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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка програмної платформи для перевірки знань шляхом тестування

Виконав: студент 6 курсу, групи СНм-61
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)

(підпис) Гайдар А.В. (прізвище та ініціали)

Керівник (підпис) Готович В.А. (прізвище та ініціали)

Нормоконтроль (підпис) Мацюк О.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) Бойко І.В. (прізвище та ініціали)

Тернопіль
2021

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

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

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
« » 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

студенту Гайдару Андрію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмної платформи для перевірки знань шляхом тестування

Керівник роботи Готович Володимир Анатолійович, кандидат технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*28*» *10* 2021 року № *4/7–908*

2. Термін подання студентом завершеної роботи 06 грудня 2021

3. Вихідні дані до роботи Літературні джерела по темі дослідження

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналіз існуючих платформ для перевірки знань шляхом тестування. Вибір архітектури та програмних рішень. Проектування платформи для перевірки знань шляхом тестування. Охорона праці та безпека в надзвичайних ситуаціях. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Титульний слайд, актуальність теми, актуальність та мета кваліфікаційної роботи, практичне значення одержаних результатів, слайд з аналізом OpenTest, аналіз GoogleClassroom, аналіз Moodle, аналіз Indigo, аналіз ITuror, клієнт-серверна архітектура, архітектура серверного додатку, архітектура клієнтського додатку, огляд NodeJS, огляд NestJS, огляд PostgresSQL, огляд Angular, структура бази даних, JWT-токени, демонстрація інтерфейсу користувача, демонстрацією інтерфейсу адміністратора, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Дмитроца Л.П. доцент		
Безпека в надзвичайних ситуаціях	Клепчик В. М., ст. вик.		

7. Дата видачі завдання 27 вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи	27.09.2021-29.09.2021	*Виконано*
2	Підбір наукових джерел про	30.09.2021-03.10.2021	*Виконано*
3	Переклад та опрацювання наукових джерел	04.10.2021-10.10.2020	*Виконано*
4	Виконання дослідження щодо систем комп'ютерного тестування	11.10.2021-17.10.2021	*Виконано*
5	Оформлення розділу «Аналіз літературних джерел та відомих рішень»	18.10.2021-24.10.2021	*Виконано*
6	Оформлення розділу «Проектування архітектури платформи»	25.10.2021-31.10.2021	*Виконано*
7	Оформлення розділу «Реалізація платформи для перевірки знань шляхом тестування»	01.11.2021-07.11.2021	*Виконано*
8	Виконання завдання до підрозділу «Охорона праці»	08.11.2021-11.11.2021	*Виконано*
9	Виконання завдання до підрозділу «Безпека в	12.11.2021-14.11.2021	*Виконано*
	надзвичайних ситуаціях»		
10	Оформлення кваліфікаційної роботи	15.11.201-24.11.2021	*Виконано*
11	Нормоконтроль	25.11.2021-28.11.2021	*Виконано*
12	Перевірка на плагіат	29.11.2021	*Виконано*
13	Попередній захист кваліфікаційної роботи	07.12.2021	*Виконано*
14	Захист кваліфікаційної роботи	20.12.2021	

Студент _____ Гайдар А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка програмної платформи для перевірки знань шляхом тестування // Кваліфікаційна робота освітнього рівня «Магістр» // Гайдар Андрій Віталійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2021 // С. 71, рис. – 26, додат. – 4, бібліогр. – 50.

Ключові слова: КЛІЄНТ-СЕРВЕР, ПЛАТФОРМА, ТЕСТУВАННЯ, ANGULAR, БАЗА ДАНИХ.

Кваліфікаційна робота присв'ячена розробці платформи для перевірки знань шляхом тестування з веб-інтерфейсом користувача.

В першому розділі проведено аналіз відомих на сьогоднішній день систем контролю знань шляхом тестування та відповідних платформ для навчання, розглянуто їхні переваги та недоліки.

В другому розділі обгрунтовано вибір трирівневої клієнт-серверної архітектури для розробки платформи, архітектуру клієнтської та серверної її частин.

В третьому розділі спроектовано та реалізовано прототип платформи, зокрема, клієнтський та серверний додатки, базу даних, графічний веб-інтерфейс користувача.

ANNOTATION

Software platform development for knowledge control via testing // Thesis of educational level "Master" // Haidar Andrii Vitaliyovych // Ternopil National Technical University named after Ivan Pulyuy, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNm-61 group // Ternopil, 2021 // P. 71, fig. – 26, annexes – 4, references – 50.

Key words: CLIENT-SERVER, PLATFORM, TESTING, ANGULAR, DATABASE.

Thesis is dedicated to the development of a platform for testing knowledge via testing with a web user interface.

The first section analyzes the currently known knowledge control systems through testing and appropriate learning platforms, discusses their advantages and disadvantages.

The second section substantiates the choice of three-tier client-server architecture for the development of the platform, the architecture of client and server parts.

In the third section, a prototype of the platform is designed and implemented, in particular, client and server applications, database, graphical web user interface.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

ПЗ – програмне забезпечення.

СУБД – система управлінням базою даних.

SQL (англ. Structured query language) – мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних.

IDE (англ. Integrated Development Environment) – інтегроване середовище проектування та розробки.

HTTP (англ. Hyper Text Transfer Protocol) – протокол передачі гіпертексту.

CSS (англ. Cascading Style Sheets) – каскадна таблиця стилів, що використовується для стилізації веб-сторінок.

JSON (JavaScript Object Notation) – відкритий стандартний формат файлів і формат обміну даними, який використовує зрозумілий людині текст для зберігання та передачі об’єктів даних, що складаються з пар атрибут-значення та масивів (або інших значень, які можна серіалізувати). Це звичайний формат даних з різноманітним використанням в електронному обміні даними, включаючи веб-додатки із серверами.

ЗМІСТ

ВИСНОВКИ

ВСТУП

Актуальність теми. Одним із найважливіших завдань у повсякденній роботі вчителів є необхідність контролю знань осіб, які навчаються. Існує багато форм контролю, але найчастіше використовуються письмові або усні. На жаль, ці форми не позбавлені недоліків.

