БАКАЛАВРСЬКА РОБОТА

БР. ІП - 11.00.00.000 ПЗ

Група ІП-21-4

Пилипів Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Пилипів Владислав Васильович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Побудова хмарно-базованої системи тестування знань

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Пилипів В.В.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Шекета Василь Іванович, д.т.н., професор

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Пилипіву Владиславу Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) **“ Побудова хмарно-базованої системи тестування знань”**

керівник проекту (роботи) Шекета В.І., проф., д.т.н.

затверджені наказом закладу вищої освіти від **“** 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області побудови хмарних інформаційних систем
2. Архітектурна та алгоритмічна реалізація хмарної системи тестування знань
3. Проектування Use Case діаграми системи тестування знань
4. Імплементація та програмна реалізація хмарно-базованої системи тестування знань
5. Представлення інтерфейсу користувача

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Багаторівнева архітектура хмарних обчислень (рис. 1.1)
2. Підключення мобільного пристрою до хмарної служби через VPN (рис. 1.2)
3. Еволюція хмарних обчислень (рис. 1.3)
4. Типи послуг, що пропонують хмарні обчислення (рис. 1.4)
5. Огляд RESTful архітектури (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **12 травня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області побудови хмарних інформаційних систем	07.05.2025	виконано
2	Архітектурна та алгоритмічна реалізація хмарної системи тестування знань	17.05.2025	виконано
3	Проектування Use Case діаграми системи тестування знань	27.05.2025	виконано
4	Імплементація та програмна реалізація хмарно-базованої системи тестування знань	02.06.2025	виконано
5	Представлення інтерфейсу користувача	06.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 76 сторінок, 19 рисунків, список використаних джерел із 39 найменуваннями, 1 додаток.

Мета роботи - розробити та реалізувати хмарну інформаційну систему тестування знань, яка забезпечує масштабованість, безпеку, зручність використання та підтримку ролей викладача і студента.

Об'єкт дослідження - хмарні інформаційні системи, призначені для автоматизованого тестування знань.

Предмет дослідження - принципи, архітектурні рішення та технології побудови хмарної системи тестування знань.

В першому розділі проаналізовано розвиток хмарних технологій, їх переваги порівняно з класичними системами, архітектурні підходи та протоколи взаємодії, що стали основою для проєктування системи

В другому розділі розглянуто функціональні можливості системи, спроєктовано її архітектуру, інтерфейси та вимоги до реалізації, що дозволяє забезпечити стабільну роботу та масштабованість

В третьому розділі описано технічну реалізацію системи, розробку інтерфейсів для викладача й студента, а також визначено можливості подальшого розвитку системи для забезпечення її гнучкості та функціонального зростання

Висновок: розроблено архітектурну модель, інтерфейси користувача для студентів і викладачів, базу даних і програмну логіку системи. Результатом роботи є створення прототипу ефективного освітнього інструменту для хмарного середовища.

КЛЮЧОВІ СЛОВА: ХМАРНІ ОБЧИСЛЕННЯ, СИСТЕМА ТЕСТУВАННЯ ЗНАНЬ, RESTFUL СЕРВІС, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, БАЗА ДАНИХ, ІНТЕРФЕЙС КОРИСТУВАЧА, ОСВІТНІ ТЕХНОЛОГІЇ.

ANNOTATION

The Bachelor's thesis contains 76 pages, 19 figures, a list of 39 references, and 1 appendix.

The goal of the work is to develop and implement a cloud-based knowledge testing information system that ensures scalability, security, usability, and supports both teacher and student roles.

The object of the research is cloud-based information systems designed for automated knowledge testing.

The subject of the research is the principles, architectural solutions, and technologies for building a cloud-based knowledge testing system.

The first chapter analyzes the development of cloud technologies, their advantages compared to classical systems, architectural approaches, and interaction protocols that formed the basis for system design.

The second chapter examines the system's functional capabilities, designs its architecture, interfaces, and implementation requirements, which ensures stable operation and scalability.

The third chapter describes the technical implementation of the system, the development of interfaces for teachers and students, and defines opportunities for the system's further development to ensure its flexibility and functional growth.

Conclusion: An architectural model, user interfaces for students and teachers, a database, and the system's software logic have been developed. The result of the work is the creation of a prototype for an effective educational tool in a cloud environment.

KEYWORDS: CLOUD COMPUTING, KNOWLEDGE TESTING SYSTEM, RESTFUL SERVICE, CLIENT-SERVER ARCHITECTURE, DATABASE, USER INTERFACE, EDUCATIONAL TECHNOLOGIES.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CSS - Cascading Style Sheets (Каскадні таблиці стилів)

GET, POST, PUT, DELETE - Основні методи HTTP-запитів в архітектурі REST

HTTPS - Hypertext Transfer Protocol Secure (Безпечний протокол передачі гіпертексту)

IaaS - Infrastructure as a Service (Інфраструктура як послуга)

MVC - Model-View-Controller (Модель-Представлення-Контролер - шаблон проектування)

Nginx - Популярний вебсервер/зворотний проксі

OMR - Optical Mark Recognition (Оптичне розпізнавання відповідей)

PaaS - Platform as a Service (Платформа як послуга)

REST - REpresentational State Transfer (Архітектурний стиль для розподілених систем)

SaaS - Software as a Service (Програмне забезпечення як послуга)

SLA - Service Level Agreement (Угода про рівень послуг)

SOA - Service-Oriented Architecture (Сервіс-орієнтована архітектура)

ВСТУП

У сучасному світі інформаційні технології стрімко проникають в усі сфери життя, і освіта — не виняток. Одним із головних напрямів цифровізації навчального процесу є впровадження хмарних рішень, які забезпечують доступність, масштабованість, ефективне управління ресурсами та підтримку дистанційного навчання. Використання хмарних технологій у сфері контролю знань дозволяє автоматизувати процес тестування, зберігати та аналізувати результати, оптимізувати роботу викладачів і підвищити якість оцінювання.

Особливої актуальності такі рішення набули в умовах глобального переходу до змішаного та дистанційного навчання, коли традиційні підходи до перевірки знань, зокрема паперові тести або системи оптичного розпізнавання відповідей, виявилися недостатньо гнучкими та не відповідали вимогам часу. У зв’язку з цим виникає потреба у створенні універсальних хмарних систем тестування знань, які можуть функціонувати у веб-середовищі, надаючи доступ до тестів з будь-якого пристрою незалежно від географічного розташування користувача.

У роботі здійснено всебічне дослідження процесів, пов’язаних з побудовою хмарних інформаційних систем, зосереджуючи увагу на специфіці систем тестування знань. Розглянуто переваги хмарних технологій, архітектуру RESTful сервісів, особливості побудови інтерфейсів користувача, проєктування баз даних та безпекових компонентів. Реалізовано прототип системи з підтримкою ролей студентів і викладачів, який демонструє здатність до масштабування, адаптивності та ефективного виконання освітніх завдань.

Таким чином, представлена розробка є актуальним прикладом застосування хмарних обчислень для автоматизації освітніх процесів і

формує основу для подальших досліджень та практичної реалізації хмарних сервісів у сфері освіти.

Актуальність дослідження

Актуальність дослідження зумовлена стрімким розвитком дистанційної освіти, а також потребою у створенні ефективних, масштабованих і безпечних інструментів для тестування знань. Хмарні обчислення відкривають нові можливості для реалізації освітніх сервісів з високою доступністю, автоматизованою обробкою результатів і гнучким адмініструванням. Тому розробка хмарної системи тестування знань є важливим і своєчасним завданням для забезпечення якісного контролю знань у сучасних умовах цифрової трансформації освіти.

Мета роботи - розробити та реалізувати хмарну інформаційну систему тестування знань, яка забезпечує масштабованість, безпеку, зручність використання та підтримку ролей викладача і студента.

Завдання дослідження

1. Провести аналіз предметної області хмарних інформаційних систем.
2. Дослідити сучасні архітектури та технології побудови хмарних застосунків.
3. Проєктувати архітектуру та інтерфейси майбутньої системи.
4. Реалізувати хмарну систему тестування знань.
5. Забезпечити безпеку, стабільність та масштабованість системи.

Об’єкт дослідження - хмарні інформаційні системи, призначені для автоматизованого тестування знань.

Предмет дослідження - принципи, архітектурні рішення та технології побудови хмарної системи тестування знань.

Методи дослідження

- Аналіз літературних джерел і існуючих систем;
- Проєктування програмного забезпечення;
- Методи об'єктно-орієнтованого моделювання (UML-діаграми);

- Прототипування та тестування вебзастосунків;

- Порівняльний аналіз архітектур і функціоналу;

- Метод системного аналізу та моделювання даних.

Наукова новизна

Запропоновано архітектурну модель хмарної системи тестування знань, що поєднує RESTful-сервіси, індивідуальний підхід до інтерфейсу користувача та сучасні принципи безпеки, що забезпечує гнучке масштабування та адаптацію до потреб конкретного освітнього середовища.

Практичне застосування

Розроблену систему можна впровадити у заклади вищої освіти, школи, навчальні центри, а також у процесах внутрішнього навчання та сертифікації працівників у компаніях.

Бакалаврська робота містить 76 сторінок, 19 рисунків, 3 розділи список використаних джерел із 39 найменуваннями, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПОБУДОВИ ХМАРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1. Опис пропонованої розробки хмарної системи тестування знань

В роботі пропонується архітектура та ключові функціональні можливості хмарної системи тестування знань, розробленої як комплексне програмне рішення для оптимізації процесів оцінювання рівня засвоєння навчального матеріалу серед студентів та забезпечення викладачів сучасними інструментами для створення, адміністрування та аналізу результатів тестових завдань. Система побудована на клієнт-серверній моделі, що забезпечує гнучкість доступу та централізоване управління даними.

Хмарна система тестування знань реалізована як розподілена система, що включає клієнтські застосунки та централізовану серверну частину. Розроблено нативні мобільні додатки для провідних мобільних платформ:

Застосунок для операційної системи Android, розроблений в інтегрованому середовищі розробки (IDE) Android Studio з використанням мови програмування Java.

Застосунок для операційної системи iOS (пристрої Apple), розроблений в IDE Xcode з використанням мов програмування Swift та Objective-C. Ці застосунки виконують функцію інтерфейсу користувача для проходження тестів студентами та взаємодіють із сервером через протокол HTTP для надсилання відповідей та отримання тестових даних.

Серверна частина. Розміщена на віртуальному приватному сервері (VPS) провайдера Digital Ocean. Серверна інфраструктура включає:

- Вебсервер Nginx, який функціонує як зворотний проксі та обробляє вхідні HTTP-запити від клієнтських застосунків.

- Скрипти серверної логіки, написані мовою програмування PHP, які відповідають за обробку запитів, взаємодію з базою даних, реалізацію бізнес-логіки (створення тестів, оцінювання, збір статистики).

- Система управління базами даних (СУБД) MySQL, що використовується для централізованого зберігання даних про користувачів (студентів та викладачів), тестові завдання, варіанти відповідей, результати проходження тестів та статистику. Взаємодія PHP-скриптів з базою даних здійснюється через вебсервер Nginx.

Розглянемо пропоновані функціональні можливості.

Система надає різні функціональні можливості для двох основних категорій користувачів:

1. Для студентів:

- Доступ до призначених тестів через мобільні застосунки.

- Проходження тестів в інтерактивному форматі.

- Надсилання відповідей на сервер для автоматизованого оцінювання.

2. Для викладачів:

- Доступ до вебсерверної частини системи через веб-інтерфейс (зазвичай, інтегрований або окремий додаток, що взаємодіє з тим же бекендом).

- Створення нових тестових завдань, включаючи визначення питань, варіантів відповідей та правильних варіантів.

- Редагування існуючих тестів та їх параметрів.

- Призначення тестів певним групам студентів або індивідуальним користувачам.

- Перегляд результатів проходження тестів студентами.

Отримання детальної статистики за результатами тестів, що відображається у зручному для аналізу форматі (наприклад, відсоток правильних відповідей, розподіл балів, аналіз питань, що викликали найбільші труднощі).

Для мінімізації можливостей несанкціонованого доступу до інформації та забезпечення академічної доброчесності під час тестування, в системі реалізовано комплекс механізмів протидії недоброчесності:

- Випадкове перемішування варіантів відповідей: Для кожного студента та кожного проходження тесту порядок варіантів відповідей до питань є унікальним, що ускладнює обмін відповідями між студентами.

- Заборона створення скріншотів: Функціональність мобільних застосунків блокує можливість створення знімків екрана під час проходження тесту.

- Заборона запису екрана: Аналогічно до скріншотів, система обмежує функціональність запису відео з екрана пристрою під час активного тестування.

- Контроль виходу з додатку: Спроба виходу з мобільного застосунку під час проходження тесту може фіксуватися системою або призводити до автоматичного завершення тестування, сигналізуючи про потенційну спробу використання сторонніх ресурсів.

Система розроблена з використанням сучасних технологій та мов програмування, що відповідають специфіці її компонентів:

- Мобільні додатки: Java (Android), Swift та Objective-C (iOS).

- Серверна логіка: PHP.

- База даних: MySQL.

- Вебсервер: Nginx.

- Протокол комунікації: HTTP-запити для взаємодії між клієнтськими застосунками та сервером.

- Середовища розробки: Android Studio та Xcode.

Хмарна система тестування знань є багатокомпонентним програмним комплексом, що ефективно вирішує завдання автоматизованого оцінювання знань. Побудована на надійній клієнт-серверній архітектурі з використанням сучасних технологій, система надає гнучкі інструменти для викладачів та

зручний інтерфейс для студентів, а реалізовані механізми протидії недоброчесності сприяють забезпеченню об'єктивності та достовірності результатів оцінювання.

1.2. Опис підходів від класичних розподілених систем до хмарних обчислень

Історичний розвиток інформаційних технологій демонструє постійне прагнення до підвищення ефективності використання обчислювальних ресурсів, забезпечення надійності сервісів та оптимізації витрат. Класичні розподілені обчислювальні системи на певному етапі відіграли вирішальну роль у досягненні цих цілей, дозволяючи розподіляти обчислювальне навантаження та дані між множиною взаємодіючих комп'ютерів, що забезпечувало підвищену доступність сервісів та потенціал до горизонтального масштабування.

Ідея використання обчислювальних ресурсів як комунальної послуги, аналогічно до постачання води чи електроенергії, була концептуально сформульована ще у 1960 році Джоном Маккарті під час його виступу в MIT. Ця візіонерська концепція передбачала модель, за якою користувачі могли б споживати обчислювальні потужності за потребою, сплачуючи лише за фактичне використання, без необхідності володіти та обслуговувати власну інфраструктуру.

Практичні кроки до реалізації цієї парадигми почали з'являтися пізніше. Важливим етапом стало розповсюдження програмного забезпечення як сервісу (SaaS), піонером якого стала компанія Salesforce, що з 1999 року почала надавати свої додатки кінцевим користувачам через веб-інтерфейс, усуваючи необхідність інсталяції та підтримки програмного забезпечення на локальних пристроях.

Подальший розвиток розподілених систем призвів до появи інфраструктури як сервісу (IaaS), що стало основою для сучасної хмарної обчислювальної парадигми. Заснування Amazon Web Services (AWS) близько 2002 року стало переломним моментом, після чого на ринок вийшли інші великі гравці, такі як Google Cloud Platform, Microsoft Azure, HP та інші. Ця трансформація докорінно змінила ландшафт IT-індустрії, запропонувавши нові моделі споживання та управління обчислювальними ресурсами.

1.2.1. Переваги хмарних обчислень порівняно з класичними розподіленими системами

Основні переваги хмарних обчислень, що відрізняють їх від традиційних підходів до розподілених систем, полягають у моделі споживання ресурсів, масштабованості та економічній ефективності.

У традиційних системах організації змушені здійснювати значні капітальні інвестиції (CapEx) у придбання та розгортання фізичного обладнання (серверів, мережевого обладнання, систем зберігання даних), часто з надлишковим резервуванням для обробки пікових навантажень. Хмарні обчислення переводять цю модель у операційні витрати (OpEx), дозволяючи користувачам сплачувати лише за фактично спожиті ресурси (час використання CPU, обсяг даних, пропускна здатність мережі). Це значно знижує початкові бар'єри для входу та оптимізує бюджетування.

Хмарні платформи надають можливість динамічно масштабувати обчислювальні ресурси вгору або вниз відповідно до поточного попиту. Ця еластичність дозволяє ефективно справлятися з непередбачуваними або періодичними піковими навантаженнями, характерними для багатьох додатків та сервісів, без необхідності утримувати постійно працюючу надмірну інфраструктуру, як це було необхідно в класичних системах для гарантування продуктивності в години пік.

Постачальники хмарних послуг агрегують значні пули обчислювальних ресурсів (сервери, сховища, мережі) та динамічно виділяють їх численним користувачам. Така мультиорендна (multi-tenant) модель забезпечує більш високий коефіцієнт використання обладнання на стороні провайдера, що, своєю чергою, дозволяє пропонувати послуги за нижчою вартістю.

Для доступу до хмарних ресурсів користувачу зазвичай достатньо веббраузера та підключення до Інтернету. Це значно спрощує розгортання, управління та доступ до складних обчислювальних середовищ навіть для невеликих команд розробників або окремих фрілансерів, яким раніше була недоступна власна високопродуктивна інфраструктура.

