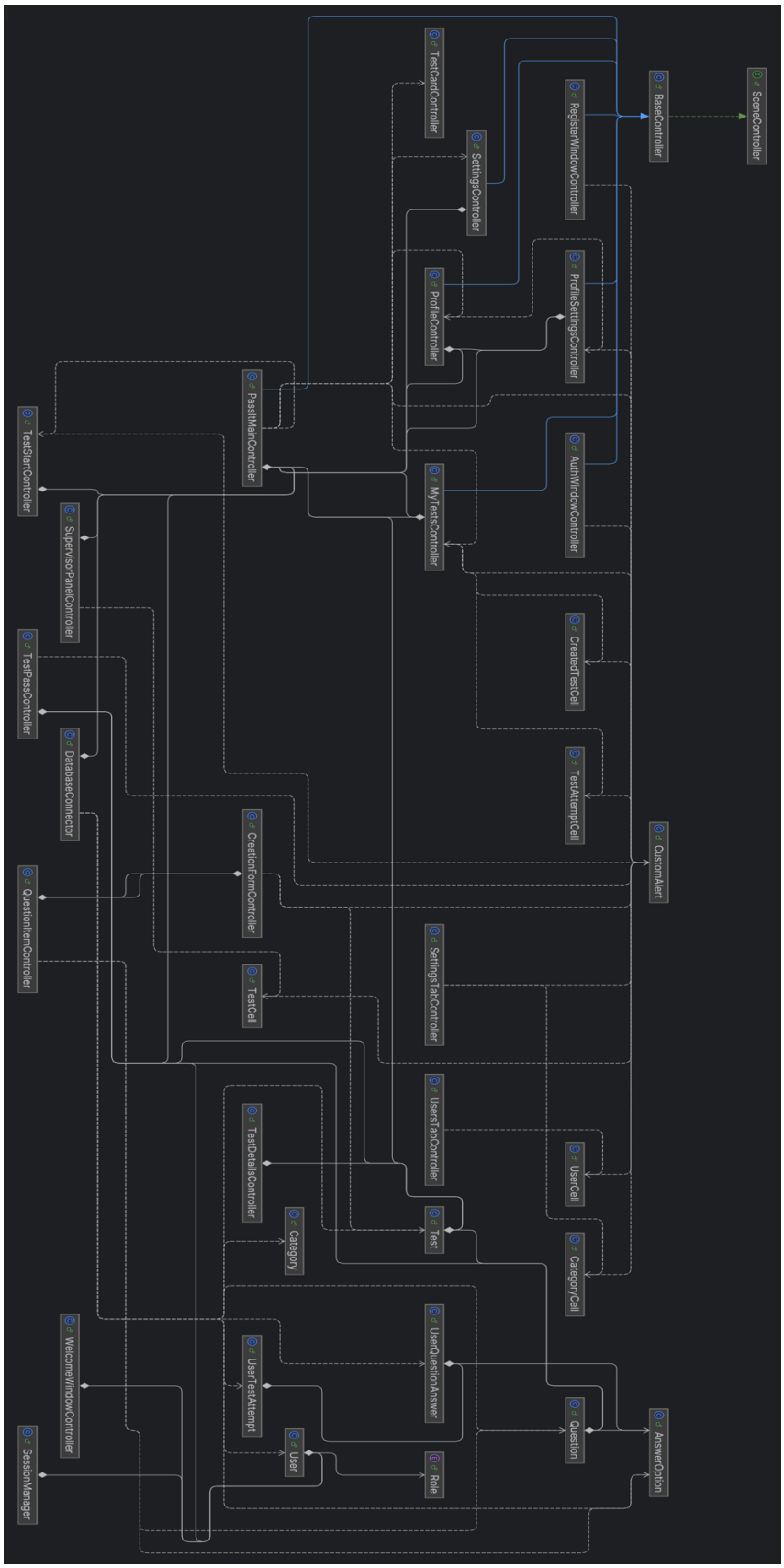

Рисунок А.1 – Діаграма класів згенерована в Intellij Idea Ultimate

ДОДАТОК Б. Тези для публікації на науково-технічну конференцію

УДК 004.415.2:004.514
Буняк В.С. – ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ТЕСТУВАННЯ ЗНАНЬ З МОДУЛЕМ ВІЗУАЛЬНОЇ АНАЛІТИКИ РЕЗУЛЬТАТІВ НА БАЗІ JAVAFX

Науковий керівник: д-р філос. І. Мудрик

Buniak V.
Ternopil Ivan Puluj National Technical University

DESIGN AND DEVELOPMENT OF A KNOWLEDGE TESTING SOFTWARE SYSTEM WITH A VISUAL ANALYTICS MODULE BASED ON JAVAFX

Supervisor: Ph.D. I. Mudryk

Ключові слова: JavaFX, тестування знань, візуальна аналітика, MySQL, MVC-архітектура, HikariCP, BCrypt.
Keywords: JavaFX, knowledge testing, visual analytics, MySQL, MVC architecture, HikariCP, BCrypt.