Тестування все частіше використовується як ефективний спосіб перевірки знань. Однією з його головних і незаперечних переваг є найменші витрати часу на отримання достовірних результатів. Для проведення тестування використовуються як паперова версія, так і електронна версія тесту. Останні особливо корисні, оскільки дозволяють отримати результати майже відразу після проведення тесту.

Актуальність даної роботи полягає в потребі розробки засобу для перевірки знань шляхом тестування, який би був позбавлений більшості недоліків відомих на сьогодні систем аналогічнго призначення.

Мета і задачі дослідження. Метою роботи є розробка клієнт-серверної платформи для перевірки знань шляхом тестування з графічним веб-інтерфейсом користувача. Для досягнення поставленої мети необхідно виконати такі завдання:

- проналізувати відомі на сьогоднішній рішення аналогічного призначення;
- обгрунтувати вибір архітектури платформи а також інструментальних засобів розробки;
- розробити прототип платформи для перевірки знань шляхом тестування.

Об’єкт дослідження автоматизація процесів організації та оцінювання рівня засвоєння знань особами, які навчаються.

Предмет дослідження засіб автоматизації процесів організації та оцінювання рівня засвоєння знань у вигляді веб-орієнтованої платформи.

Наукова новизна одержаних результатів. Результатом виконання роботи є прототип платформи для перевірки знань шляхом тестування, яка усуває важливі недоліки відомих систем аналогічного призначення.

Практичне значення одержаних результатів. Виконано прототипування програмної платформи для перевірки знань шляхом тестування.

Апробація результатів кваліфікаційної роботи. Окремі результати роботи було представлено на двох наукових конференціях:

1) X Міжнародна науково-технічна конференція молодих учених та студентів «Актуальні задачі сучасних технологій» на тему “Застосування комп’ютерно-інформаційних засобів в процесі навчання”;

2) IX науково-технічна конференція «Інформаційні моделі, системи та технології» на тему “Розробка платформи для перевірки знань шляхом тестування”.

Публікації. Основні результати кваліфікаційної роботи опубліковано у двох працях конференції (див. додатки А та Б).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку літератури з 50 найменувань та 4 додатків. Загальний обсяг кваліфікаційної роботи складає 71 сторінку, з них 58 сторінок основного тексту, який містить 26 рисунків.

1 АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ТА ВІДОМИХ РІШЕНЬ

1.1 Аналіз відомих платформ для перевірки знань шляхом тестування

В даний час існує багато систем, призначених для автоматизації перевірки знань. Розглянемо платформи, які враховують особливосьі вітчизняної освіти. Будуть розглянуті три параметри: можливість роботи через Інтернет; платформа та тип ліцензії. Інші параметри схожі в більшості систем.

Комп'ютерна система тестування знань дистанційного навчання OpenTEST 2.02.0 – це комп'ютерна система перевірки знань, призначена для особистого контролю теоретичного збору даних, засвоєння знань та практичних навичок студентів у великих організаціях, таких як підприємства зі складною розподіленою структурою. Ця система дистанційного тестування знань студентів побудована на основі мережевих технологій і може працювати у мережі компанії (інтранет) та глобальній мережі Інтернет. Використання та управління системою OPENTEST повністю реалізується за допомогою веб - інтерфейсу, який має багато суттєвих переваг:

- використовувати гіпертекст (надання посилань на різну інформацію);
- браузер сервера або клієнтська програма (забезпечує єдиний, досить простий, швидкий і звичний інтерфейс користувача); інтерфейс (дозволяє віддалене управління).

Основною особливістю системи OpenTEST 2.0 є зосередження на наданні найсуворіших звітів для тестів учнів. При створенні OPENTEST 2.0 були використані такі програмні продукти:

- мова сценарію, реалізована в HTML;

- невелика та швидка реляційні СКБД. ЇЇ переваги: багатопоточність, підтримка декількох одночасних запитів, записи фіксованої та змінної довжини;
- найпоширеніший веб-сервер у світі [1].

OpenTest – це проект з відкритим кодом, тому можуть розповсюджуватися безкоштовно. Він є багатоплатформенним, тобто також може нормально працювати під Windows, UNIX та іншими операційними системами. Структура тестової системи є модульною, що дозволяє підключати модулі з додатковими функціями, змінювати режим роботи окремого модуля та керувати налаштуваннями системи, не впливаючи на інші.

Система OpenTEST 2.0 приділяє велику увагу питанням безпеки під час процесу тестування. Оскільки остаточна оцінка відіграє величезну роль у контрольному тесті, об’єктивність її твердження має бути найбільшою, а всі можливі результати фальсифікації слід виключити. З цією метою система OpenTEST 2.0 розробила унікальний програмний метод для забезпечення безпеки комп'ютерного тестування:

- тимчасове блокування облікового запису, блокування IP адреси при спробі злому паролів до всіх модулів системи;
- унікальний алгоритм динамічного зміни хеш-ідентифікатора клієнта, що в свою чергу не дозволяє одночасну роботу під одним аккаунтом з декількох комп'ютерів і запобігає будь-які несанкціоновані спроби використання даних на проміжних ланках мережі;
- протокол SSL для захищеної передачі даних;
- структуроване логування подій;
- детальна статистика логів подій за певними критеріями;
- щоденний звіт зібраний з логів ;

Паролі шифруються на стороні клієнта, це не дозволяє перехопити чийсь пароль при роботі з системою OpenTEST 2.0 навіть без протоколу SSL.

Переваги OpenTest:

- гнучкий редактор тестів;
- кросплатформеність;
- дистанційне тестування;
- низькі вимоги до апаратних і програмних ресурсів;
- розширення функціональності без необхідності внесення змін в існуючі модулі;

Недоліки:

- незручний в роботі інтерфейс;
- надмірний функціонал [2].