1.2.2. Сучасне значення та застосування хмарних обчислень

Широке впровадження хмарних обчислень свідчить про їх переконливі переваги. Більшість сучасних масштабних веб-сервісів, включаючи соціальні мережі, стрімінгові платформи та корпоративні додатки, базуються на хмарній інфраструктурі. Це підтверджує здатність хмарних платформ задовольняти потреби широкого кола клієнтів, від великих корпорацій до стартапів та індивідуальних розробників.

Отже, перехід від класичних розподілених обчислювальних систем до парадигми хмарних обчислень є значним кроком у розвитку IT-індустрії. Ключовими факторами цього переходу стали масштабованість, еластичність, економічна ефективність моделі оплати за фактом використання та спрощення доступу до обчислювальних ресурсів. Хмарні обчислення не лише забезпечили нові можливості для бізнесу та розробників, але й демократизували доступ до високопродуктивної інфраструктури, зробивши її доступною та керованою для широкого кола користувачів, що робить використання альтернативних, не-хмарних рішень для багатьох сучасних завдань менш ефективним або навіть застарілим.

1.3. Архітектура хмарних сервісів

Архітектура сучасних хмарних обчислювальних систем проєктується з урахуванням ключових принципів, що забезпечують ефективне, масштабоване та економічно вигідне надання обчислювальних ресурсів та сервісів кінцевим користувачам. Відмінність хмарної архітектури від традиційних інфраструктурних рішень полягає у фундаментальній зміні моделі споживання та управління ресурсами.

Одним із наріжних каменів хмарної архітектури є принцип оплати за фактом використання (Pay-as-you-go). На відміну від класичної парадигми, де користувачі або організації здійснювали значні капітальні інвестиції (CapEx) у придбання фізичного обладнання, хмарна модель базується на операційних витратах (OpEx). Користувачі отримують доступ до пулу ресурсів та сплачують лише за обсяг фактично спожитих ресурсів (наприклад, час процесорного часу, обсяг зберігання даних, пропускна здатність мережі) протягом певного періоду. Ця модель значно знижує початкові витрати та ризики, пов'язані з надмірним інвестуванням у власну інфраструктуру.

Ключовою характеристикою, що реалізується архітектурними засобами хмарних платформ, є еластичність (Elasticity). Вона передбачає можливість динамічного масштабування обчислювальних ресурсів — як у бік збільшення (масштабування вгору), так і в бік зменшення (масштабування вниз) — відповідно до поточного попиту. Ця функціональність є критично важливою для додатків з варіативним навантаженням, зокрема тих, що мають чітко виражені "пікові години" або періоди підвищеного попиту. У класичних системах забезпечення необхідної продуктивності під час пікових навантажень вимагало придбання значного обсягу надлишкових ресурсів, які простоювали у періоди низької активності. Архітектура хмарних сервісів дозволяє уникнути цієї неефективності, автоматично або за запитом

користувача виділяючи додаткові ресурси саме тоді, коли вони потрібні, та звільняючи їх, коли навантаження спадає.

Принцип об'єднання ресурсів (Resource Pooling) є ще одним фундаментальним аспектом хмарної архітектури. Постачальники хмарних послуг агрегують великі обсяги фізичних обчислювальних ресурсів (сервери, системи зберігання, мережеве обладнання) у великих центрах обробки даних. Завдяки механізмам віртуалізації та управління ресурсами, ці фізичні ресурси абстрагуються та надаються користувачам як логічні пули, з яких динамічно виділяються необхідні обсяги ресурсів. Така модель мультиорендності (multi-tenancy), коли множина користувачів спільно використовують одну фізичну інфраструктуру (хоча й логічно ізольовано), забезпечує високий рівень використання ресурсів на рівні провайдера та дозволяє оптимізувати витрати для кінцевих споживачів.

Доступність до хмарних ресурсів, що реалізується архітектурно через веб-орієнтовані інтерфейси та API, значно спрощується. Для доступу до обчислювальних потужностей, сховищ даних чи програмних сервісів користувачеві здебільшого потрібен лише веббраузер та підключення до мережі Інтернет. Така універсальна модель доступу, що є наслідком сервісно-орієнтованого підходу в архітектурі хмари, робить потужні обчислювальні можливості доступними для широкого кола користувачів, включаючи малий та середній бізнес, стартапи та індивідуальних розробників, для яких розгортання та підтримка власної масштабної інфраструктури було б економічно недоцільним або технічно складним.

Масове впровадження хмарних сервісів у сучасному цифровому світі, зокрема використання їх більшістю соціальних медіа платформ та великих вебсайтів, слугує переконливим емпіричним доказом ефективності та зрілості хмарної архітектури. Основні чинники, що зумовили таку широкомасштабну міграцію до хмарних рішень, безпосередньо пов'язані з архітектурними перевагами: високою масштабованістю, гнучкістю у споживанні ресурсів,

зниженням операційних витрат та суттєвим спрощенням процесів розгортання та управління IT-інфраструктурою порівняно з традиційними підходами. Таким чином, архітектура хмарних сервісів стала основою для нової парадигми надання обчислювальних послуг, що значно перевершує класичні моделі за багатьма ключовими показниками.

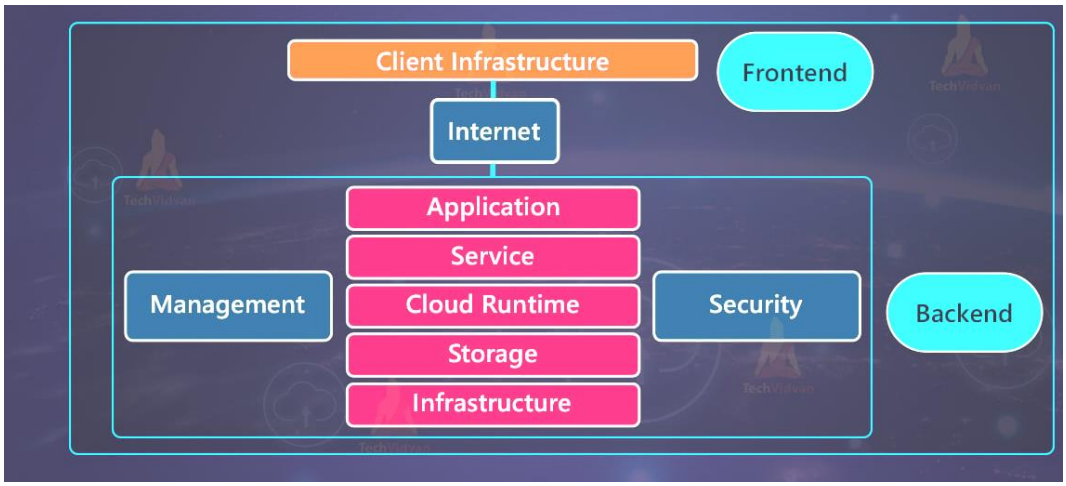

Рисунок 1.1 - Багаторівнева архітектура хмарних обчислень

На рисунку 1.1 представлена спрощена багаторівнева архітектура хмарних обчислень, яка ілюструє взаємодію між користувачем та хмарними сервісами, а також ключові компоненти, що складають хмарну платформу.

Основні елементи архітектури:

- Client Infrastructure (Клієнтська інфраструктура) - цей блок представляє обладнання та програмне забезпечення на стороні користувача (наприклад, комп'ютери, мобільні пристрої, веббраузери), яке використовується для доступу до хмарних сервісів.

- Internet (Інтернет) виступає як мережеве середовище, що забезпечує зв'язок та передачу даних між клієнтською інфраструктурою та хмарною платформою.

- Frontend (Фронтенд) - цей термін вказує на частину системи, з якою безпосередньо взаємодіє користувач. У контексті хмари це може включати інтерфейси користувача додатків, веб-портали управління хмарними

ресурсами тощо. На рисунку він асоціюється з клієнтською стороною та верхніми рівнями сервісів.

- Backend (Бекенд) - представляє собою серверну частину хмарної платформи, яка включає всю базову інфраструктуру, сервіси, дані та логіку обробки, недоступну безпосередньо користувачеві. На рисунку він охоплює основні шари хмарної архітектури.

- Центральний блок хмарної платформи (Backend) - блок складається з декількох вертикально розташованих шарів:

- Application - рівень прикладних програм (SaaS - Software as a Service), які надаються користувачам як готові сервіси (наприклад, CRM-системи, офісні пакети).

- Service - рівень платформних сервісів (PaaS - Platform as a Service), що надає середовище для розробки, тестування та розгортання додатків (наприклад, бази даних як сервіс, сервіси черг повідомлень).

- Cloud Runtime - рівень, що забезпечує виконання додатків та сервісів, часто включаючи віртуалізацію та контейнеризацію.

- Storage - рівень сервісів зберігання даних (наприклад, об'єктне сховище, блокове сховище, файлові системи).

- Infrastructure - найнижчий рівень (IaaS - Infrastructure as a Service), що включає базові обчислювальні ресурси (сервери, мережеве обладнання) та їх віртуалізацію.

- Supporting Layers/Functions (Допоміжні рівні/функції):

- Management - компонент, що відповідає за моніторинг, розподіл ресурсів, балансування навантаження, автоматизацію та оркестрацію в межах хмарної платформи. Дозволяє провайдеру та користувачам управляти хмарною інфраструктурою та сервісами.

- Security - компонент, що забезпечує захист даних, управління доступом, ідентифікацією та інші аспекти кібербезпеки на всіх рівнях хмарної архітектури.

Таким чином, рисунок демонструє, як клієнти через Інтернет отримують доступ до багатошарової хмарної платформи, яка надає різні типи сервісів (від базової інфраструктури до готових додатків), підтримується системами управління та забезпечується механізмами безпеки. Фронтенд є видимою для користувача частиною, тоді як бекенд включає основні обчислювальні та програмні компоненти хмари.

1.4. Взаємодія клієнт-сервер у хмарних застосунках з використанням RESTful сервісів

Взаємодія між клієнтським застосунком (наприклад, мобільним додатком) та віддаленими хмарними сервісами, що реалізовані за допомогою архітектури REST (Representational State Transfer), є типовою парадигмою для розробки сучасних розподілених систем. У такій архітектурі значна частина логіки обробки даних та обчислень виконується на серверній стороні, тоді як клієнт переважно відповідає за представлення інтерфейсу користувача (User Interface, UI) та збір вхідних даних.

Процес обробки вхідних даних користувача та динамічного відображення результатів на мобільному пристрої відбувається за наступною схемою:

а) збір вхідних даних на клієнті. Клієнтський застосунок, інтерфейс якого розміщений та виконується на фізичному пристрої користувача, фіксує дії та вхідні дані, введені користувачем через елементи UI (наприклад, натискання кнопок, введення тексту, вибір опцій). Система може використовувати внутрішні механізми або "агенти" для моніторингу цих подій введення.

б) передача даних на віддалений сервер. Зібрані вхідні дані або ініційовані користувачем події формуються у запити, які надсилаються на віддалений серверний компонент, розміщений у хмарному середовищі. Як

правило, для цього використовуються стандартизовані веб-протоколи, такі як HTTP, у форматі RESTful запитів (наприклад, GET, POST запити до певних ресурсів). Така архітектура дозволяє безпечно та надійно передавати дані для подальшої обробки.

в) обробка даних на хмарному сервері (Backend Processing). Віддалений хмарний сервер приймає та обробляє вхідні запити. Цей процес включає виконання необхідних алгоритмів та бізнес-логіки. Наприклад, першим етапом обробки вхідних даних від користувача може бути автентифікація – перевірка дійсності наданих облікових даних (логіна/пароля) для доступу до запитуваних ресурсів або сервісів (наприклад, тесту). Після успішної автентифікації та валідації, сервер виконує подальші обчислення:

- Вибірка або генерація необхідних даних з бази даних (наприклад, запитань та варіантів відповідей для конкретного тесту).

- Виконання складної логіки, яка не може або не повинна виконуватися на клієнті (наприклад, оцінювання відповідей, обчислення статистики).

г) важливою перевагою використання хмарного сервера є можливість його динамічного масштабування (збільшення або зменшення ресурсів, таких як обчислювальна потужність, обсяг пам'яті) відповідно до поточного рівня сукупного попиту від усіх підключених користувачів. Це забезпечує стабільну продуктивність системи навіть при значних коливаннях навантаження.

д) після завершення обробки сервер формує відповідь, яка може містити запитувані дані (наприклад, вміст екзамену), результати обчислень або інструкції для клієнтського застосунку. Ця відповідь надсилається назад на мобільний пристрій користувача через Інтернет, знову ж таки, за допомогою протоколів, сумісних з RESTful архітектурою (наприклад, у форматі JSON або XML).

е) Клієнтський мобільний застосунок отримує відповідь від сервера. Оскільки UI додатка зберігається локально, застосунок використовує отримані дані (наприклад, список запитань тесту, текст запитань, варіанти відповідей) для динамічного оновлення свого інтерфейсу користувача. Це дозволяє відобразити актуальний вміст екзамену або іншу релевантну інформацію, створену чи оброблену на сервері, представляючи її користувачеві у візуально зрозумілому форматі на сторінці тесту або іншому відповідному екрані додатка.

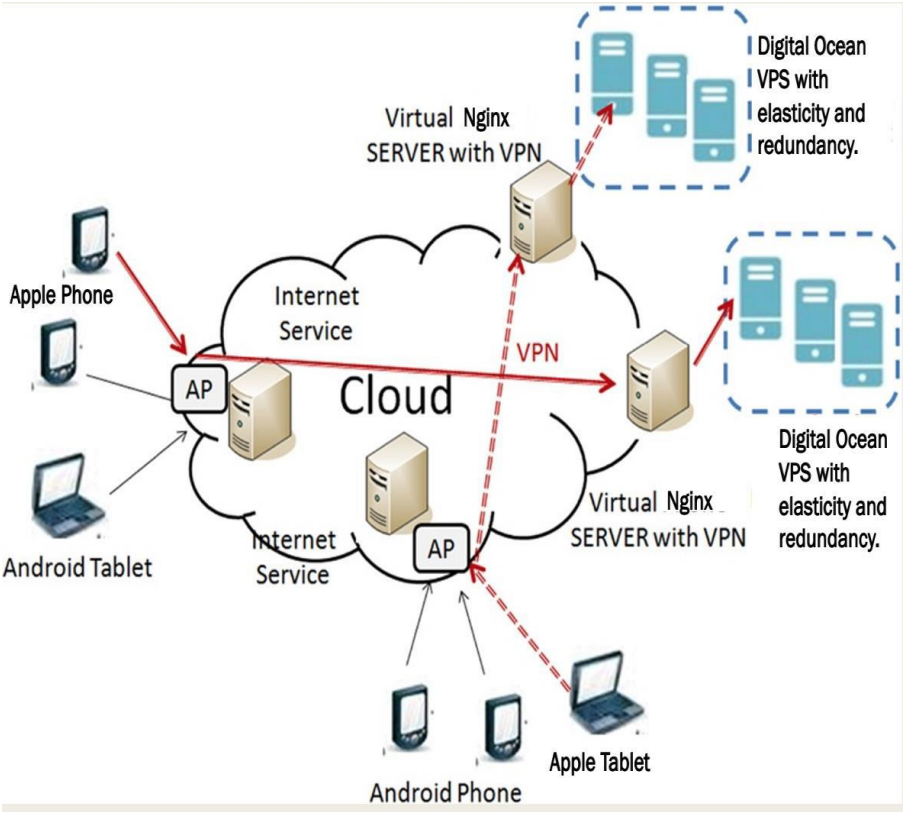

Рисунок 1.2 - Підключення мобільного пристрою до хмарної служби через VPN

Схематичне зображення такої взаємодії, коли мобільні пристрої підключаються до віддаленого хмарного сервера (наприклад, розміщеного на Digital Ocean) для виконання обчислень, пов'язаних з обробкою даних для екзаменів, може бути представлене на рисунку 1.2. Цей рисунок ілюструє потік даних від клієнта до хмари для обробки та зворотну передачу

результатів для відображення на пристрої, підкреслюючи централізований характер обчислень у хмарному середовищі.

1.5. Огляд реалізації хмарних технологій

Уявіть собі звичайний робочий день на вашій робочій станції або в офісі, скільки ресурсів вашого комп'ютера, як ви очікуєте, використовується? Як зазначив Марстон у статті "Хмарні обчислення з точки зору бізнесу", середнє значення для більшості користувачів становить близько 10% процесора, 60% пам'яті та 20% пропускної здатності під час пікових годин. Незалежно від того, скільки ресурсів потрібно користувачеві або організації, вони платять 100% вартості цих ресурсів заздалегідь, коли вони купують комп'ютер. Тепер уявіть собі велику корпорацію з багатьма комп'ютерами, які використовуються неефективно. Якщо ці комп'ютери можна було б використовувати на повну потужність або компанія інвестувала б менше ресурсів, вони могли б оптимізувати свою продуктивність.