Широке впровадження інформаційних технологій в освітній процес та зростання потреби у якісному дистанційному оцінюванні знань зумовлюють актуальність розробки зручних і надійних програмних систем тестування. Метою роботи є проєктування та розробка десктопної системи PassIt — платформи для створення, проходження тестів та аналітики результатів на базі JavaFX 23.

Систему пропонується реалізувати згідно з архітектурним патерном MVC із використанням FXML для декларативного опису інтерфейсу та CSS для стилізації. Бізнес-логіку розподілено між модулями: Modules (моделі даних та сервіси), Controllers (обробники подій) та Views (кастомні компоненти відображення). Для взаємодії з базою даних MySQL використовується пул з'єднань HikariCP, що забезпечує ефективне управління ресурсами та стабільну роботу при паралельних запитах до бази даних.

Система передбачає реєстрацію та автентифікацію користувачів із хешуванням паролів за алгоритмом BCrypt, розмежування ролей (користувач, supervisor), а також повний цикл роботи з тестами: перегляд каталогу, пошук за назвою та категорією, проходження тестів із питаннями типу одиночного та множинного вибору, збереження результатів спроб у базі даних. Передбачено панель адміністратора для управління користувачами та категоріями тестів.

Ключовим складником розробки є модуль візуальної аналітики результатів, що реалізується засобами JavaFX Charts. Модуль передбачає побудову кругових діаграм розподілу результатів спроб, лінійних графіків динаміки успішності користувача в часі та стовпчастих діаграм порівняльної статистики по категоріях тестів. Дані для аналітики формуються на основі збережених спроб (UserTestAttempt) та відповідей на окремі питання (UserQuestionAnswer), що уможливлює деталізований аналіз прогалин у знаннях кожного користувача.

Для забезпечення прозорості роботи системи застосовується бібліотека Log4j/SLF4J, яка веде структуровані журнали подій із розмежуванням рівнів логування. Архітектура застосунку є

розширюваною — додавання нових типів питань або модулів аналітики не потребує суттєвих змін у наявному коді.

Результатом проєктування та розробки є програмний застосунок PassIt із зручним графічним інтерфейсом та надійним захистом даних. Розроблений продукт може використовуватись в освітніх закладах і корпоративному навчанні, слугуючи повноцінною альтернативою веб-орієнтованим платформам тестування знань.

ДОДАТОК В. Лістинг коду для реалізації алгоритму відстані Лівенштейна

```
package vt.passit.Modules;

public class AnswerEvaluator {
    public static double calculateSimilarity(String s1, String s2) {
        String str1 = s1.trim().toLowerCase();
        String str2 = s2.trim().toLowerCase();

        if (str1.equals(str2)) return 1.0;

        int distance = computeLevenshteinDistance(str1, str2);
        return 1.0 - ((double) distance / Math.max(str1.length(),
str2.length()));
    }

    private static int computeLevenshteinDistance(CharSequence lhs,
CharSequence rhs) {
        int len0 = lhs.length() + 1;
        int len1 = rhs.length() + 1;
        int[] cost = new int[len0];
        int[] newcost = new int[len0];
        for (int i = 0; i < len0; i++) cost[i] = i;
        for (int j = 1; j < len1; j++) {
            newcost[0] = j;
            for (int i = 1; i < len0; i++) {
                int match = (lhs.charAt(i - 1) == rhs.charAt(j - 1)) ? 0 :
1;
                int cost_replace = cost[i - 1] + match;
                int cost_insert = cost[i] + 1;
                int cost_delete = newcost[i - 1] + 1;
                newcost[i] = Math.min(Math.min(cost_insert, cost_delete),
cost_replace);
            }
            int[] swap = cost; cost = newcost; newcost = swap;
        }
        return cost[len0 - 1];
    }
}
```

ДОДАТОК Г. Лістинг коду для будування графіків

```
public static Map<String, Integer> getDailyActivityStats() {
    Map<String, Integer> stats = new LinkedHashMap<>();

    String sql = "SELECT DATE(end_time) as date, COUNT(*) as count " +
            "FROM TestAttempts " +
            "WHERE end_time >= DATE_SUB(CURDATE(), INTERVAL 7 DAY) " +
            "GROUP BY DATE(end_time) " +
            "ORDER BY date ASC";

    try (Connection conn = getInstance().getConnectionFromPool();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {

        while (rs.next()) {
            stats.put(rs.getString("date"), rs.getInt("count"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return stats;
}

public static Map<Integer, Integer> getScoreDistributionStats() {
    Map<Integer, Integer> stats = new TreeMap<>();
    for (int i = 0; i <= 90; i += 10) stats.put(i, 0);

    String sql = "SELECT FLOOR((score * 100.0 / max_score) / 10) * 10 as bucket,
COUNT(*) as count " +
            "FROM TestAttempts " +
            "WHERE max_score > 0 " +
            "GROUP BY bucket";

    try (Connection conn = getInstance().getConnectionFromPool();
         PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            int bucket = rs.getInt("bucket");
            if (bucket >= 100) bucket = 90;
            if (bucket >= 0) stats.put(bucket, rs.getInt("count"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return stats;
}
```