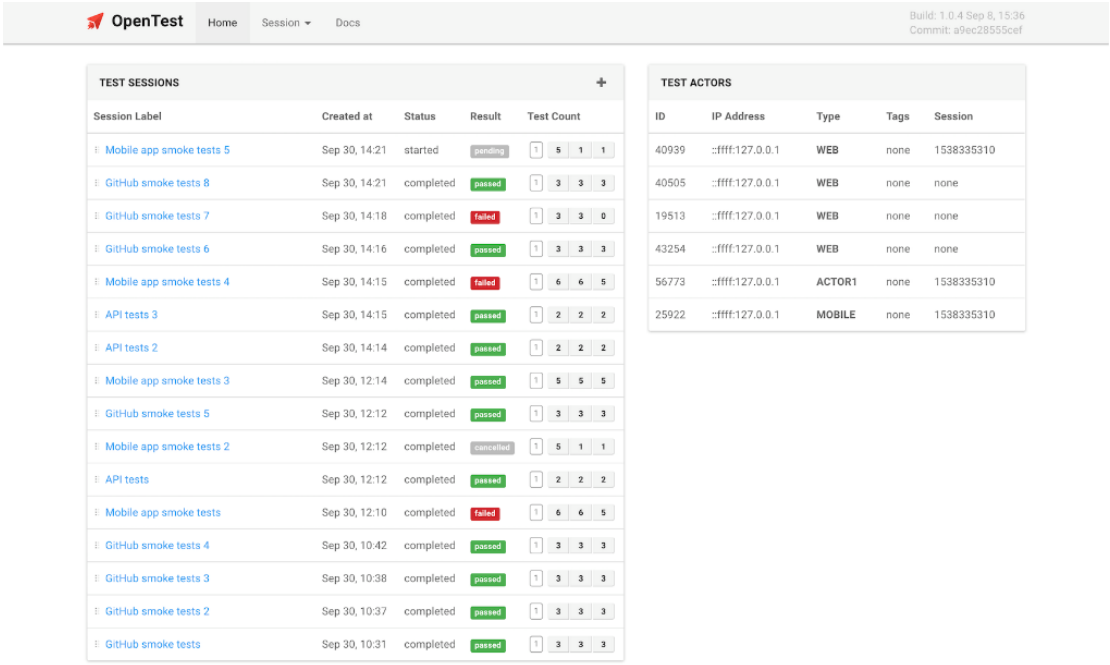

Рисунок 1.1 – Графічний інтерфейс програми OpenTest

Google Classroom — це інструмент, який з'єднує Документи Google, Google Диск і Gmail. Він може допомогти створювати й упорядковувати завдання, оцінки, коментарі та організовувати ефективне спілкування зі студентами в режимі реального часу. Основним елементом Google Classroom є група. Форуми з функціональною груповою структурою, оскільки вони дозволяють користувачам легко надсилати повідомлення іншим

користувачам, з якими вони часто спілкуються в групі. Тому що ця платформа вбирає в себе всі необхідні функції та процедури групової взаємодії між студентами [3].

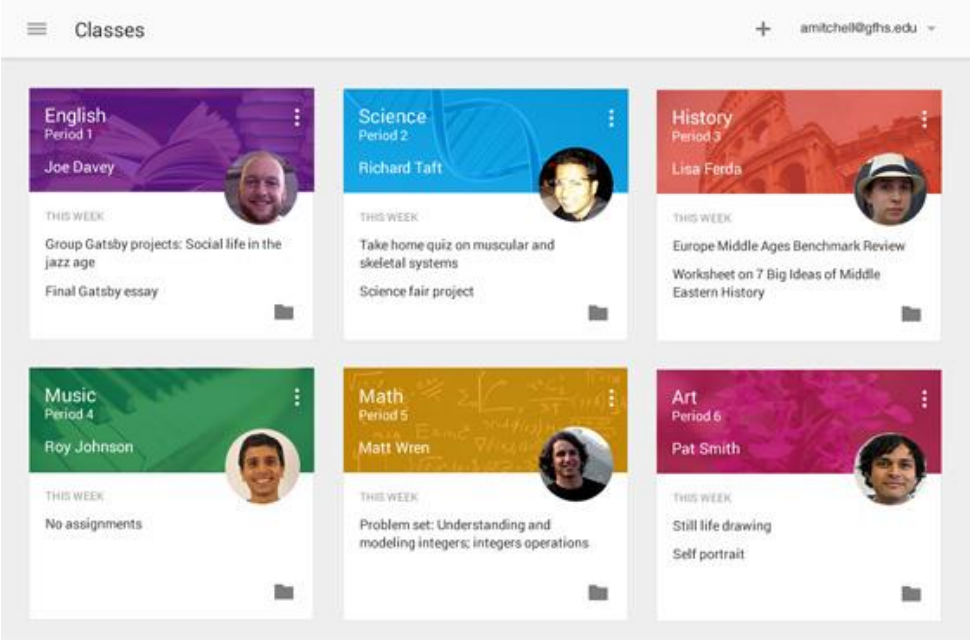

Рисунок 1.2 – Графічний інтерфейс Google Classroom

Функції програми прості у використанні і забезпечують наступні можливості:

- створити окремі класи для кожної окремої групи учнів;
- створіть оголошення для однієї або кількох груп;
- створення завдань з можливістю прикріплення посилань, відео- та аудіо-контенту (в тому числі YouTube), файлів з різними типами, а також збереження файлів на Google «диску»; установка термінів складання кожного конкретного завдання з точністю до хвилини;
- графа виставлення оцінок за виконання завдання яка може гнучно налаштовуватися для кожного конкретного завдання;
- можливість редагування і коментування з даних учням і завдань з динамічним відображенням поправок в режимі реального часу [4].

Переваги Google Classroom:

- легко інтегрується з іншими сервісами;
- розповсюджується з безкзоштовною ліцензією;
- підтримка української мови.

Недоліки:

- немає перевірки на “чесність” складеного тесту;
- обов’язкова наявність електронної пошти Gmail;
- незручний інтерфейс користувача [5].