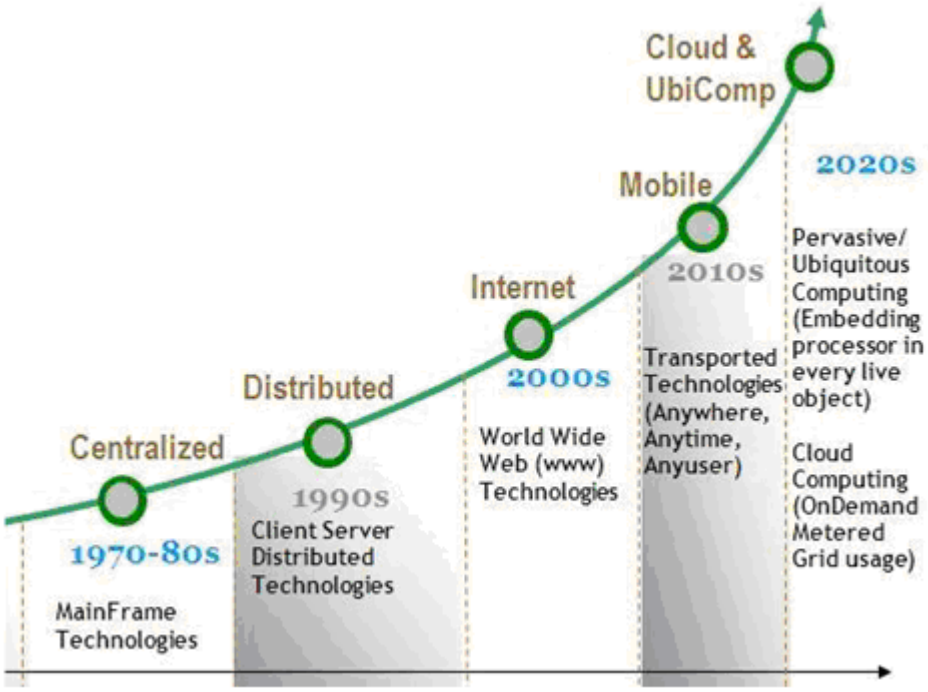

Рисунок 1.3 – Еволюція хмарних обчислень

Однак це не так, оскільки багато продуктів повинні бути гнучкими та масштабованими за попитом. Неможливо купувати та продавати комп'ютери на льоту, тут на допомогу приходять хмарні обчислення. Вебсервери, сервери додатків та сервери баз даних зазвичай використовуються з мінімальною навантаженням, тоді як вони могли б використовувати спільні ресурси.

При використанні хмарної обчислювальної платформи вашій компанії не потрібно інвестувати великі кошти у вигляді капітальних витрат на купівлю обладнання, а просто укладати контракт з постачальником послуг за моделлю "плати за користування" або "плати за те, що використовуєш". Ця модель означає, що компанії можуть перейти від моделі капітальних витрат (CapEx) до моделі операційних витрат (OpEx) лише для задоволення обчислювальних потреб.

Клієнтські комп'ютери (кінцевий користувач може взаємодіяти з хмарою за допомогою клієнтських комп'ютерів), розподілені комп'ютери (сервери розподілені в різних місцях, але працюють так, ніби вони працюють разом), та центри обробки даних (компіляція серверів).

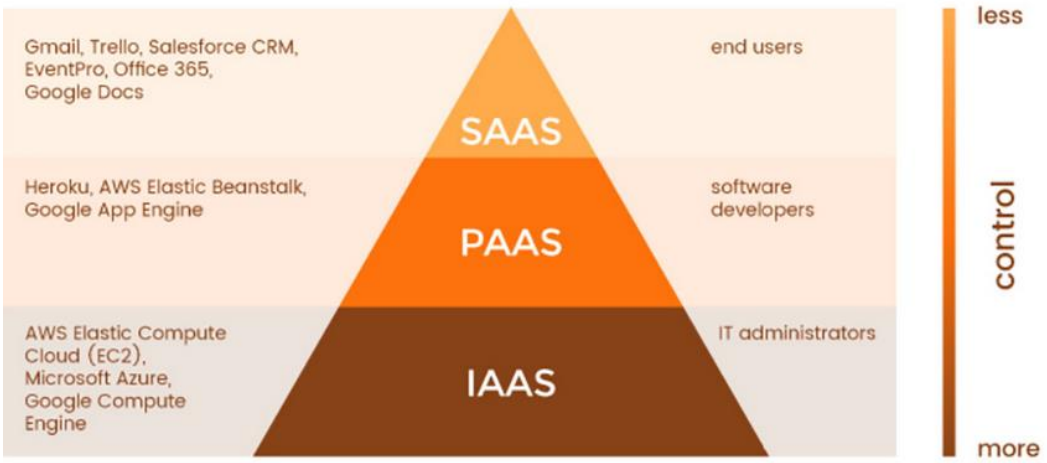

Рисунок 1.4 – Типи послуг, що пропонують хмарні обчислення

Існують три основні послуги, які пропонують хмарні обчислення: програмне забезпечення як послуга (SaaS), платформа як послуга (PaaS) та інфраструктура як послуга (IaaS). У публічній хмарній системі доступ

додається будь-кому з інтернет-з'єднанням, і ця система може існувати будь-де у світі. Цей тип послуги має проблему цілісності даних, яка частково виникає через нормативні вимоги. Деякі корпорації, наприклад, ті, що базуються в Сполучених Штатах, не можуть зберігати дані споживачів в інших країнах. Ця проблема настільки руйнівна для деяких, що вони обирають приватні хмарні рішення. Приватна хмарна система надає послуги одній ентитеті. Ця ентитет може бути урядом, корпорацією або будь-якою іншою особою, готовою платити за ресурси. Хоча ця послуга доступна для будь-кого, вона дорога і зазвичай доступна лише для великих підприємств та урядів. Нарешті, рішення для спільноти хмар можна досить точно описати як проміжний варіант між приватними та публічними хмарними послугами. Однак системи спільноти хмар мають переваги та недоліки обох реалізацій. У цій реалізації ентитети зі спільними інтересами можуть об'єднувати свої ресурси для створення так званої гібридної хмари.

1.6. Опис протоколу RESTful

REST (REpresentational State Transfer) — це архітектурний стиль, заснований на передачі представлень ресурсів від сервера до клієнта. Це стиль, який лежить в основі вебу в цілому, і був використаний як більш простий метод, ніж SOAP/WSDL для реалізації вебсервісів. RESTful вебсервіс ідентифікується своїм URI (Універсальний ідентифікатор ресурсів) і спілкується за допомогою протоколу HTML. Він реагує на запити HTML GET, PUT, POST і DELETE та повертає представлення ресурсу клієнту. Простота, POST означає створення, GET означає читання, PUT означає оновлення, а DELETE означає видалення. RESTful сервіси мають менші витрати, ніж так звані "великі вебсервіси", і використовуються багатьма організаціями для реалізації сервісних систем, які не залежать від зовнішніх послуг.

REST — це гнучкий і потужний метод спілкування в розподілених системах. Протокол HTTP надає прямий доступ без стану до веборегацій. Як зазначено в статті "Керування авторизацією з RESTful XML", "Причина використання REST як методу взаємодії полягає у підтримці операцій, пов'язаних з діями HTTP, відсутності складного управління станами та варіабельності щодо характеристик ресурсів". REST може працювати з багатьма типами ресурсів, навіть якщо вони представлені в спеціальних форматах або JSON.

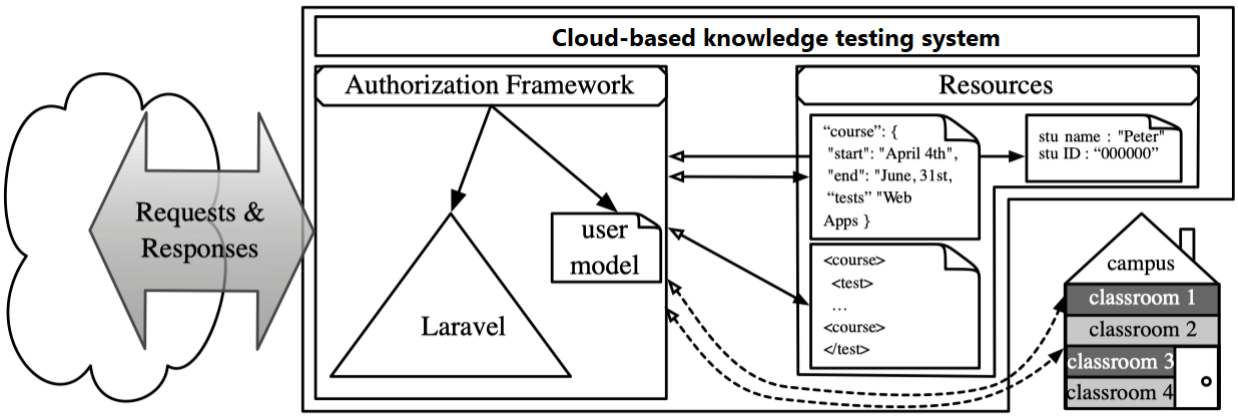

Рисунок 1.5 – Огляд RESTful архітектури

На рисунку 1.5 показано, як хмарна система тестування знань реалізує протоколи RESTful. Фреймворк Laravel обробляє запити, які є ресурсами всередині проекту. Ці ресурси створюються в файлах, таких як курс, клас та HTML-файли. Поза межами рисунка ми можемо побачити кампус з кількома аудиторіями, які використовують систему.

Ось чому REST-сервіси використовуються в застосунку тестування знань, де дані користувача, що передаються, є сумішшю рядкових (string) та цілочислових (integer) значень, а також форматованих типів даних JSON. Хоча широке впровадження хмарних обчислень, здається, вигідне всім сторонам, це може бути не так, коли мова йде про безпеку [8]. Безпека є комплексною, вона стосується не тільки використання системи провайдером,

але й вами та будь-яким проміжним користувачем чи застосунком. При оцінці безпеки системи ми повинні застосовувати наскрізний (end-to-end) підхід. Від кінцевого користувача до центру обробки даних можуть діяти численні регуляторні норми та виникати проблеми безпеки, які впливають на цілісність даних. Якщо дані, що надсилаються користувачем до центру обробки даних або центром обробки даних до користувача, можуть бути перехоплені та прочитані, цілісність наших даних буде скомпрометована. Єдиний спосіб забезпечити максимальну цілісність даних, здається, полягає в захисті кінцевих точок, якими є користувач і центр обробки даних [8]. Крім того, канал передачі даних, яким пересилається інформація, має бути захищеним у будь-який час. Ці дані для передачі також повинні бути в зашифрованому стані, щоб уникнути атак типу "людина посередині" (man in the middle), коли якась сутність перехоплює повідомлення і може використовувати його на свій розсуд. Завдяки захищеному та зашифрованому каналу стороннє програмне забезпечення не може скомпрометувати цілісність даних хмарної системи. Більшість хмарних систем використовують ключі шифрування при шифруванні каналу. Іншим методом є використання ланцюжка відкритих ключів, наприклад, центру сертифікації (Certificate Authority, CA), для перевірки цього відкритого ланцюжка; при цьому методі обидві сторони повинні узгодити протоколи та ключі, які мають бути перевірені як дійсні.

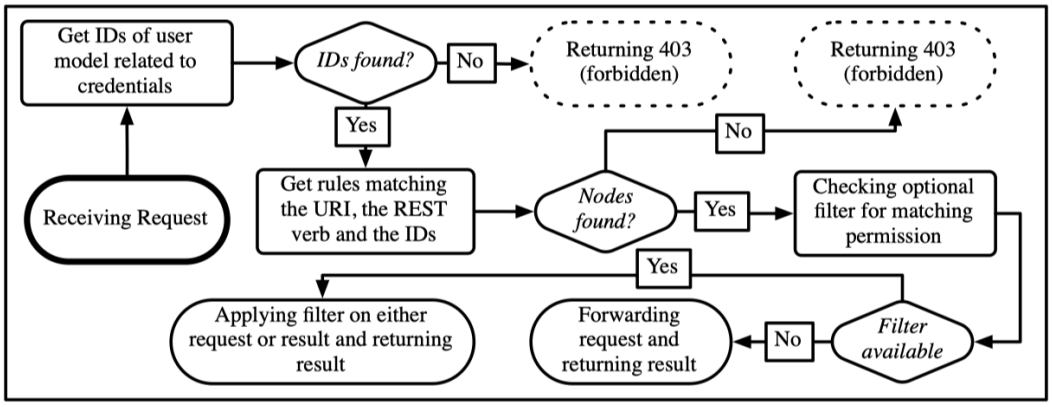

Рисунок 1.6 - Робочий процес автентифікації

На рисунку 1.6 можна побачити, як фреймворк автентифікації Laravel обробляє запити. Цей фреймворк містить численні рівні перевірки на помилки при отриманні запиту від мобільного пристрою.

Оскільки віртуальне хмарне середовище, як правило, працює в публічній хмарі, важливо, щоб дані користувачів передавалися через захищений канал, і всі кінцеві точки також були захищені. Крім цього, важливо вибрати надійного постачальника послуг IaaS при розгортанні віртуального хмарного середовища. У випадку системи Bubble-In хмарний сервер розміщений на Digital Ocean VPS, який є невеликим, але також надійним постачальником хмарних послуг. Digital Ocean пропонує сучасні заходи безпеки від наземних заходів безпеки, резервні генератори, повне відеоспостереження, контроль доступу на основі ролей для запобігання необмеженого доступу до будь-якого екземпляра. Екземпляр Bubble-In, розміщений на Digital Ocean, також пропонує моніторинг безпеки, а також опції резервного копіювання та відновлення екземплярів. Для забезпечення більш високого рівня безпеки використовуються розширені методи аутентифікації та шифрування для забезпечення та встановлення безпечного каналу зв'язку. Зберігання ключів SSH для з'єднання забезпечує це шифрування та аутентифікацію до хмарного сервера. Для більш високого рівня безпеки створюється кілька користувацьких облікових записів з мінімально необхідними дозволами для адміністративних та розробницьких облікових записів.

1.7. Заключні положення щодо архітектури, переваг та безпеки хмарних обчислень

Хмарні обчислення визначено як ключовий каталізатор трансформаційних процесів, що чинять суттєвий вплив на соціальну, професійну та політичну сфери життя суспільства. Переваги хмарних

обчислень, зокрема економічна ефективність, еластичність та доступність, значно перевершують можливості традиційних парадигм розподілених обчислювальних систем та мережевих сервісів. Аналіз вартості розгортання та експлуатації сервісів у хмарному середовищі демонструє суттєве зниження початкових (CapEx) та операційних (OpEx) витрат порівняно з необхідними інвестиціями у власну інфраструктуру. Це сприяє спрощенню процесу виведення на ринок нових програмних продуктів та ідей, роблячи його більш економічно обґрунтованим, швидким та ефективним.

Незважаючи на наявність певних викликів у сфері безпеки даних та інфраструктури в умовах розподіленого хмарного середовища, застосування науково обґрунтованих та практично реалізованих заходів безпеки дозволяє ефективно мінімізувати потенційні ризики. Реалізація захищених каналів комунікації, зокрема через використання протоколів HTTPS та механізмів аутентифікації в рамках архітектури RESTful сервісів, є критично важливою. Посилена увага до безпеки зростаючої кількості кінцевих точок доступу, що є характерною рисою хмарних систем, дозволяє підтримувати необхідний рівень захисту інформаційних активів.

Загалом, сукупність переваг, які надають хмарні обчислення (масштабованість, гнучкість, економічна ефективність), значно переважає їхні потенційні недоліки, що робить їх логічним та стратегічно обґрунтованим вибором для розвитку інформаційних технологій як у промисловості, так і в освітній сфері на найближчу перспективу. У специфічних реалізаціях, таких як хмарні системи тестування знань (наприклад, розглянута система), гнучкість, що забезпечується RESTful сервісами та динамічними хмарними ресурсами, є критичною для забезпечення можливості швидкого масштабування відповідно до потреб користувачів. У контексті розробки подібних систем, аналогічні питання безпеки, які розглядалися в даній роботі (включаючи механізми протидії недоброчесності), можуть бути успішно вирішені шляхом впровадження

багаторівневих заходів безпеки (на рівнях застосунку та сервера) та механізмів строгої аутентифікації користувачів та компонентів системи. Комплексний підхід до забезпечення безпеки та цілісності системи, що включає ретельне проєктування та імплементацію відповідних механізмів, дозволяє розгортати та експлуатувати такі рішення навіть у публічних хмарних середовищах, мінімізуючи ризики несанкціонованого доступу або компрометації даних, спричинених потенційною недбалістю або зовнішніми загрозами.

Висновки до розділу

В результаті аналізу предметної області було визначено ключові аспекти, що формують основу для розробки ефективної хмарної системи тестування знань. Розглянуто особливості переходу від класичних розподілених систем до хмарних обчислень, що дозволило підкреслити переваги останніх — масштабованість, гнучкість, зниження витрат на інфраструктуру та високу доступність.

Також детально проаналізовано архітектурні підходи до побудови хмарних сервісів, особливості взаємодії клієнта і сервера у середовищі RESTful-сервісів, що є сучасним стандартом побудови веб-застосунків. Описано принципи роботи протоколу RESTful, який забезпечує легкість інтеграції, масштабованість і незалежність компонентів.

Окрему увагу приділено питанням безпеки та перевагам архітектури хмарних обчислень у контексті створення надійних і доступних сервісів. Отримані результати створюють фундамент для подальшого проектування та реалізації хмарної системи тестування знань з урахуванням сучасних технологічних вимог і практик.

РОЗДІЛ 2. АРХІТЕКТУРНА ТА АЛГОРИТМІЧНА РЕАЛІЗАЦІЯ ХМАРНОЇ СИСТЕМИ ТЕСТУВАННЯ ЗНАНЬ

2.1. Порівняльний аналіз сучасних хмарних систем тестування та традиційних систем оптичного розпізнавання відповідей

В оцінюванні рівня знань застосовуються різноманітні підходи та технологічні рішення. Розглянемо характеристики сучасної хмарної системи тестування та проведемо її порівняння з традиційними системами оптичного розпізнавання відповідей (Optical Mark Recognition, OMR).

Типові можливості та переваги, що реалізуються в сучасних хмарних платформах для оцінювання знань, включають:

- Механізми забезпечення цілісності процесу тестування, тобто впровадження засобів протидії недоброчесності, таких як обмеження на функції пристрою під час тестування (наприклад, заборона створення знімків екрана, запису екрана, переходу між застосунками) та моніторинг дій користувача.

- Автоматизована рандомізація, тобто можливість автоматичного змішування послідовності запитань у тесті та/або варіантів відповідей до кожного запитання для різних користувачів.

- Автоматичний розрахунок статистики і надання функціональності для автоматизованого обчислення статистичних показників за результатами проходження тестування (наприклад, середній бал, розподіл відповідей, аналіз складності питань).

- Миттєвий зворотний зв'язок, а саме негайне надання користувачеві результатів тестування одразу після його завершення.

- Висока доступність рішення з мінімальними або відсутніми початковими інвестиціями в обладнання.

- Відносно низькі або відсутні витрати на поточне обслуговування системи.

- Можливість автоматичного формування та завантаження звітів з результатами у портативних форматах (наприклад, PDF, CSV) для подальшого аналізу або друку.

- Сукупне зниження витрат для користувачів та освітніх установ порівняно з альтернативними методами.

- Можливість налаштування структури та формату тестів, підтримка різних типів питань (текстові, множинний вибір тощо).

- Зручний доступ до системи з мобільних пристроїв через спеціалізовані застосунки.

Традиційні OMR-системи мають певні особливості, що включають як переваги, так і суттєві недоліки в сучасному контексті.

Переваги:

- Історично інтегровані в інфраструктуру багатьох освітніх установ.

- Відносно знайомі користувачам, які мали досвід роботи з ними раніше.

Недоліки:

- Високі початкові витрати, бо є необхідність придбання спеціалізованого обладнання (сканерів).

- Постійні витрати на обслуговування обладнання, придбання витратних матеріалів (спеціальні бланки).

- Можлива ненадійність розпізнавання відповідей через якість заповнення бланків, стан бланків або обладнання.

- Як правило, відсутні вбудовані засоби автоматичного розрахунку детальної статистики або складного аналізу результатів.

- Практично повна відсутність технічних засобів для запобігання недоброчесності під час проходження тестування.

Порівнюючи дві зазначені парадигми, стає очевидним, що сучасні хмарні системи тестування пропонують значно ширший спектр функціональних можливостей та вирішують багато проблем, властивих традиційним OMR-системам. Зокрема, вони виграють за показниками економічної ефективності (нижчі початкові та операційні витрати), наявності вбудованих механізмів безпеки та протидії недоброчесності, а також автоматизації процесів аналізу та генерації звітів. Хоча OMR-системи можуть бути знайомими та вже інтегрованими, їх обмеженість у функціональності та значні витрати на підтримку роблять їх менш привабливими в умовах сучасних вимог до гнучкості, безпеки та аналітичних можливостей систем оцінювання.

На основі проведеного порівняльного аналізу характеристик, сучасні хмарні системи тестування демонструють переконливі переваги перед традиційними системами оптичного розпізнавання відповідей. Реалізовані в хмарних рішеннях функціональні можливості, зокрема вдосконалені засоби безпеки, автоматизація процесів оцінювання та аналізу, а також суттєва економічна вигода, роблять їх більш ефективним, надійним та сучасним інструментом для проведення оцінювання знань у широкому спектрі застосувань.

2.2. Ключові функціональні можливості та аспекти реалізації системи тестування знань

Розділ описує ключові функціональні можливості та аспекти реалізації системи тестування знань, спрямовані на забезпечення цілісності оцінювання, ефективності адміністрування та доступності для користувачів.

В рамках системи реалізовано комплекс механізмів протидії недоброчесності з метою забезпечення академічної доброчесності під час тестування. Першочерговим завданням є блокування можливості створення

знімків екрана (скріншотів) та запису екрана в межах клієнтського застосунку під час активної сесії тестування. Також передбачено функцію моніторингу активності користувача, що фіксує випадки виходу з програми під час проходження тесту. У випадку фіксації перевищення встановленої межі дозволеної кількості виходів (наприклад, більше одного разу), тест буде автоматично завершено та негайно надіслано на сервер для оцінювання. Доступ до записів про подібні інциденти надається викладачам або адміністраторам для подальшого аналізу та прийняття відповідних заходів реагування.

Для контролю дотримання часових обмежень на виконання тесту реалізовано функцію таймера. Цей механізм автоматично завершує можливість надсилання відповідей після спливання призначеного для тесту часу, гарантуючи дотримання регламенту оцінювання.

Система також надає адміністративні функції для викладачів, зокрема можливість видалення створених тестів. Постійно проводяться роботи з покращення серверної частини програмного забезпечення з метою оптимізації обчислювальних процесів та підвищення загальної продуктивності системи. Реалізовано метод рандомізованого перемішування послідовності запитань тесту для кожного окремого студента, що є ефективним заходом для запобігання обміну відповідями та підвищення об'єктивності результатів.

Клієнтські застосунки розробляються для підтримки мобільних пристроїв під управлінням провідних операційних систем – Android та iOS, що забезпечує широку сумісність та доступність для більшості потенційних користувачів (студентів). Також враховується можливість використання системи на планшетних пристроях, що може слугувати альтернативним варіантом доступу у випадках, коли власні пристрої студентів недоступні або мають обмежену сумісність.

Розміщення серверної частини системи на хмарному рішенні є архітектурним рішенням, обраним з метою забезпечення ключових нефункціональних вимог, таких як висока масштабованість (здатність системи ефективно функціонувати при зростанні кількості користувачів та навантаження), надійність (забезпечення безперебійного доступу до сервісу) та стабільність функціонування, що є критично важливим, особливо під час проведення синхронних масових екзаменів.

2.3. Проектування діаграм розгортання та опис інтерфейсів користувача

Цей додаток був розроблений в Android Studio та XCode. Ці функції були реалізовані за допомогою: Java, Swift, Obj-C, PHP, HTTP-запитів та MySQL. Додаток взаємодіє з базою даних через вебсервер Nginx.

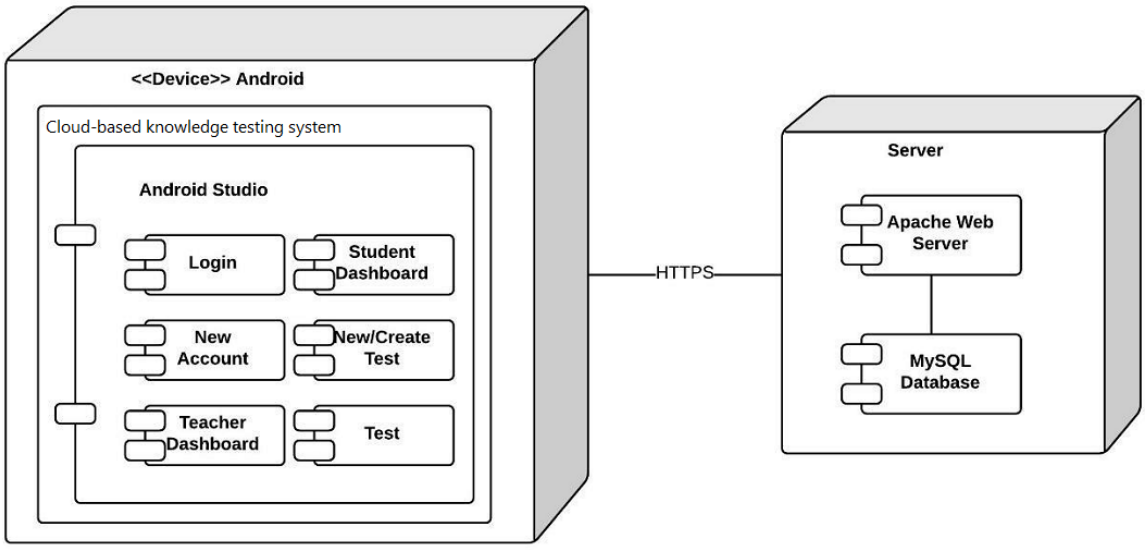

Рисунок 2.1 – Діаграма розгортання пропонованої системи

На рисунку 2.1 представлена діаграма, що ілюструє архітектуру хмарно-базованої системи тестування знань, зосереджуючись на взаємодії

між мобільним клієнтським пристроєм під управлінням операційної системи Android та серверною частиною.

Діаграма складається з двох основних компонентів (контейнерів):

- **Клієнтський пристрій (Android Device).** Представлений блоком з позначкою <<Device>> Android. Всередині цього блоку зображена сама система (Cloud-based knowledge testing system), яка, своєю чергою, містить кілька внутрішніх модулів або функціональних частин:

- Android Studio: Хоча Android Studio є інтегрованим середовищем розробки, у контексті цієї діаграми блок позначає програмне забезпечення, розроблене в цьому середовищі, або базовий контекст мобільного застосунку.

- Login: Модуль, що відповідає за автентифікацію користувача.

- New Account: Модуль для створення нових облікових записів користувачів.

- Teacher Dashboard: Інтерфейс або модуль для викладачів, що надає функції управління тестами та результатами.

- Student Dashboard: Інтерфейс або модуль для студентів, що відображає їхній прогрес та доступні тести.

- New/Create Test: Модуль, що дозволяє викладачам створювати нові тести.

- Test: Модуль, що відповідає за процес проходження тестування студентами. Кожен з цих модулів має інтерфейси (позначені кружечками та сокетами), що вказує на їхню взаємодію між собою та/або із зовнішніми компонентами.

- **Сервер (Server).** Представлений окремим блоком, що містить два основні компоненти:

- Apache Web Server: Вебсервер, що відповідає за прийом та обробку вхідних запитів від клієнтських пристроїв та надсилання відповідей.

- MySQL Database: Система управління базами даних, що використовується для зберігання даних системи (інформація про користувачів, тести, результати тощо). Зображено зв'язок між вебсервером та базою даних, що означає, що вебсервер взаємодіє з базою даних для отримання та збереження інформації.

Зв'язок між клієнтським пристроєм (системою тестування на Android) та сервером показано однією лінією, позначеною HTTPS. Це вказує на те, що комунікація між мобільним застосунком та серверною частиною здійснюється за допомогою безпечного протоколу HTTP (Hypertext Transfer Protocol Secure), що забезпечує шифрування даних під час передачі.

Загалом, діаграма ілюструє двокомпонентну клієнт-серверну архітектуру, де мобільний застосунок на пристрої Android (з різними функціональними модулями) взаємодіє з серверною частиною, яка складається з вебсервера Apache та бази даних MySQL, використовуючи безпечний протокол HTTPS для обміну даними.

2.3.1. Інтерфейси користувача

Тепер розглянемо основні інтерфейси користувача пропонованої системи тестування знань.

Сторінка входу - дозволяє існуючому користувачеві увійти до свого облікового запису та надає посилання на сторінку створення нового облікового запису. Успішний вхід перенаправляє на сторінку панелі управління викладача. Студенти не потребують входу і продовжать як студенти.

Сторінка створення нового облікового запису - дозволяє викладачеві створити обліковий запис для початку управління тестами та оцінками.

Сторінка панелі управління викладача - ця сторінка містить список усіх тестів, створених викладачем, і дозволяє викладачеві переглядати оцінки

студентів та ключі для кожного тесту. Ця сторінка також дозволяє викладачеві створити новий тест.

Сторінка створення тесту - ця сторінка дозволяє викладачеві створити новий тест. Викладач зможе згенерувати випадковий ідентифікатор тесту та вказати кількість запитань.

Сторінка студента - використовується для зберігання всієї інформації про одного конкретного студента в курсі. Включає зараховані курси та всі попередні оцінки на екзаменах.

Сторінка викладача - використовується для зберігання всіх даних про облікові записи викладачів. Тут зберігаються всі курси, створені викладачами, разом з усіма екзаменами, створеними викладачами.

Сторінка тесту - Використовується для зберігання екзаменів, які використовуються викладачами на даний момент.

Apache Web Server - Сервер, який використовується для хостингу хмарного сервера системи.

2.3.2. Інтерфейси програмного забезпечення

Java - Основна мова для розробки будь-яких функцій, які будуть виконуватися на пристроях Android.

Swift - Основна мова для розробки будь-яких функцій, які будуть виконуватися на пристроях Apple.

PHP - Використовується для спілкування з сервером та надсилання/отримання даних з бази даних.

HTTP - Спілкується з вебсервером. Також використовує JSON-запити для отримання/надсилання даних до бази даних.

MySQL - Використовується для створення бази даних та таблиць, в яких зберігаються дані, а також для виконання будь-яких необхідних запитів.

Цей додаток буде використовувати або мобільний зв'язок (4G, 5G) або Wi-Fi для запитів та надсилання даних з сервера.

Цей додаток розроблений для всіх систем Android, починаючи з Android 4.4 KitKat (API 19) і вище. Пристрої Apple, починаючи з iOS 10.3 і вище, також підтримуються. Ця підтримка забезпечує сумісність з 90% пристроїв.

2.4. Проектування Use Case діаграми системи тестування знань

Представлена діаграма (рис. 2.2) є діаграмою варіантів використання (Use Case Diagram), яка ілюструє функціональні вимоги до системи тестування знань з погляду різних категорій користувачів (акторів).

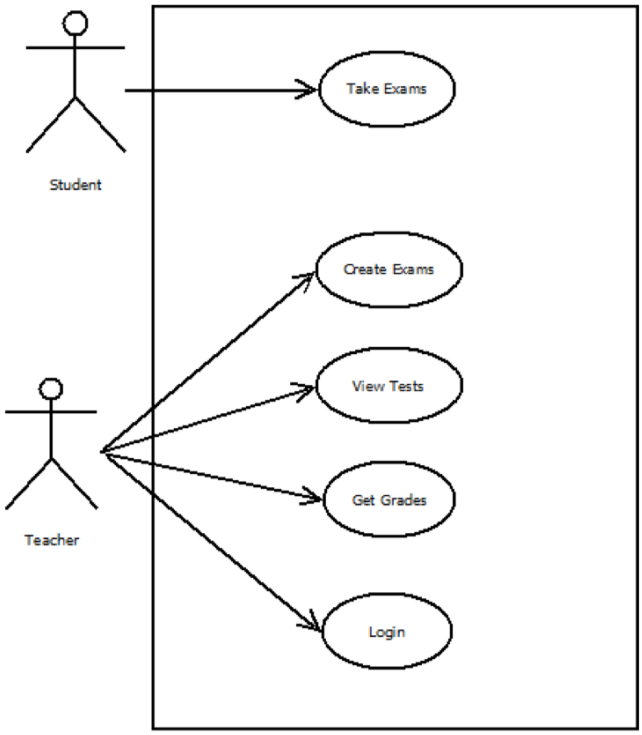

Рисунок 2.2 - Use Case діаграма системи тестування знань

Діаграма містить:

- Межа системи (System Boundary). Прямокутник, що окреслює межі самої системи, показуючи, які варіанти використання належать до даної системи.

- Актори (Actors). Зовнішні сутності, що взаємодіють із системою. На діаграмі представлено двох акторів:

- Student - користувач, який проходить тестування.

- Teacher - користувач, який створює, адмініструє та переглядає результати тестів.

- Варіанти використання (Use Cases) - овали всередині межі системи, що представляють собою функції або дії, які актори можуть виконувати за допомогою системи. На діаграмі визначено такі варіанти використання:

- Take Exams (Пройти екзамени) - функціональність, що дозволяє студенту пройти призначений тест.

- Create Exams (Створити екзамени) - функціональність, що дозволяє викладачеві створити нові тестові завдання.

- View Tests (Переглянути тести) - функціональність, що дозволяє викладачеві переглядати існуючі тести.

- Get Grades (Отримати оцінки) - функціональність, що дозволяє викладачеві переглядати результати або оцінки студентів за тести. (Зазвичай студенти також можуть переглядати свої оцінки, але на цій діаграмі цей варіант використання пов'язаний лише з викладачем).

- Login (Увійти) - функціональність, що дозволяє користувачам (Викладачеві) отримати доступ до системи шляхом автентифікації. (На діаграмі пов'язано лише з Викладачем, хоча, як правило, Студенти також виконують вхід).

- Зв'язки (Relationships) - лінії зі стрілками, що показують, які актори ініціюють або беруть участь у виконанні певних варіантів використання.

- Студент взаємодіє з варіантом використання "Take Exams".

- Викладач взаємодіє з варіантами використання "Create Exams", "View Tests", "Get Grades" та "Login".

Діаграма варіантів використання для системи тестування знань графічно представляє, як основні користувачі системи (Студент та Викладач)

взаємодіють із системою для виконання ключових функцій, таких як проходження тестів студентами, а також створення тестів, їх перегляд, отримання оцінок та вхід у систему викладачами. Вона надає високорівневий огляд функціональності системи з точки зору зовнішніх користувачів.

У розробці програмного забезпечення функціональні вимоги визначають функцію системи або її компонента.

Сторінка входу

Викладачі зможуть створити обліковий запис, якщо у них ще немає, і увійти за допомогою свого імені користувача та пароля. Якщо користувач є студентом, він повинен продовжити як студент.

Сторінка тесту

Користувач-студент зможе відповідати на запитання та надіслати їх до бази даних. Користувач-викладач використовуватиме цю сторінку для створення ключів тестів.