Moodle – одна з найбільш популярних систем управління електронним освітою. Зараз в світі налічується більше мільйона ресурсів, які працюють з нею. До можливостей системи придивляються і компанії для навчання своїх співробітників – в першу чергу через її безкоштовну модель роботи. Moodle знаходиться у відкритому доступі: її можна завантажити з офіційного сайту і встановити на свій комп'ютер. Система підтримує більше 120 мов, в тому числі українську.

Установчий пакет Moodle складається з трьох елементів:

- код Moodle, який завантажується на веб-сервер;
- база даних, керована MySQL, PostgreSQL, Microsoft SQL Server, MariaDB або Oracle;
- мховище для завантажених і згенерованих файлів.

Всі три частини можуть працювати як на одному сервері, так і на декількох для розподілу навантаження.

Moodle відрізняється гнучкою схемою наповнення. У неї можна завантажувати прості схеми на кшталт файлів і папок або більш складні: розділи Wiki, глосарії (можуть заповнювати учні), завдання, різні форми тестування [6].

Найголовніша перевага яку надає платформа, – безкоштовне використання Moodle. Платформа працює за схемою – відкритого вихідного коду. За рахунок цього, велика кількість програмістів регулярно створює нові

розширення і модулі в середовищі Moodle – зараз таких є близько 1500. Наприклад:

- відеоконференції;
- модуль аудіо- та відеочати;
- плагін для управлінням навчальних матеріалів та конвертація в PDF-файл;
- мотиваційні модулі;
- інтерактивні звіти які формуються за навчальним планом;
- модуль розсилки повідомлень;
- портфоліо.

Завдяки безкоштовній версії і гнучкості налаштувань Moodle активно використовується в вузах і коледжах, а також репетиторами для персонального навчання і створення дистанційних курсів.

Навчальний курс в Moodle, який створюється через вбудований редактор, може складатися з декількох елементів. Наприклад, найпростіший і найпоширеніший - це лекції, завдання і тести. У систему завантажується навчальний контент, з якого формуються лекції. Це можуть бути текстові файли у форматі PDF, XLS, а також відео, фото, презентації та аудіо [7].

Переваги:

- головна перевага системи Moodle – безкоштовне розповсюдження. Це означає, що при її самостійному впровадженні витрати дорівнюють нулю. Оплачувати ліцензію або місячну «підписку» не потрібно.
- можливість адаптації під конкретні цілі та завдання. Система має відкритий системний код, тому її легко змінювати за допомогою плагінів.
- свобода вибору. Можна встановити систему як на сервер, так і на локальний ПК викладача, тоді Moodle буде доступний тільки в корпоративній мережі.

- платформа підтримує міжнародні стандарти обміну навчальними матеріалами: SCORM і AICC. Завдяки цьому можна впроваджувати в Moodle електронні курси від різних розробників;
- має безліч інструментів для створення електронних курсів;
- можливість розробки в “корпоративному” стилі;

Недоліки:

- головним недоліком плафтформи є відсутність технічної підтримки. Справа в тому, що над кодом працюють безліч програмістів з всього світу, і комунікація між учасниками розробки ускладнена;
- Moodle вимогливий до сервера. Безкоштовний хостинг дозволяє встановлювати тільки старі версії системи. До того ж сервіс споживає чимало ресурсів, що збільшує фінансові витрати;
- незважаючи на часті оновлення, у Moodle досить застарілий, та не дружній до користувача інтерфейс. Для того аби зробити управління курсами зручнішим - доведеться окремо налаштовувати всю систему;
- Moodle – велика система з великою кількістю функцій, частина з яких часто не використовується. А прибрати їх можна тільки людина з навичками веб-програмування, і то не всі;
- інструменти аналітики є, але вони не дають великої глибини і охоплення для задач бізнесу [8].

Рисунок 1.3 – Графічний інтерфейс Moodle

Indigo – являє собою повнофункціональне програмне забезпечення, що автоматизує тестування і видачу результатів. Продукт був розроблений в 2010 році. Система INDIGO – це універсальний інструмент, який можна використовувати для вирішення різноманітних завдань:

- перевірка та контроль знань учнів з різних предметів;
- визначати професійний рівень працівників (у тому числі при підборі на роботу);
- автоматизація психологічних тестів, у тому числі з профорієнтації (відбір кар'єри);
- проведення опитування (соціологія, маркетинг, визначення домінуючих думок тощо);
- автоматизація вікторин та конкурсів.

Використання INDIGO поділяється на дві частини: інтерфейс адміністратора та інтерфейс користувача.

Інтерфейс Адміністратора є додатком який працює в операційній системі Windows, який реалізує наступні функції:

- редагування та створення тестів;
- управління базою тестів;
- управління базою користувачів;
- управління Web-сервером і правилами тестування;
- доступ до результатів тестів.

Інтерфейс користувача тестової оболонки являє собою web-інтерфейс, які реалізує наступні функції:

- реєстрація та авторизація користувачів в системі;
- перегляд доступних тестів;
- вибір тесту і проведення тестування;
- перегляд результатів тестування і помилок;
- доступ до журналу результатів [9].

Основні переваги:

- проста інсталяція;
- зручний інтерфейс користувача;
- підтримка всього сімейства Windows (XP / 2003 / Vista / 7/8, 10); та всіх популярних браузерів; web-інтерфейс користувачів;
- ієрархічне групування тестів і користувачів (правила тестування);
- гнучкі можливості конструктора тестів.

Недоліки:

- платна ліцензія;
- немає розділення обов'язків між адміністратором та викладачем (не завжди викладач має навички роботи з подібного роду системами);
- відсутність перевірки на "чесність" проходження тесту;
- система споживає відносно багато пам'яті;
- високі системні вимоги[10].