Вимоги до продуктивності

Додаток потребує стабільного інтернет-з'єднання для спілкування з базою даних. Час відповіді на індивідуальні оцінки для студента повинен бути в межах кількох хвилин з моменту надсилання відповідей студента.

Доступність

Додаток завжди буде доступним. Однак функціональність буде надана через доступ, який надають викладачі.

2.5. Характеристики системи та вимоги

У цьому розділі описано ключові характеристики та вимоги до системи тестування знань, що визначають її функціональність, обмеження, взаємодію з користувачами та іншими компонентами, а також аспекти безпеки та якості.

Характеристики користувача

Система передбачає дві основні ролі користувачів: Студент та Викладач. Обидві категорії користувачів можуть отримати доступ до функціональності системи через клієнтські застосунки, розгорнуті на їхніх мобільних пристроях.

Обмеження операційної системи

Для коректного функціонування клієнтського застосунку необхідна відповідність операційної системи пристрою визначеним мінімальним вимогам:

- Пристрої під управлінням операційної системи Android повинні використовувати версію 4.4 KitKat (API 19) або новішу.

- Пристрої під управлінням операційної системи iOS повинні використовувати версію 10.3 або новішу.

Припущення та залежності

Функціонування системи залежить від виконання наступних умов та наявності компонентів:

- Клієнтський застосунок потребує стабільного підключення до мережі Інтернет на мобільному пристрої користувача для взаємодії із серверною частиною.

- Необхідна наявність серверної інфраструктури з розгорнутою базою даних для централізованого зберігання інформації про користувачів та тестові матеріали.

Інтерфейси користувача

У цьому підрозділі описано ключові інтерфейси користувача (сторінки) системи, з якими безпосередньо взаємодіють користувачі.

Сторінка входу

При доступі до застосунку користувачеві надається інтерфейс автентифікації (сторінка входу), який забезпечує механізми ідентифікації та верифікації облікових даних користувача.

Сторінка тесту

Інтерфейс проходження тестування є основним вікном, де студенти взаємодіють з екзаменаційними завданнями, сформованими викладачем, надаючи відповіді на запитання.

Сторінка підтвердження надсилання

Цей інтерфейс відображається користувачеві після спроби завершення тестування та підтверджує статус успішної передачі результатів на сервер для подальшої обробки.

Апаратні інтерфейси

Підтримка апаратних платформ визначається сумісністю клієнтського програмного забезпечення з мобільними пристроями, що працюють під управлінням операційних систем Android версії 4.4 (API 19) та новіших, а також iOS версії 10.3 та новіших.

2.6. Інтерфейси програмного забезпечення та функціональні вимоги

Цей розділ описує взаємодію системи з іншим програмним забезпеченням та компонентами.

Інтерфейси спілкування

Для забезпечення обміну даними між клієнтським застосунком та серверною частиною (включаючи доступ до бази даних), система потребує встановлення мережевого з'єднання через Wi-Fi або мобільні мережі Інтернет.

Функціональні вимоги

Функціональні вимоги до системи визначають набір операцій та послуг, які система повинна виконувати для задоволення потреб користувачів та виконання своїх завдань.

Функціональність сторінки входу

Функціональність інтерфейсу автентифікації передбачає можливість створення нового облікового запису для користувачів з роллю Викладач (якщо така функція реалізована в застосунку) та здійснення процедури входу до системи шляхом введення та верифікації облікових даних (ім'я користувача та пароль). Користувачі з роллю Студент використовують аналогічний механізм автентифікації.

Функціональність сторінки тесту

Функціональні вимоги до інтерфейсу проходження тестування різняться залежно від ролі користувача:

- Користувач з роллю Студент має можливість переглядати запитання, надавати відповіді та ініціювати процедуру надсилання результатів на сервер.

- Користувач з роллю Викладач може використовувати цей або пов'язаний інтерфейс для визначення коректних відповідей (ключів) для тестових завдань під час їх створення або редагування.

Вимоги до продуктивності

Вимоги до продуктивності системи включають необхідність забезпечення стабільного мережевого з'єднання для коректної комунікації із серверною частиною. Час відгуку системи при надсиланні студентом відповідей та ініціації процедури отримання індивідуальної оцінки (якщо така функція передбачена) має перебувати в межах прийнятних часових рамок, наприклад, не перевищувати кількох хвилин з моменту завершення надсилання.

Обмеження проєктування

Обмеження проєктування визначаються цільовими апаратними платформами – система розроблена для функціонування виключно на мобільних пристроях під управлінням операційних систем Android та iOS, що накладає певні обмеження на можливості розгортання та використання на інших типах пристроїв.

Атрибути програмного забезпечення

Атрибути програмного забезпечення визначають нефункціональні вимоги до системи, що стосуються якості її функціонування, включаючи надійність, доступність та безпеку.

Надійність

Надійність системи оцінюється через її здатність коректно функціонувати на широкому спектрі мобільних пристроїв під управлінням підтримуваних версій операційних систем Android та iOS, мінімізуючи ймовірність збоїв або некоректної роботи.

Доступність

Доступність системи як програмного продукту для встановлення та запуску є постійною. Проте доступність до специфічної функціональності (наприклад, проходження конкретного тесту або доступ до адміністративних функцій) регулюється механізмами управління доступом, що контролюються користувачами з роллю Викладач.

2.7. Безпека системи і тестування

Безпека системи передбачає комплекс заходів, спрямованих на захист даних та функціональності від несанкціонованого доступу, зловмисних дій або випадкових пошкоджень, а також на забезпечення цілісності процесу тестування.

Заходи безпеки на системному рівні включають: обмеження доступу студентів до проходження тесту виключно в межах визначеного інтервалу часу та встановленої тривалості тесту; фіксацію та реєстрацію підозрілих дій користувача в межах застосунку (наприклад, спроби переведення застосунку у фоновий режим або створення знімків екрана). Використання веб-фреймворку (наприклад, Laravel) на стороні сервера надає певні вбудовані механізми захисту та валідації даних (хоча твердження про приховування

ним вихідного коду є некоректним для інтерпретованих мов). Механізм рандомізованого перемішування запитань тесту для кожного студента є засобом протидії прямому копіюванню відповідей між користувачами, сприяючи унікальності екзаменаційного варіанту для кожного учасника.

Тестування програмного забезпечення є процесом оцінки якості розробленої системи шляхом виявлення дефектів, перевірки відповідності функціональним та нефункціональним вимогам, а також оцінки її надійності та безпеки. Воно включає декілька ключових етапів.

Модульне тестування передбачає ізольовану перевірку функціональності окремих, мінімально ділимих компонентів системи (модулів), таких як окремі екрани інтерфейсу користувача або програмні класи, для підтвердження їх коректної роботи відповідно до специфікацій.

Інтеграційне тестування спрямоване на перевірку коректності взаємодії та обміну даними між окремими модулями після їх об'єднання в єдине ціле (наприклад, інтеграція різних екранів у мобільний застосунок) шляхом виконання різних сценаріїв використання, що охоплюють взаємодію компонентів.

Приймальне тестування є фінальним етапом тестування, що проводиться після успішного модульного та інтеграційного тестування. Його основна мета – перевірити відповідність функціональності та нефункціональних атрибутів системи вимогам кінцевого замовника (або ключових користувачів) та підтвердити готовність продукту до розгортання та експлуатації в реальному середовищі.

2.8. Реалізація індивідуального дизайну архітектури

На рисунку 2.3 представлена колекція діаграм, об'єднаних під загальною назвою "Individual Architecture Design" (дизайн індивідуальної архітектури). Кожна з дев'яти менших діаграм, є спрощеною діаграмою

класів або компонентів, що ілюструє структуру ключових елементів клієнтської частини системи, найімовірніше, мобільного застосунку під Android, зважаючи на використання таких базових класів як AppCompatActivity та ArrayAdapter.

Кожна з менших діаграм демонструє окремий програмний модуль, переважно або екран користувацького інтерфейсу (Activity), або компонент для відображення списків даних (Adapter).

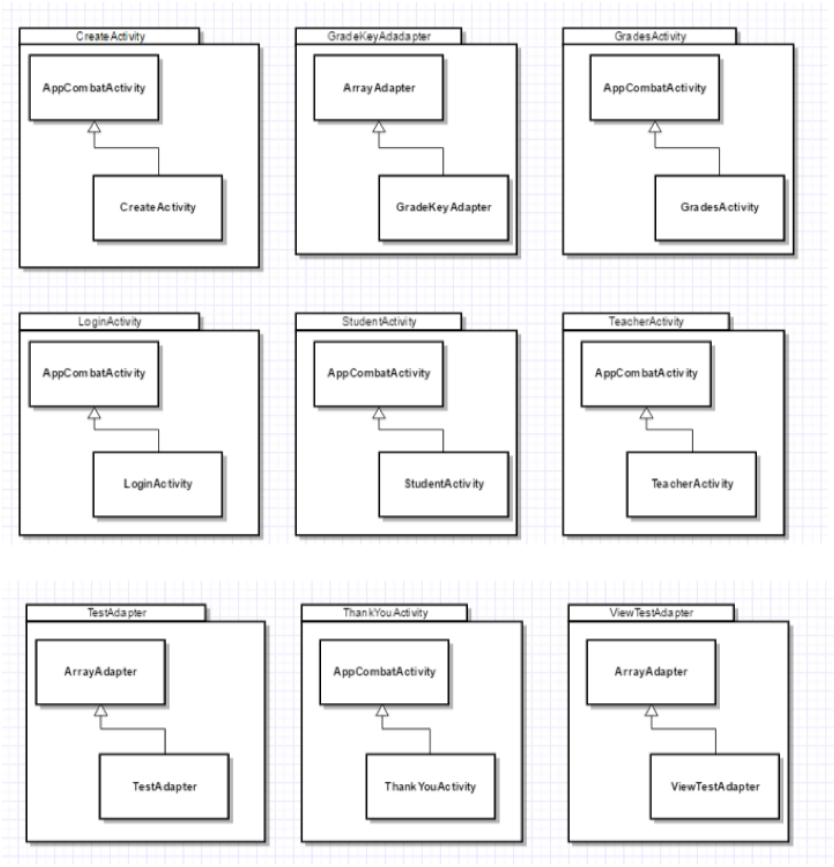

Рисунок 2.3 - індивідуального дизайну архітектури

Загалом, діаграма показує:

- Модулі екранів (Activity Modules). Шість діаграм (CreateActivity, GradesActivity, LoginActivity, StudentActivity, TeacherActivity, ThankYouActivity) ілюструють структуру класів, що відповідають за конкретні екрани або інтерфейси користувача застосунку. Кожна з цих

діаграм показує, що відповідний клас (наприклад, LoginActivity) успадковується від стандартного базового класу Android AppCompatActivity, який надає базову функціональність для екранів додатку, сумісну з різними версіями Android.

- Модулі адаптерів (Adapter Modules): Три діаграми (GradeKeyAdapter, TestAdapter, ViewTestAdapter) ілюструють структуру класів, що відповідають за адаптери даних. Адаптери в Android використовуються для зв'язування даних зі списковими компонентами інтерфейсу (наприклад, ListView або RecyclerView). Кожна з цих діаграм показує, що відповідний клас адаптера (наприклад, TestAdapter) успадковується від стандартного базового класу Android ArrayAdapter, який є адаптером для роботи з масивами або списками даних.

Призначення окремих модулів:

- CreateActivity: Екран для створення нового об'єкта (нового тесту або екзамену).

- GradesActivity: Екран для перегляду оцінок.

- LoginActivity: Екран для входу користувача в систему.

- StudentActivity: Основний екран або "дашборд" для користувачів з роллю Студент.

- TeacherActivity: Основний екран або "дашборд" для користувачів з роллю Викладач.

- ThankYouActivity: Екран підтвердження, що відображається після виконання певної дії (наприклад, після надсилання тесту).

- GradeKeyAdapter: Адаптер для відображення даних, пов'язаних з ключами оцінювання або правильними відповідями.

- TestAdapter: Адаптер для відображення інформації про тести.

- ViewTestAdapter: Адаптер, для відображення інформації про тести в режимі перегляду.

Отже, дана діаграма представляє деталі внутрішньої архітектури клієнтського мобільного застосунку, показуючи структуру ключових компонентів користувацького інтерфейсу (Activity) та компонентів для представлення даних (Adapter). Вона ілюструє використання стандартних патернів розробки під Android, де конкретні реалізації екранів та адаптерів успадковують функціональність від базових системних класів.

Висновки до розділу

У другому розділі було проведено ґрунтовний аналіз сучасних підходів до реалізації хмарних систем тестування знань, а також здійснено порівняння з традиційними системами оптичного розпізнавання відповідей. Встановлено, що хмарні рішення значно перевершують за гнучкістю, масштабованістю та зручністю використання.

Розглянуто ключові функціональні можливості системи, такі як автоматизація перевірки, збереження результатів, адаптивне тестування та інші. Особливу увагу приділено проектуванню інтерфейсів користувача та програмного забезпечення, що забезпечують зручну взаємодію з системою як для студентів, так і для адміністраторів.

Розроблено Use Case діаграму, яка відображає основні сценарії взаємодії з системою. Описано архітектуру розгортання, функціональні вимоги до системи, а також характеристики її безпечної та стабільної роботи в умовах хмарного середовища.

Важливою складовою реалізації стала індивідуалізація архітектурних рішень, що дозволило адаптувати систему під конкретні потреби користувачів. Таким чином, розділ заклав надійну технічну базу для створення повноцінної, ефективної та безпечної хмарної системи тестування знань.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ХМАРНО-БАЗОВАНОЇ СИСТЕМИ ТЕСТУВАННЯ ЗНАНЬ

3.1. Архітектура та реалізація системи тестування знань

Реалізація розглядуваної системи тестування знань базується на принципах сервіс-орієнтованої архітектури (SOA), що передбачає структурування функціональності у вигляді набору взаємодіючих сервісів. Така архітектура забезпечує високу гнучкість, можливість повторного використання компонентів та спрощення інтеграції.

Серверна частина системи розгорнута у хмарному середовищі на віртуальному приватному сервері (VPS). Вона реалізована з використанням PHP-фреймворку Laravel, який відповідає шаблону проектування Модель-Представлення-Контролер (MVC). Laravel є фреймворком, призначеним для розробки веб-додатків, і надає низку важливих функціональних можливостей:

- Модульна система пакування з інтегрованим менеджером залежностей.

- Різноманітні методи доступу та взаємодії з реляційними базами даних.

- Інструменти, що спрощують процеси розгортання та обслуговування додатків.

- Орієнтація на розробку із застосуванням зрозумілого синтаксису коду.

З метою мінімізації обчислювального навантаження на клієнтські мобільні пристрої, основні обчислювальні завдання виконуються на стороні хмарного сервера. Це включає:

- Оцінювання відповідей студентів.

- Генерацію статистичних даних за результатами екзаменів (наприклад, аналіз частоти вибору відповідей для кожного питання).

- Формування друкованих документів, зокрема у форматі PDF (наприклад, звітів за результатами тестування).

Взаємодія між мобільними клієнтськими застосунками та серверною частиною реалізована за RESTful парадигмою. Відповідно до принципів Representational State Transfer (REST), комунікація здійснюється через стандартизовані HTTP-запити, такі як GET (отримання даних), POST (створення/відправка даних), PUT (оновлення даних) та DELETE (видалення даних).

Сервіс-орієнтована архітектура, застосована в системі, надає кілька ключових переваг:

- Сприяє агрегації та повторному використанню наявних функціональних можливостей сервісів для створення нових додатків або розширення існуючих.

- Надає набір принципів проектування, які структурують процес розробки системи та забезпечують механізми для інтеграції різнорідних компонентів як у централізовані, так і в децентралізовані конфігурації.

- Дозволяє здійснювати декомпозицію загальної функціональності системи на дискретний набір взаємодіючих сервісів, які можуть бути інтегровані у різні програмні системи, що належать до різних бізнес-доменів.

Розробка системи здійснюється з використанням стандартних веб-технологій та мов програмування, зокрема HTML, CSS, PHP та JavaScript. Управління версіями програмного коду здійснюється за допомогою децентралізованої системи контролю версій Git. Серверне оточення на базі Nginx та з використанням бази даних MySQL для зберігання даних було налаштовано за допомогою відповідних інструментів, включаючи командний термінал.

Для забезпечення безпечного та довіреного каналу зв'язку між клієнтами та сервером, хмарний сервер оснащений сертифікатом SSL (Secure Sockets Layer). Мобільні клієнтські застосунки розробляються нативними засобами для відповідних платформ з використанням інтегрованих середовищ розробки XCode (для iOS) та Android Studio (для Android).

Архітектура системи проєктувалася з урахуванням можливості її масштабування та одночасного використання багатьма різними організаціями без необхідності внесення змін у базовий програмний код, що є реалізацією принципу мультиорендності та переваг розгортання у хмарному середовищі.

3.2. Проектування дизайну архітектури додатку

На рисунку 3.1 показано дизайн архітектури додатку.