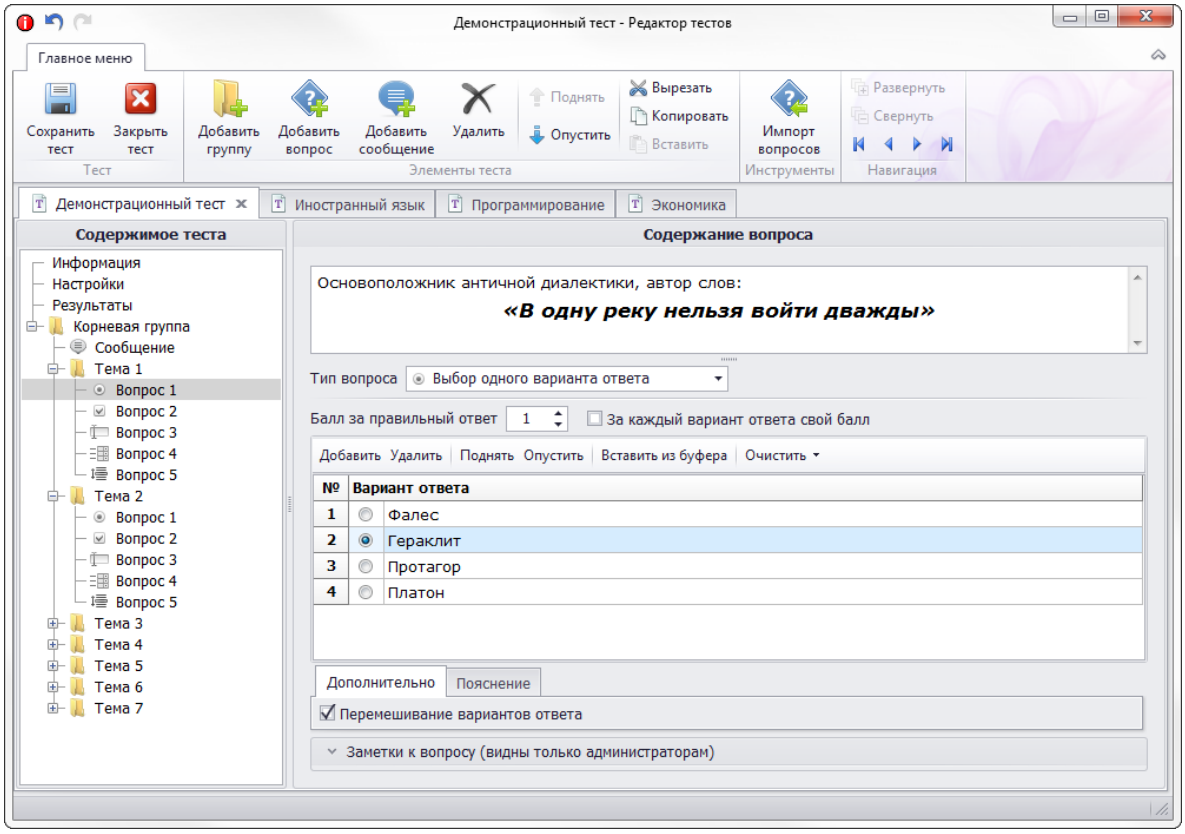

Рисунок 1.4 – Графічний інтерфейс Indigo

ATutor – відноситься до систем керування навчальним матеріалом LCMS (Learning Course Management System) та систем керування навчанням LMS (Learning Management System). Програма є зручною у користуванні та відрізняється від аналогів простим і ефективно структурованим інтерфейсом. Українізацію системи було проведено в ТНТУ авторами даних методичних вказівок. Користувачі, які працюють із системою ATutor, можуть мати два види прав доступу дої ресурсів: права студента та права інструктора (викладача/лектора). Після реєстрації та активації профілю користувач системи отримує права студента. Для отримання прав викладача потрібно через інтерфейс натиснути на закладці “Запит прав інструктора” в розділу “Мої курси”, де потрібно буде коротко описати курси які користувач планує створити. Лист розгляне адміністратор системи та вирішить, чи варто надавати права доступу.

Після отримання прав викладача, користувач може створити курс. Курс має наступні властивості:

- назва предмету відповідно до навчального плану чи робочої програми.
- основна мова. Мова, яка за замовчуванням буде використовуватися для відображення елементів інтерфейсу;
- опис. Шифри і назви напрямів (спеціальностей), для яких призначено предмет;
- категорії;
- експорт матеріалу. Дозволяє студентам завантажити матеріали курсу на свій комп’ютер і потім користуватися ними у форматі електронного підручника без виходу в Інтернет
- розсилання оголошень. Дає можливість розсилати оголошення, які публікуються в курсі, через систему RSS (Rich Site Summary);
- доступ. За рівнем доступу курси поділяються на: відкриті(можуть переглядати навіть незареєстровані користувачі), захищені(входити в систему

обов’язково, а записуватись на курс – ні.), закриті (після підтвердження автора курсу);

- дата публікації. Дозволяє встановити дату початку доступу матеріалів для інших користувачів.

Створення тестів в системі ATutor також дуже гнучне. Запитання створюються окремо, без долучення до конкретного тесту. Вони можуть використовуватись у декількох тестах одночасно. Всі запитання курсу зберігаються у “Базі даних питань”. Система дозволяє створити такі типи запитань:

- запитання з множиною варіантів (у якому є декілька варіантів відповіді (до 10), при цьому як правильну можна вказати тільки одну відповідь);
- запитання з множинним вибором (у якому є декілька варіантів відповіді (до 10), при цьому як правильні можуть бути декілька відповідей);

альтернативне запитання (у якому є лише два варіанти відповіді “Істина” та “Неправда”);

- відкрите запитання (на яке студент повинен сам написати відповідь, при цьому обсяг відповіді може бути таким: одне слово, одне речення, невеликий абзац та одна сторінка);

Створення запитання показано на рис. 1.5.