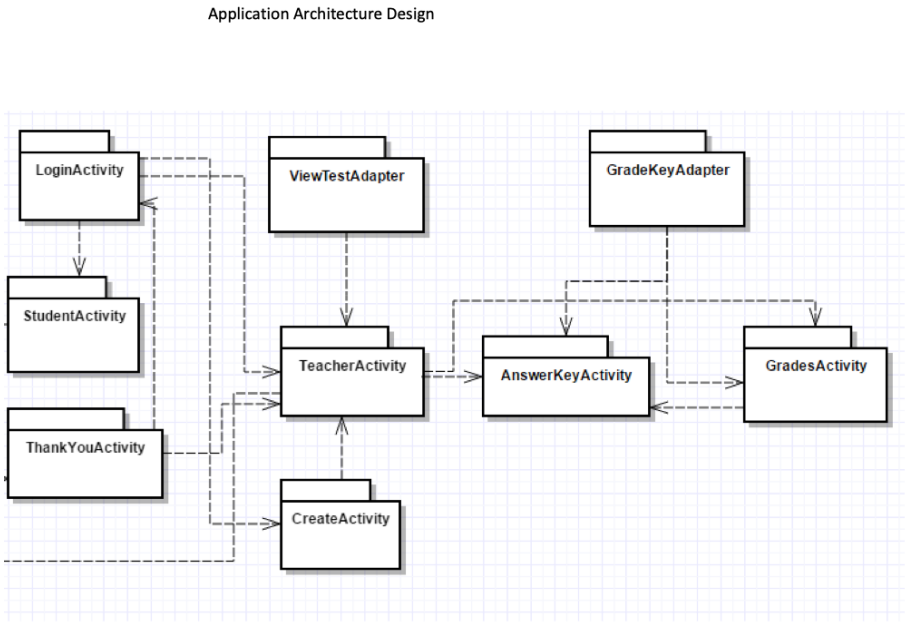

Рисунок 3.1 - Дизайн архітектури додатку

LoginActivity - Коли користувач входить до додатка, він зіткнеться зі сторінкою входу. Якщо користувач є викладачем, він увійде за допомогою свого імені користувача та пароля. Якщо користувач є студентом, є кнопка,

яка дозволяє користувачеві продовжити як студент. Також є кнопка "Створити обліковий запис", якщо у викладача ще немає облікового запису.

WarningActivity - Відображає попереджувальне повідомлення про те, що додаток використовує механізми протидії обману. Додаток забороняє створення скріншотів, вихід з додатка та інші функції.

TeacherActivity - Після того, як викладач вибере ключ тесту, який вже був створений, його перенаправляють на цю сторінку. Ця сторінка містить три різні колонки, серед яких користувач може вибирати. Один містить оцінки студентів для цього конкретного тесту. Середня колонка містить відповіді на тест, які можна редагувати, а також індивідуальні ваги, пов'язані з ними. Остання колонка містить налаштування для цього конкретного тесту.

ThankyouActivity - Повідомляє користувачеві, що їхній тест був успішно надісланий.

3.3. Проектування бази даних

База даних складається з наступних таблиць.

Таблиця Test - Зберігає інформацію про тести, включаючи ідентифікатор тесту, ім'я тесту, кількість запитань, час створення та час оновлення.

Таблиця Question - Зберігає інформацію про запитання, включаючи ідентифікатор запитання, текст запитання, правильну відповідь та вагу запитання.

Таблиця Student - Зберігає інформацію про студентів, включаючи ідентифікатор студента, ім'я студента, адресу електронної пошти та пароль.

Таблиця Teacher - Зберігає інформацію про викладачів, включаючи ідентифікатор викладача, ім'я викладача, адресу електронної пошти та пароль.

Таблиця Submission - Зберігає інформацію про надіслані тести, включаючи ідентифікатор надісланого тесту, ідентифікатор студента, ідентифікатор тесту, час надіслання та оцінку.

Розглянемо атрибути таблиць та зв’язки між ними.

Таблиця Test

- test_id (PK, integer)
- test_name (varchar)
- question_count (integer)
- created_at (timestamp)
- updated_at (timestamp)

Таблиця Question

- question_id (PK, integer)
- test_id (FK, integer, посилається на Test.test_id)
- question_text (varchar)
- correct_answer (varchar)
- weight (integer)

Таблиця Student

- student_id (PK, integer)
- student_name (varchar)
- email (varchar, унікальний)
- password (varchar)

Таблиця Teacher

- teacher_id (PK, integer)
- teacher_name (varchar)
- email (varchar, унікальний)
- password (varchar)

Таблиця Submission

- submission_id (PK, integer)
- student_id (FK, integer, посилається на Student.student_id)

- test_id (FK, integer, посилається на Test.test_id)

- submission_time (timestamp)

- score (integer)

Представимо зв’язки між таблицями.

Test ↔ Question (один до багатьох):

- Один тест (Test) може мати багато запитань (Question).

- Поле test_id у таблиці Question є зовнішнім ключем, що посилається на test_id у таблиці Test.

Student ↔ Submission (один до багатьох):

- Один студент (Student) може мати багато надісланих тестів (Submission).

- Поле student_id у таблиці Submission є зовнішнім ключем, що посилається на student_id у таблиці Student.

Test ↔ Submission (один до багатьох):

- Один тест (Test) може мати багато надісланих результатів (Submission).

- Поле test_id у таблиці Submission є зовнішнім ключем, що посилається на test_id у таблиці Test.

Лістинг 3.1. Код побудови таблиць бази даних

```
    CREATE TABLE Test (
    test_id INTEGER PRIMARY KEY,
    test_name VARCHAR(255) NOT NULL,
    question_count INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Question (
    question_id INTEGER PRIMARY KEY,
    test_id INTEGER NOT NULL,
    question_text VARCHAR(1000) NOT NULL,
    correct_answer VARCHAR(255) NOT NULL,
```

```
        weight INTEGER NOT NULL,
        FOREIGN KEY (test_id) REFERENCES Test(test_id) ON DELETE
CASCADE
    );

    CREATE TABLE Student (
        student_id INTEGER PRIMARY KEY,
        student_name VARCHAR(255) NOT NULL,
        email VARCHAR(255) NOT NULL UNIQUE,
        password VARCHAR(255) NOT NULL
    );

    CREATE TABLE Teacher (
        teacher_id INTEGER PRIMARY KEY,
        teacher_name VARCHAR(255) NOT NULL,
        email VARCHAR(255) NOT NULL UNIQUE,
        password VARCHAR(255) NOT NULL
    );

    CREATE TABLE Submission (
        submission_id INTEGER PRIMARY KEY,
        student_id INTEGER NOT NULL,
        test_id INTEGER NOT NULL,
        submission_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        score INTEGER,
        FOREIGN KEY (student_id) REFERENCES Student(student_id)
ON DELETE CASCADE,
        FOREIGN KEY (test_id) REFERENCES Test(test_id) ON DELETE
CASCADE
    );
```

3.4. Представлення інтерфейсу користувача в режимі студента

На рисунку 3.2 представлено два скріншоти інтерфейсу користувача мобільного за стосунку для авторизації студента в розробленій системі тестування знань.

Обидва скріншоти відображають одну й ту саму сторінку з простим дизайном на синьому фоні. Основними елементами інтерфейсу є:

- Три текстові мітки (лейбли): "Student Name", "Student Id", "Test Code".
- Три поля для введення тексту, розташовані під відповідними мітками.
- Кнопка "Take Test" (Пройти тест), розташована під полями введення.

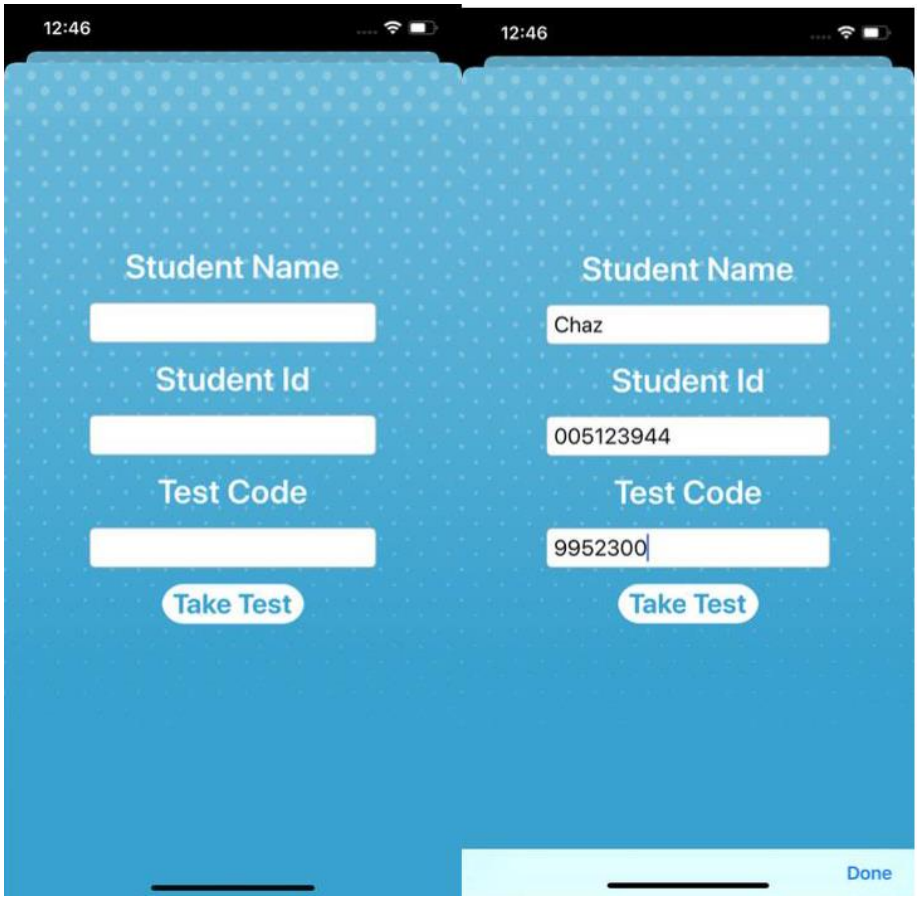

Рисунок 3.2 – Сторінка входу в систему (роль: студент)

Дана сторінка є інтерфейсом для ідентифікації студента та введення коду конкретного тесту, який він бажає пройти, перед початком тестування. Кнопка "Take Test" ініціює процес переходу до тесту після введення необхідних даних.

У системі тестування знань, зокрема в її клієнтському застосунку, інтегровано комплекс механізмів, спрямованих на забезпечення цілісності процесу оцінювання та протидію недоброчесності під час проходження тестування. Ці механізми включають автоматизоване рандомізоване перемішування послідовності запитань та відповідних варіантів відповідей,

блокування можливості створення знімків екрана та запису відео з екрана пристрою в межах застосунку, а також контроль за спробами користувача передчасно завершити роботу програми шляхом виходу з неї.

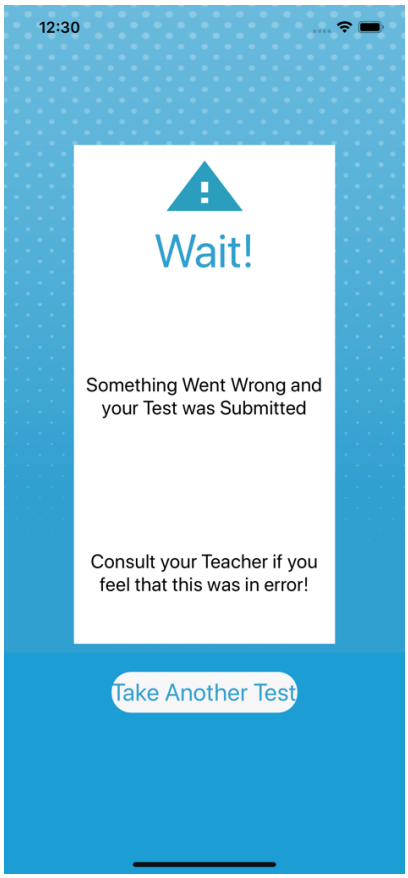

Рисунок 3.3 - Екран при фіксації порушення студентом

В рамках реалізації механізмів контролю за цілісністю процесу тестування, система здійснює моніторинг стану клієнтського застосунку на пристрої користувача. Зокрема, фіксується факт переведення застосунку у фоновий режим або його передчасного завершення користувачем під час активної сесії тестування. При виявленні такої події, ініціюється відображення відповідного інтерфейсу користувача (наприклад, екрана попередження або повідомлення про автоматичне надсилання тесту). Подія, що спричинила втрату застосунком активного стану (перехід у фоновий режим), автоматично реєструється у системному журналі та передається на

сервер для подальшого аналізу та формування звітів про поведінку користувача під час тестування.

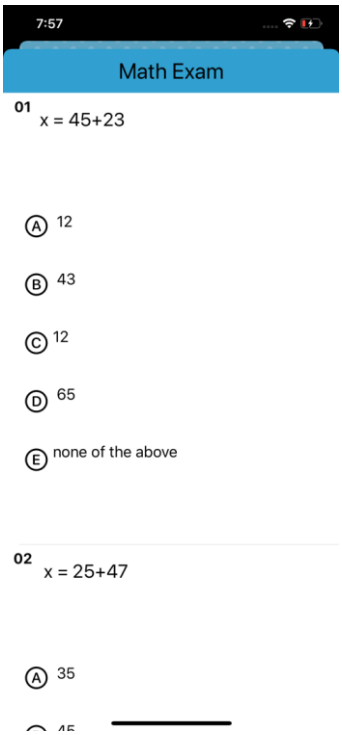

Рисунок 3.4 – Вигляд проходження тестування студентом

3.5. Представлення інтерфейсу користувача в режимі викладача

На рисунку 3.5 показано екран входу для викладача.

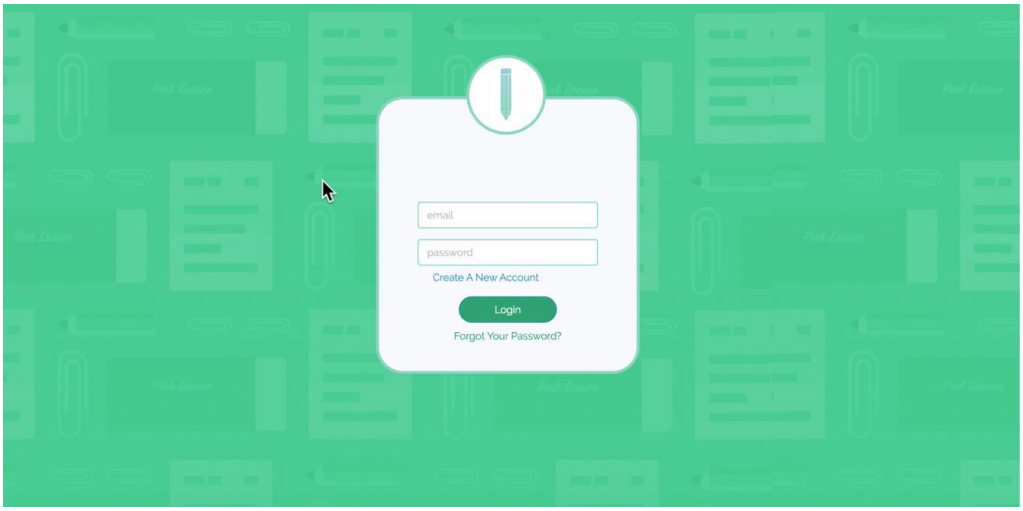

Рисунок 3.5 – Сторінка вхожу для викладача

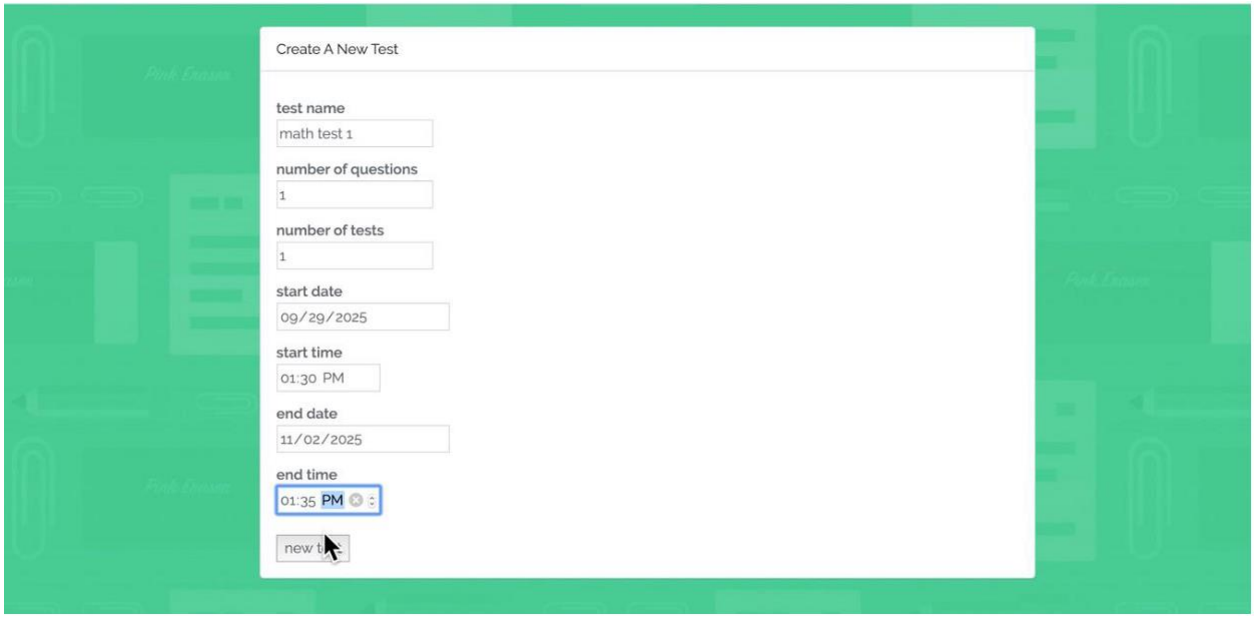

Рисунок 3.6 – Вікно створення нового тесту

На рисунку 3.6 представлено інтерфейс користувача для створення нового тесту в системі. Цей екран дозволяє користувачеві (викладачеві) визначити ключові параметри майбутнього тесту.

На інтерфейсі присутні поля для введення наступної інформації:

- test name: Назва тесту.

- number of questions: Загальна кількість запитань у тесті.

- number of tests: Кількість варіантів тесту або кількість спроб проходження.

- start date: Дата початку періоду, протягом якого тест буде доступний для проходження.

- start time: Час початку періоду, протягом якого тест буде доступний для проходження.

- end date: Дата завершення періоду, протягом якого тест буде доступний для проходження.

- end time: Час завершення періоду, протягом якого тест буде доступний для проходження.

Також на екрані присутня кнопка з позначкою "new" (новий), яка ініціює процес збереження введених даних та створення нового тесту на основі заданих параметрів.

Таким чином, функціональність, показана на цьому скріншоті, полягає у наданні інтерфейсу для конфігурації та ініціації створення нового тесту в системі, дозволяючи визначити його назву, обсяг, часові рамки доступу.

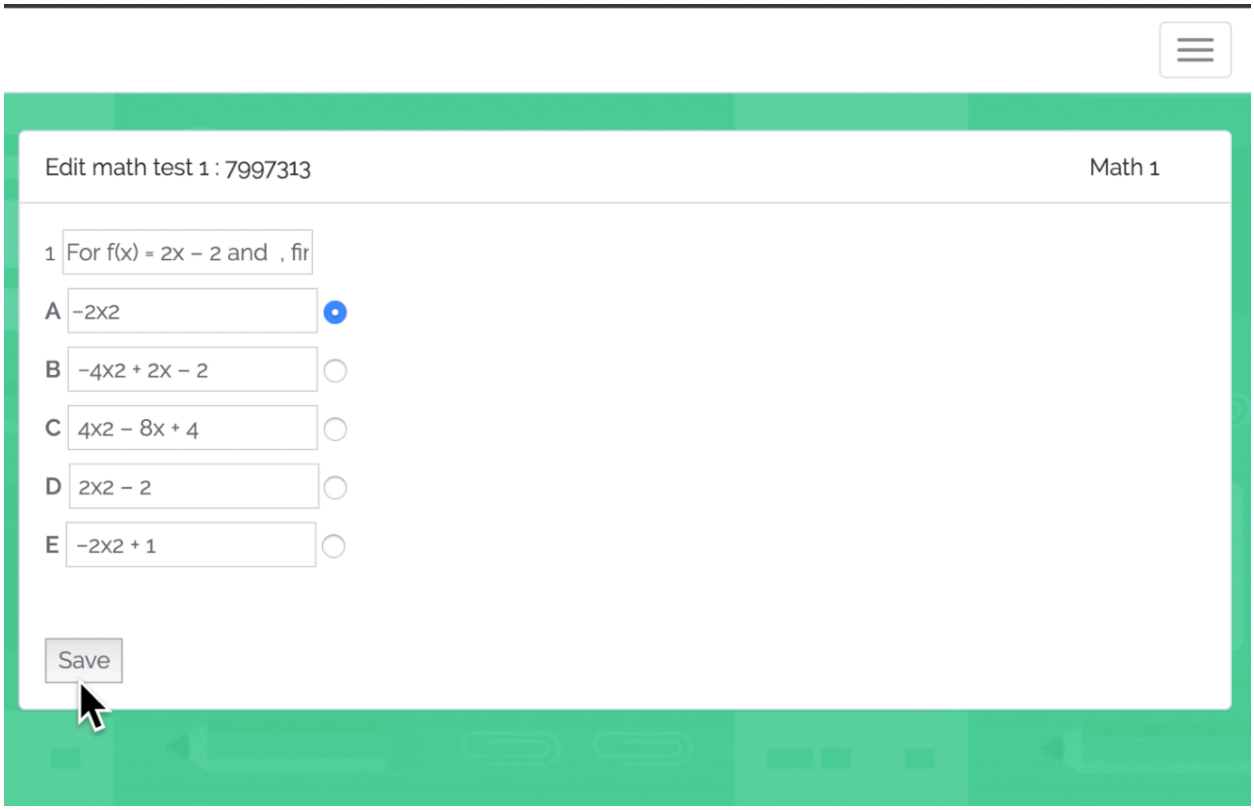

Рисунок 3.7 – Вікно редагування тесту

На рисунку 3.7 представлено інтерфейс користувача для редагування існуючого тесту, зокрема для внесення змін до окремих запитань та варіантів відповідей.

На екрані відображається форма редагування конкретного тесту з назвою "Edit math test 1 : 7997313" та категорією "Math 1". Інтерфейс зосереджений на редагуванні окремих запитань тесту і включає наступні елементи функціональності:

- Відображення запитання. Відображається номер запитання (у даному випадку "1") та текстова область з текстом самого запитання ("For f(x) = 2x - 2 and , fir...").

- Редагування варіантів відповідей. Присутні поля для введення тексту п'яти варіантів відповідей, позначених літерами A, B, C, D, E. Це дозволяє користувачеві змінювати текст кожного варіанта відповіді.

- Визначення правильної відповіді. Біля кожного поля варіанта відповіді є елемент вибору (схожий на радіокнопку). Можливість встановлення позначки біля одного з варіантів (як показано біля варіанта A синьою точкою) свідчить про функціональність вибору та позначення коректної відповіді для даного запитання.

- Збереження змін: Кнопка "Save" (Зберегти) дозволяє користувачеві застосувати внесені зміни до запитання та його варіантів відповідей, зберігаючи оновлену інформацію в системі.

Таким чином, функціональність, показана на цьому скріншоті, полягає у наданні інтерфейсу для модифікації вмісту існуючого тесту на рівні окремих запитань, дозволяючи редагувати текст запитань, змінювати варіанти відповідей та вказувати правильний варіант.

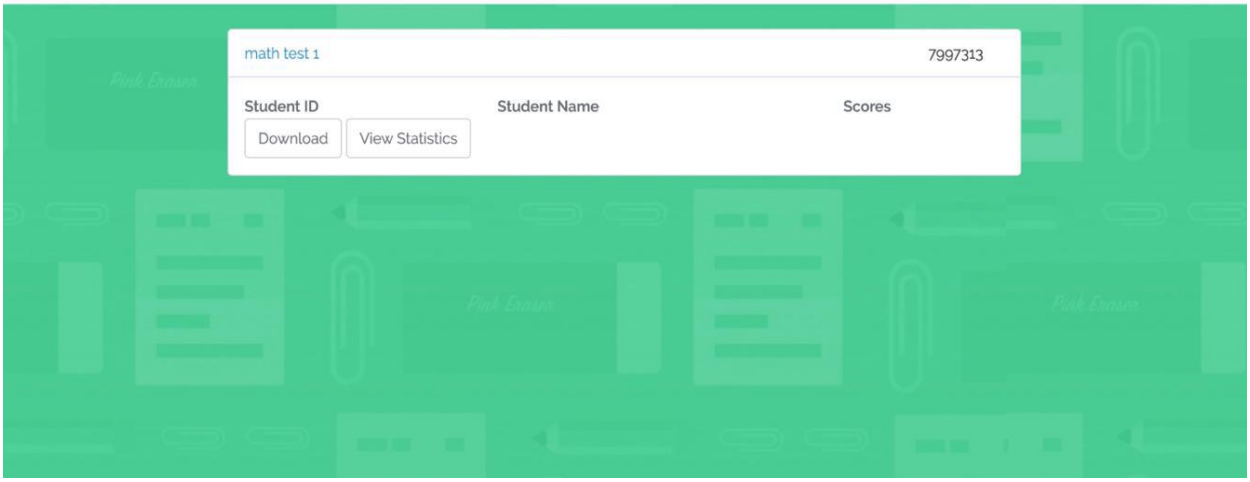

Рисунок 3.8 – Сторінка для відображення оцінок

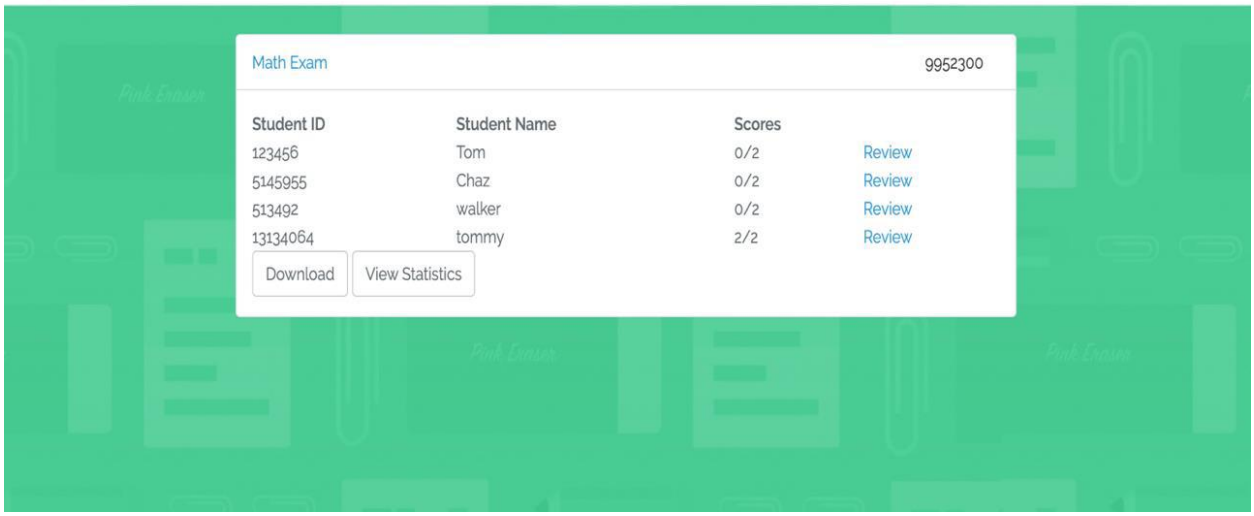

Рисунок 3.9 – Сторінка з оцінками студентів

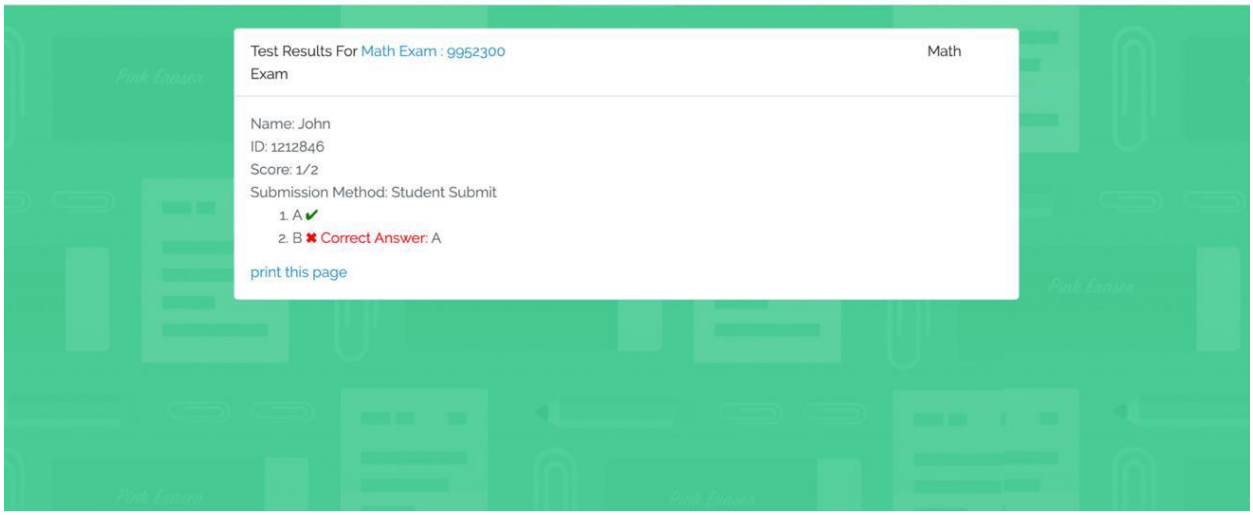

Рисунок 3.10 - Сторінка перегляду відповіді студента

На рисунку 3.11 представлено інтерфейс користувача для відображення статистичних даних за результатами конкретного екзамену. Екран має заголовок "Statistics For Math Exam" (Статистика для екзамену з математики) із зазначеним ідентифікатором або кодом екзамену (9952300).

На інтерфейсі представлено наступні статистичні показники, що характеризують результати проходження даного екзамену:

- Mean. Середнє значення балів.

- Median. Медіана балів.

- Mode. Мода балів (найчастіше зустрічається значення).

- Highest Score. Найвищий набраний бал.
- Lowest Score. Найнижчий набраний бал.
- Range. Діапазон балів (різниця між найвищим та найнижчим балом).

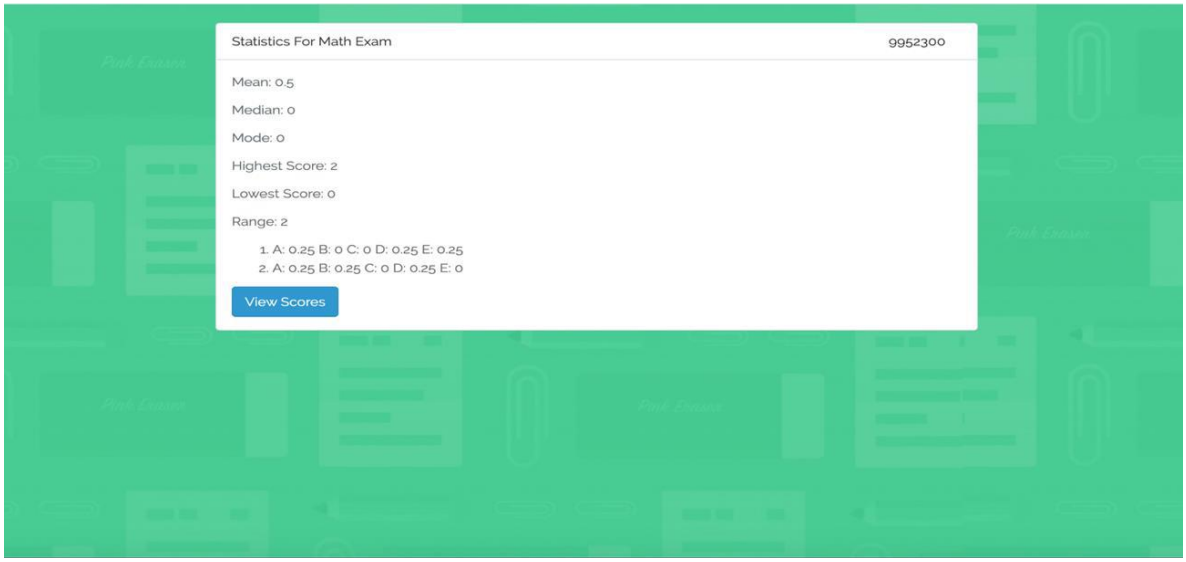

Рисунок 3.11 – Сторінка статистики

Нижче базових статистичних показників представлено аналіз відповідей на рівні окремих запитань. Формат "1. A: 0.25 B: 0 C: 0 D: 0.25 E: 0" і "2. A: 0.25 B: 0.25 C: 0 D: 0.25 E: 0" відображає частку або частоту вибору кожного з варіантів відповіді (A, B, C, D, E) для першого та другого запитань відповідно. Це надає деталізацію відповідей студентів на кожне запитання.

Внизу екрана присутня кнопка "View Scores" (Переглянути оцінки), яка, ймовірно, дозволяє перейти до більш детального перегляду результатів, можливо, з можливістю побачити індивідуальні бали кожного студента або повніші дані.

Таким чином, функціональність, показана на цьому скріншоті, полягає у наданні інструменту аналізу результатів екзамену, пропонуючи як зведені статистичні показники, так і деталі відповідей на рівні окремих запитань, що є корисним для викладачів для оцінки складності запитань та загальної успішності групи

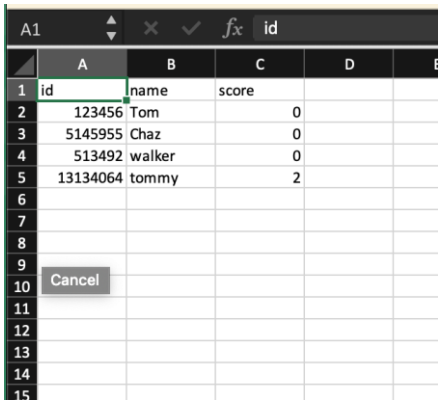

Рисунок 3.12 - Статистика у форматі .CSV (Excel)

Система надає можливість експорту результатів оцінювання у структурованому форматі даних. Згенерований серверною частиною файл у форматі .csv містить сукупні результати за конкретний екзамен. Даний формат файлу є сумісним з поширеними системами управління навчанням (наприклад, Blackboard), що дозволяє викладачам безпосередньо імпортувати результати оцінювання. Високий рівень довіри до цілісності отриманих даних забезпечується завдяки реалізації у клієнтських застосунках різноманітних механізмів протидії недоброчесності, що унеможливлюють або фіксують спроби обману під час тестування. Додатковою мірою забезпечення унікальності умов тестування та запобігання копіюванню є автоматизоване рандомізоване перемішування послідовності запитань та варіантів відповідей для кожного студента.