При створенні тесту потрібно заповнити такі атрибути:

- “Назва” – вказує назву тесту. Саме цю назву будуть бачити студенти на їхній сторінці тестування;
- “Дозволяється спроб” – визначає кількість спроб здавання тесту для одного студента. Можливі значення – необмежено та від 1 спроби до 20;
- “Додати посилання зі сторінки Мої курси” – додає посилання на даний тест на сторінці “Мої курси”. Це може бути корисним аби студенти звернули увагу на наявність тестів у електронному курсі ще перед входом у нього. Тому рекомендоване значення цієї опції “Так”;

- “Анонімний”. При активації цієї опції здавання тесту не оцінюються і не фіксуються по користувачах. Корисна при створенні анонімних опитувань. Для проведення навчальних тестувань потрібно вказувати “Ні”;
- “Публікувати результати” – визначає доступність для студентів результатів проходження тестів. Якщо вказати у цій опції “Після проходження тесту”, то студент отримає результати одразу після завершення тестування;
- “Показувати” – дає можливість вибрати формат виводу білета: всі запитання на сторінці або ж одне запитання на сторінку (рекомендується);
- “Вибирати запитання випадково” – визначає, чи вибирати запитання випадково для тесту з бази даних запитань і яку їх кількість;
- “Дата початку” та “Дата закінчення” – визначає дату початку та закінчення тестування. Протягом цього періоду тест буде доступний для проходження;
- “Загальна тривалість” – задає граничну тривалість проходження тесту і може становити від 1 до 180 хвилин, а також бути необмеженою.

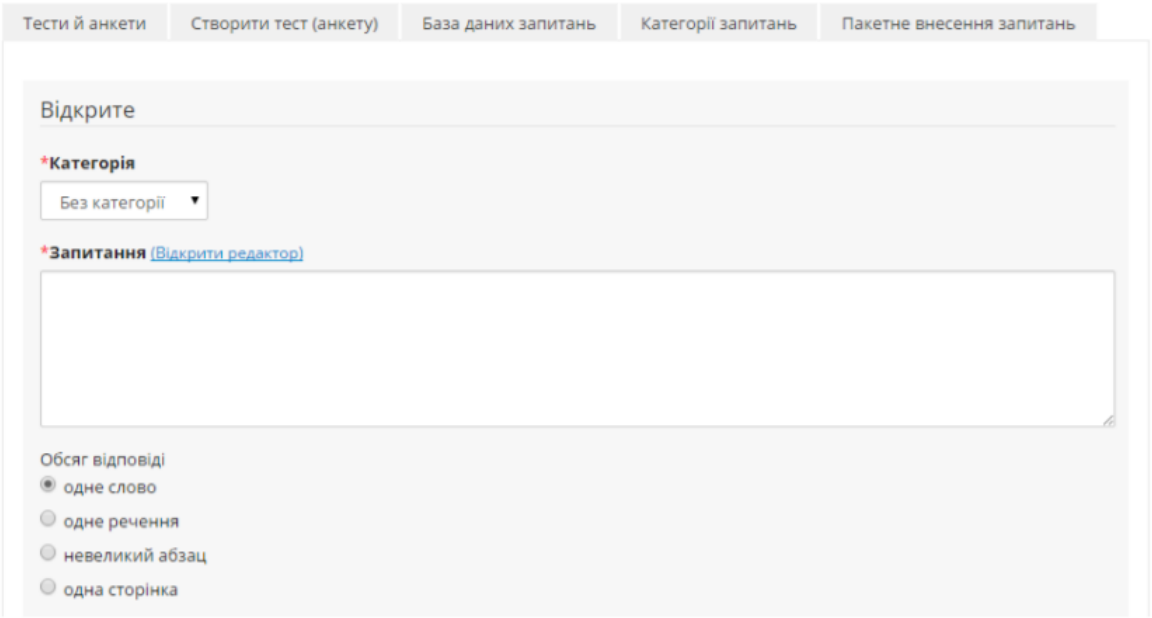

Рисунок 1.5 – Сторінка створення запитання

Також система містить статистику: по кожному студенту, групі, предмету та загальну, легко інтрегрується з іншими системами (Skype, BigBlueButton), скриньку завдань, журнал та залікову книжку тощо. Інтерфейс користувача показаний на рис. 1.6.

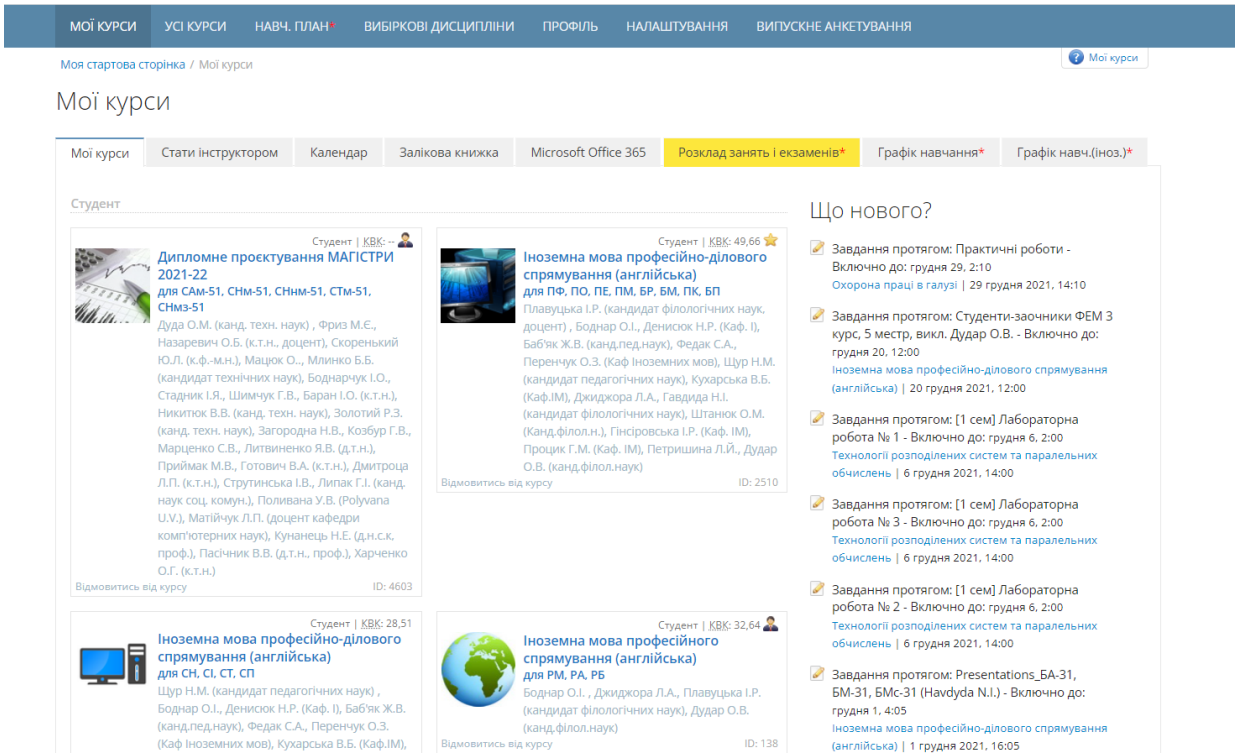

Рисунок 1.6 – Інтерфейс коритсувача системи ATutor

Якщо узагальнити то до плюсів можна віднести:

- гнучний редактор тестів
- обширна статистика
- зручний поділ на ролі студент/викладач
- інтеграцію з іншими системами.

Але незважаючи на це система має і недоліки:

- дещо “застарілий” інтерфейс;
- перевірку на неактивність можна обманути (вимкненням Javascript у браузері) [11].

1.2 Постановка завдання

Одним із способів удосконалення навчального процесу є розробка системи моніторингу знань, умінь та навичок, що дозволяє об’єктивно оцінювати знання. В даний час існує багато різних методів моніторингу та оцінки знань.

Найбільш широко застосовується тестування як один з методів контролю засвоєння учнями знань з дисципліни, що має низку певних переваг перед традиційними методами контролю знань (контрольна робота, усну відповідь тощо). Інструментом для вимірювання за шкалою досягнень учня є правильно сконструйований тест, який відповідає не тільки предмету навчання, але і його завданням і служить розвитку системного підходу до вивчення навчальної дисципліни [12].

Тестування в освітніх установах використовується, як засіб об'єктивного контролю знань навчальної програми в учнів. Систематичний контроль рівня знань учнів з одного боку визначає успішність учня, а з іншого боку є показником ефективності методики навчання і організації навчального процесу. Включення різних форм тестових завдань в процес навчання мотивує учнів до активізації роботи по засвоєнню навчального матеріалу і формує прагнення розвивати свої здібності. На даний момент в навчальних закладах використовують два види тестування комп'ютерне і у вигляді опитувальних листів.

Тестування відбувається максимально просто і зручно, як і для автора тесту, тому що полегшує контроль та надає статистику по проходженню, так для того хто тест проходить. Під час тесту людина бачить питання, варіанти відповідей, час завершення тесту та після його проходження результат.

Комп'ютерне тестування має ряд переваг:

- швидке формування результатів випробування;

- звільнення викладача від трудомісткої роботи з обробки результатів тестування;
- об'єктивна в оцінці;
- економія трудовитрат.

Щоб провести тестування у вигляді опитувальних листів викладачеві необхідно:

- розпечатати всі варіанти тестів на папері;
- забезпечити навчаються листками з варіантами тестів;
- зафіксована на паперовому носії або в електронній записній книжці всі прізвища і варіанти, які їм належать;
- перевірити кожен тест і позначити помилки, які були допущені;
- зафіксована на паперовому носії або в електронній записній книжці оцінки [13].

Якщо порівняти процедуру проведення тестування у вигляді опитувальних листів і комп'ютерне, то очевидно, що комп'ютерне тестування є більш ефективним .З цього випливає, що розробка web-плафтформи для комп'ютерного тестування, яке знизить витрати як фінансові, так і трудові покладені на викладача, є розумним рішенням [14]. При роботі з системою людина яка проходить тестування повинна мати можливість:

- доступу через Інтернет до тестової системи під своїм обліковим записом;
- відповідати на питання тесту в зручній для нього послідовності;
- переглянути результат пройденого тесту [15].

1.3 Формулювання вимог до платформи

Платформа повинна працювати стабільно, тобто не вилітати, не містити помилок які заважають роботі програми, або тих програма працює некоректно, не містити вразливостей які загрожують програмним взломом системи.

Архітектрура програмного забезпечення повинна забезпечувати гарну масштабованість, тобто при збільшенні навантаження на систему (приплив користувачів) або добавлення нового функціоналу повинно відбуватися без зайвих зусиль.

1.4 Висновки до першого розділу

В даному розділі було проведено загальний аналіз систем, їхні осноні переваги та недоліки. Роглянуто особливості тестування його важливість у навчальному процесі, переваги тестування з використанням інформаційних технологій над звичайним тестуванням. З метою уникнення важливих недоліків проаналізованих платформ, а саме: незручний інтерфейс, платна ліцензія, складність встановлення програмного забезпечення, відсутність перевірки на чесність тощо, було вирішено розробити власну платформу, як веб-додаток.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПЛАТФОРМИ

2.1 Обгрунтування вибору технологій для розробки платформи

Розроблювана система буде побудована на основі клієнт-серверної архітетури. Вона включає такі частини, як:

- база даних. Для даного типу завдань може використовуватися будь-яка СУБД (Postgres, Oracle, MySQL, MongoDB, Microsoft SQL Server,тощо.);
- прикладний рівень. На даному рівні зосереджена основна частина бізнес логіки;
- рівень відображення. На даному рівні, реалізується логіка відображення даних, та частина бізнес логіки яка не може оброблятися на сервері (Наприклад: перевірка неактивності користувача);

Серед усіх альтернатив було обрано базу даних Postgres, для розробки прикладного рівня (сервера) було обрано технологію NodeJS та фреймворк NestJS, для розробки рівня відображення було обрано фреймворк Angular.

Для написання коду було обрано IDE Webstorm, для проектування баз даних та написання sql-коду було обрано клієнт pgAdmin.

NodeJS – це платформа, дозволяє використовувати Javascript за межами браузера. В основі платформи лежить виключно швидкий рушій JavaScript, яка взята Chrome, V8, до якого додано швидку та надійну бібліотеку асинхронного мережевого вводу/виводу. Основний упор в Node створюється на створенні і високопродуктивних, масштабованих клієнтських і серверних додатків для веб-додатків у реальному часі.