Отже, в рамках виконання даного проєкту було здобуто практичний досвід роботи з хмарними серверами та реалізації розподілених систем, включаючи розгортання та управління екземплярами хмарної системи тестування знань. Проєкт дозволив поглибити розуміння хмарних обчислень як фундаментальної технології, що стимулює розвиток сучасних систем. Набуті знання щодо внутрішньої організації та функціонування хмарних середовищ можуть бути застосовані для підвищення ефективності майбутніх розробок у сфері інформаційних технологій.

Окрему цінність становить досвід розробки нативних мобільних застосунків. Реалізація клієнтської частини системи для платформ Android та iOS сприяла поглибленню знань мов програмування Java та Swift відповідно, а також специфіки роботи з відповідними комплектами розробки програмного забезпечення (SDK).

В процесі розробки серверної частини було набуто досвід роботи з мовою програмування PHP та популярним фреймворком Laravel. Також застосовувалися стандартні веб-технології, такі як JavaScript та HTML, що є типовими для розробки веб-орієнтованих систем.

Досвід проектування та реалізації структури бази даних з використанням MySQL був розширений, включаючи практику застосування RESTful протоколів для організації взаємодії серверної частини з мобільними клієнтськими пристроями. Ця взаємодія передбачає використання стандартних методів HTTP, зокрема GET, POST, PUT та DELETE.

Загалом, архітектура системи розроблялася та функціонує відповідно до парадигми сервіс-орієнтованої програмної інженерії (Service-Orientated Software Engineering, SOSE), що є актуальним підходом у сучасній розробці розподілених систем на основі сервісів. Досвід, отриманий в рамках проєкту, охоплює ключові аспекти проектування, розробки та розгортання хмарно-базованих застосунків.

3.6. Подальші напрямки розвитку системи

Наступні передбачувані кроки в розвитку хмарної системи тестування знань передбачають додавання та удосконалення механізмів протидії недоброчесності в клієнтських мобільних застосунках для платформ Android та Apple iOS.

Також розглядається можливість інтеграції функції обліку відвідуваності студентів. Це включатиме верифікацію їхнього фізичного

розташування під час складання екзамену з метою підтвердження проходження тестування в аудиторії або іншому визначеному місці, що є важливим аспектом забезпечення об'єктивності оцінювання.

Система позиціонується як конкурентоспроможний продукт на ринку засобів оцінювання (порівняно з існуючими аналогами, такими як продукти бренду iClicker), і її подальший розвиток передбачає імплементацію функціональних можливостей, що перевершують пропозиції конкурентів, для підвищення її привабливості та ефективності.

Одним із напрямків оптимізації є дослідження альтернативних методів рандомізованого перемішування запитань та відповідей тестів. Поточна реалізація передбачає змішування даних "на льоту" під час обробки запиту студента на сервері. Альтернативний підхід полягає у попередньому змішуванні варіантів тестів та їх зберіганні на сервері. Необхідно провести порівняльний аналіз цих двох підходів з точки зору ресурсних витрат. Зберігання попередньо змішаних варіантів тестів потребуватиме більшого обсягу дискового простору на сервері (який, як правило, є відносно недорогим ресурсом). Натомість, змішування "на льоту" вимагає значних обсягів оперативної пам'яті під час виконання, хоча поточна оптимізація системи забезпечує низький рівень її використання. Оцінка ефективності обох реалізацій потребує моніторингу відповідних метрик продуктивності та споживання ресурсів, доступних на панелі керування хмарного провайдера (наприклад, Digital Ocean), для обґрунтованого вибору оптимальної стратегії.

Висновки до розділу

У третьому розділі було безпосередньо реалізовано хмарну систему тестування знань, що передбачає завершення етапу від концепції до робочого програмного продукту. Детально розглянуто архітектуру застосунку, яка

забезпечує логічну структурованість, гнучкість розгортання та стабільну роботу в хмарному середовищі.

Здійснено проектування архітектурного дизайну та побудову реляційної бази даних, яка підтримує зберігання тестів, результатів, облікових записів та іншої важливої інформації з урахуванням цілісності та оптимізації запитів. Розроблено інтерфейси користувача як для студентів, так і для викладачів — інтуїтивно зрозумілі, адаптивні та орієнтовані на зручність користування.

Окрему увагу приділено напрямкам подальшого розвитку системи, зокрема розширенню функціоналу, підвищенню рівня безпеки, впровадженню аналітичних модулів та мобільної підтримки. Таким чином, у цьому розділі завершено практичну реалізацію хмарної системи тестування знань, що готова до впровадження та масштабування в освітніх середовищах.

ВИСНОВКИ

В дипломній роботі було комплексно проаналізовано, спроєктовано та реалізовано хмарну інформаційну систему тестування знань, що відповідає сучасним вимогам до гнучкості, масштабованості та безпеки освітніх технологій.

У першому розділі проведено аналіз предметної області хмарних обчислень, визначено їхні переваги порівняно з класичними розподіленими системами, а також розглянуто архітектурні принципи побудови хмарних сервісів. Окрему увагу приділено протоколу RESTful як основі для побудови взаємодії клієнта і сервера у хмарному середовищі. Отримані результати створили теоретичне підґрунтя для розробки системи.

У другому розділі розглянуто архітектурні та алгоритмічні основи розробки системи. Здійснено порівняльний аналіз сучасних хмарних рішень із традиційними підходами, сформульовано вимоги до функціональності, спроєктовано Use Case діаграми, інтерфейси користувача та програмного забезпечення, а також охарактеризовано основні параметри безпеки та надійності системи. Це дозволило закласти структуровану та ефективну основу для реалізації програмного продукту.

У третьому розділі реалізовано безпосередню імплементацію системи, що охоплює проєктування архітектури додатку, бази даних, інтерфейсів користувачів для різних ролей (студента та викладача) та розробку функціонального прототипу. Представлено можливі напрями розвитку системи, що забезпечує потенціал для її масштабування та удосконалення у майбутньому.

Здійснено проектування архітектурного дизайну та побудову реляційної бази даних, яка підтримує зберігання тестів, результатів, облікових записів та іншої важливої інформації з урахуванням цілісності та оптимізації запитів. Розроблено інтерфейси користувача як для студентів, так

і для викладачів — інтуїтивно зрозумілі, адаптивні та орієнтовані на зручність користування.

Загалом, виконана робота продемонструвала ефективність застосування хмарних технологій у сфері освітніх сервісів, а розроблена система відповідає вимогам сучасного програмного забезпечення щодо доступності, продуктивності та захисту даних.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. R. Buyya, J. Broberg, and A. M. Goscinski, Cloud Computing: Principles and Paradigms. Wiley, 2011.
2. S. Newman, Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015.
3. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design, 5th ed. Addison-Wesley, 2011.
4. M. Kleppmann, Designing Data-Intensive Applications: The Essential Guide to Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017.
5. V. Kumar, Web Application Architecture: From Zero to Hero. Leanpub, 2018.
6. M. L. Abbott and M. T. Fisher, Scalability Rules: 50 Principles for Scaling Websites, Blogs, and Other Internet Properties. Addison-Wesley, 2011.
7. R. Puttini and F. Martins, Cloud Computing Architecture. Packt Publishing, 2013.
8. N. Murphy et al., Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media, 2016.
9. Amazon Web Services, AWS Well-Architected Framework. Amazon Web Services. Available: [https://aws.amazon.com/architecture/well-architected/].
10. Microsoft Azure, Azure Architecture Center. Microsoft Azure. Available: [https://docs.microsoft.com/en-us/azure/architecture/].
11. Google Cloud Platform, Google Cloud Architecture Center. Google Cloud Platform. Available: [https://cloud.google.com/architecture].
12. MySQL, MySQL Reference Manual. Available: [https://dev.mysql.com/doc/].
13. PostgreSQL Global Development Group, PostgreSQL Documentation. Available: [https://www.postgresql.org/docs/].

14. OWASP, OWASP Top 10. Available: [https://owasp.org/www-project-top-ten/].
15. D. Stuttard and M. Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2nd ed. Wiley, 2011.
16. R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd ed. Wiley, 2008.
17. N. Ferguson, B. Schneier, and T. Kohno, Cryptography Engineering: Design Principles and Practical Applications. Wiley, 2010.
18. Google, Android Developer Documentation. Available: [https://developer.android.com/docs].
19. Apple, iOS Developer Documentation. Available: [https://developer.apple.com/documentation/].
20. Django Software Foundation, Django Documentation. Available: [https://docs.djangoproject.com/].
21. M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed. Addison-Wesley, 2003.
22. A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 7th ed. McGraw-Hill, 2019.
23. Microsoft Azure, Designing and Building Security Into Your Cloud Applications. Microsoft Docs. Available: [https://www.google.com/search?q=https://docs.microsoft.com/en-us/azure/security/fundamentals/design-implement-security].
24. C. Wang et al., "Security and Privacy in Cloud Computing: A Survey," Int. J. Parallel Program., vol. 40, pp. 741-769, 2012. (Example of a survey paper)
25. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50–58. https://doi.org/10.1145/1721654.1721672

26. Mell, P., & Grance, T. (2011). The NIST definition of cloud computing (Special Publication 800-145). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145
27. Sultan, N. (2010). Cloud computing for education: A new dawn? International Journal of Information Management, 30(2), 109–116. https://doi.org/10.1016/j.ijinfomgt.2009.09.004
28. Al-Zoube, M. (2009). E-learning on the cloud. International Arab Journal of e-Technology, 1(2), 58–64. https://www.iajet.org/issue/vol1-no2/e-learning-on-the-cloud.pdf
29. Chandra, D., & Borah, M. D. (2012). Cost benefit analysis of cloud computing in education. 2012 International Conference on Computing, Communication and Applications, 1–6. https://doi.org/10.1109/ICCCA.2012.6179173
30. Masud, M. A. H., & Huang, X. (2012). An e-learning system architecture based on cloud computing. World Academy of Science, Engineering and Technology, 62, 74–78. https://doi.org/10.5281/zenodo.1057354
31. Jain, A., & Dutta, M. (2020). Cloud based education system using RESTful web services. International Journal of Innovative Technology and Exploring Engineering, 9(5), 3042–3046. https://doi.org/10.35940/ijitee.E2521.049520
32. Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7–18. https://doi.org/10.1007/s13174-010-0007-6
33. Rittinghouse, J. W., & Ransome, J. F. (2016). Cloud computing: Implementation, management, and security (2nd ed.). CRC Press.
34. Chowdhury, M. J. M., Colman, A., Kabir, M. A., Han, J., & Sarker, I. H. (2021). An overview of cloud computing technology. Journal of Network and Computer Applications, 174, 102887. https://doi.org/10.1016/j.jnca.2020.102887

35. Al-Janabi, S., & Al-Shourbaji, I. (2016). A study of cyber security awareness in educational environment in the Middle East. Journal of Information & Knowledge Management, 15(1), 1650001. https://doi.org/10.1142/S0219649216500011
36. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine). https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
37. Mahmood, Z. (Ed.). (2014). Cloud computing: Methods and practical approaches. Springer. https://doi.org/10.1007/978-1-4471-6452-6
38. Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2013). A survey of mobile cloud computing: Architecture, applications, and approaches. Wireless Communications and Mobile Computing, 13(18), 1587–1611. https://doi.org/10.1002/wcm.1203
39. Kuo, Y.-H. (2011). Opportunities and challenges of cloud computing to improve health care services. Journal of Medical Internet Research, 13(3), e67. https://doi.org/10.2196/jmir.1867

ДОДАТКИ

Додаток А

Фрагменти програмних кодів

Код контролера тесту для Android

```
public class TestViewActivity extends AppCompatActivity {
    // Тестові запитання та час
    int numberofQs = 0;
    int TestTime = 0;
    int SubmissionID = 0;
    String TestName = "";
    String TestCode = "";

    // json об'єкт для бази даних
    JSONObject jsonObject;
    boolean canSumbit = false;

    // таймери для виходу з додатка
    public long pausedTime;
    public long allowedTimeOutsideApp = 10000; // Це представлення часу в мілісекундах
    public int onResumeCounter = 0; // кількість разів, які ви залишили додаток
    public boolean exited = false;

    // Представлення для відключення панелі статусу
    private WindowManager manager;
    private WindowManager.LayoutParams localLayoutParams;
    private customViewGroup view;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        DisableStatusBar();

        setContentView(R.layout.activity_test);
        try {
            // Заповнення масивів
            // Кількість елементів TestViewActivity = змінна, яка була надіслана з сервера.
            // робить кількість запитань подвійною
            for (int i = 1; i <= numberofQs; i++) {
                if (i < 10) {
                    numbers.add("0" + String.valueOf(i) + ".");
                } else {
                    numbers.add(String.valueOf(i) + ".");
                }
                // заповнює масив, який буде надісланий, усіма 0
                submitted.add("0");
            }

            // передача аркуша через сторінку адаптера
            Intent sub = getIntent();
            sub.putStringArrayListExtra("sub", submitted);

            // ініціалізація списку
            scantron = (ListView) findViewById(R.id.listView);
```

```
        // Додавання заголовка вгорі списку
        header = new TextView(this);
        header.setBackgroundColor(Color.parseColor("#EF5D16"));
        header.setTextColor(Color.parseColor("#FFFFFF")); // Білий текст
        header.setTextSize(30);
        header.setGravity(Gravity.CENTER_VERTICAL | Gravity.CENTER_HORIZONTAL);
        scantron.addHeaderView(header);

        // встановлення назви тесту в TestViewActivity
        header.setText(TestName);

        scantron.addFooterView(submit);

        // Встановлення адаптера та заповнення рядків
        TestAdapter testadapter = new TestAdapter(this, R.layout.activity_test_layout, numb
        scantron.setAdapter(testadapter);

        // встановлення таймера відліку часу
        Log.i("Test Time", Integer.toString(TestTime));
        Log.i("Test Time", Integer.toString(TestTime * 1000));
        if (TestTime <= 86400) {
            new CountDownTimer(TestTime * 1000, 1000) {
                @Override
                public void onTick(long millisUntilFinished) {
                    if (millisUntilFinished % (60 * 1000) == 0) {
                        if (millisUntilFinished / (60 * 1000) < 5) {
                            header.setText(TestName + Long.toString(millisUntilFinished / (
                        }
                    }

                public void onFinish() {
                    for (int i = 0; i < submitted.size(); i++) {
                        studentAnswers += submitted.get(i);
                    }
                    try {
                        Submission("OutOFTime", studentAnswers, SubmissionID);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }.start();
        }

        // Встановлення кнопки для зміни тексту при натисканні та утриманні
        submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.i("STUDENT ANSWERS", studentAnswers);
                if (submit.getText() == "Submit Test") {
                    canSumbit = true;
                    int i = 1;
                    for (String answer : submitted) {
                        studentAnswers += answer;
                    }
                    try {
                        Submission("Student Submit", studentAnswers, SubmissionID);
                    } catch (JSONException e) {
                        e.printStackTrace();
```

```
                    }
                }
            }
        });

        submit.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                Log.i("SUBMIT", "Attempting to submit the test");
                // надіслав тест
                if (canSumbit) {
                    Log.i("Student Answers", studentAnswers);
                    try {
                        Submission("Student Submit", studentAnswers, SubmissionID);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
                return true;
            }
        });
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

// Встановлення таймера для відліку часу
public void onPause() {
    super.onPause();
    pausedTime = System.currentTimeMillis() + allowedTimeOutsideApp;
    Log.i("Time paused", String.valueOf(System.currentTimeMillis()));
    Log.i("Acceptable reentry time", String.valueOf(pausedTime));
    manager.removeView(view);
}

public void onResume() {
    super.onResume();
    // функція onResume викликається при запуску, що встановлює лічильник на 2, що дозволяє
    onResumeCounter++;
    if ((System.currentTimeMillis() > pausedTime) && (onResumeCounter > 1)) {
        exited = true;
        for (int i = 0; i < submitted.size(); i++) {
            studentAnswers += submitted.get(i);
        }
        try {
            Submission("Time Exited", studentAnswers, SubmissionID);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}

// Функція для відправки даних на сервер
public void Submission(final String method, final String answers, final int submissionID) {
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                URL url = new URL("https://yourserver.com/submit");
                HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
                conn.setRequestMethod("POST");
                conn.setRequestProperty("CONTENT_TYPE", "application/json;charset=UTF-8");
```

```
                    conn.setRequestProperty("ACCEPT", "application/json");
                    conn.setDoOutput(true);

                    String jsonInputString = "{\"method\":\"" + method + "\", \"answers\":\"" +

                    try (OutputStream os = conn.getOutputStream()) {
                        byte[] input = jsonInputString.getBytes("utf-8");
                        os.write(input, 0, input.length);
                    }

                    int code = conn.getResponseCode();
                    Log.i("Server Response Code", String.valueOf(code));
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
        thread.start();
    }
}
```

Код контролера сервера

```
if ($test->simpleT == "no") // Не простий тест, змішування порядку.
{
    Log::info("NOT A SIMPLE TEST. WILL SHUFFLE");
    array_multisort($OTF, $QTF, $ATF, $BTF, $CTF, $DTF, $ETF);
    $string = join($KTF);
    $test->answer_key = $string;
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Побудова хмарно-базованої системи тестування знань”

Обсяг пояснювальної записки: 76 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____