Архітектура Node побудовна аби забезпечити найвищу здатність до масштабованості – за рахунок поєднання асинхронного вводу/виводу, використанні мови JavaScript на сервері та однопоточного подієво-орієнтованого підходу [16].

Прийнята у Node модель принципово відрізняється від поширених платформ для побудови серверних додатків, у яких масштабованість досягається за рахунок багатопоточності. Через подієво-орієнтовану архітектуру знижується споживання пам'яті, зменшується складність системи та підвищується пропускна спроможність. На даний момент Node швидко розвивається, і тому багато хто розглядає її альтернативою звичного підходу до розробки веб-додатків – на базі .Net, Apache, Java, PHP, Python тощо [17].

Node побудована на основі віртуальної машини JavaScript з розширеннями, що довзволяє використовувати її для програмування загального призначення з ухилом в розробку серверів. Платформу Node не має сенсу порівнювати безпосередньо ні з мовами програмування, які зазвичай використовуються для створення веб-серверів (PHP/Python/Ruby/Java та інші), ні з контейнерами, що реалізують протокол HTTP (Apache/Tomcat/Glassfish іт. д.). У той же час вважається, що потенційно вона може замінити традиційні стеки веб-додатків [18].

NestJS – це серверний фреймворк Node.js з відкритим вихідним кодом, за замовчуванням Nest використовує популярний та надійний HTTP-сервер Express, при потребі, можна обрати та налаштувати також Fastify, що написаний на TypeScript, або написати свої реалізацію обробки HTTP-записів. Однак, хоча для Node.js існує дуже багато корисних бібліотек та інструментів, жоден з них на відміну від NestJS не вирішує головну проблему – забезпечення надійной архітектури та складність при масштабуванні коду [19].

NestJS сумісний JavaScript і побудований за допомогою TypeScript і поєднує їх сильні сторони:

- ООП (об'єктно-орієнтоване програмування);
- ФП (функціональне програмування);
- ФРП (функціональне реактивне програмування).

Плюси фреймворку NestJS:

- розширюваність: гнучкість при використанні інших бібліотек.

- універсальність: екосистема яка швидко адаптується для всіх типів серверних додатків.
- прогресивність: вносить нові функції JavaScript та паттерни проектування в спільноту Node.

Nest.js багато в чому дуже схожий на Angular. Він як і Angular містить контейнер для DI, важливоу функції яку не містять в собі інші популярні фреймворки. Архітектура та синтаксис також збігаються з Angular [20].

WebStorm – це інтегроване середовище для розробки на JavaScript та пов'язаних з ним технологіях. Як і інші IDE JetBrains, WebStorm дозволяє автоматизувати рутинну роботу та легко справлятися зі складними завданнями. До її плюсів також можна віднести: ширикі можливості для кастомізації, зручні підказки при написання коду, просунутий функціонал тощо. Серед мінусів варто виділити: високе споживання оперативної пам’яті, високу вартість ліцензії [21].

PostgreSQL – це потужна система баз даних об'єктно-реляційного типу, з відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які забезпечують безпеку зберігання і масштабують найскладніші навантаження даних. СУБД PostgreSQL заслужила міцну репутацію завдяки своїй архітектурі, цілісності даних, надійності, надійному набору функцій, розширюваності та відданості спільноти щодо відкритиго вихідного коду, яка стоять за програмним забезпеченням, щоб постійно надавати продуктивні та інноваційні рішення.

PostgreSQL постачається з багатьма функціями, які допомагають розробникам створювати програми, адміністраторам – захищати цілісність даних і створювати відмовостійке середовище, а також допомагають керувати даними незалежно від того, наскільки великий чи маленький набір даних. Окрім того, що PostgreSQL є безкоштовним і відкритим вихідним кодом, який можна легко розширювати. Наприклад, можна задекларувати та визначати

власні типи даних, створювати власні функції, в тому числі написати код з використанням різних мов програмування без перекомпіляції бази даних [22].

PostgreSQL намагається відповідати стандарту SQL, якщо така відповідність не суперечить традиційним функціям або може призвести до неправильних архітектурних рішень. Багато функцій, необхідних стандартом SQL, підтримуються, хоча іноді вони мають дещо відмінний синтаксис або функції. З часом можна очікувати подальших кроків до більшої відповідності. Починаючи з версії 14 у вересні 2021 року, PostgreSQL відповідає щонайменше 170 із 179 обов’язкових функцій для відповідності SQL:2016 Core. На момент написання цієї статті жодна реляційна база даних не відповідає повній відповідності цьому стандарту [23].

Платформа pgAdmin – це платформа з відкритим вихідним кодом для адміністрування та розробки для PostgreSQL і пов'язаних з нею системою управління базами даних. Платформа написана на Python і jQuery і підтримує всі функції PostgreSQL. Можна використовувати pgAdmin для будь-яких операцій, починаючи з налаштувань базових запитів SQL і завершенням моніторингу ваших баз даних і просунутих архітектурних баз даних [24].

Angular це фреймворк який створила компанія Google для створення додатків для клієнтської частини. Насамперед він орієнтований на розробку Single Page Application (односторінкових додатків), на відміну від стандартного підходу коли є багато html-файлів, в SPA є тільки один html-файл та в задежності від маршрутизації або від різноманітних умов підставляються необхідні представдення. У цьому плані Angular є правонаступником іншого фреймворку AngularJS. У той же час, Angular це не нова версія AngularJS, а принципово інший фреймворк [25].

Angular має такий функцінал як двостороннє зв'язування, що дозволяє синхронізувати дані в батьківському і дочірньому компоненті, шаблони, маршрутизація, анімації, бібліотеку для роботи з формами тощо. Однією з його особливостей є те, що він використовує мову програмування TypeScript,

що робить розробку простішою та зменшує кількість помилок які може зробити програміст. Іншою особливістю фреймворка є те що він реалізує компонентний підхід [26].

Проекти написані з Angular можуть масштабуватися від проектів з одним розробником до програм корпоративного рівня. Angular розроблено, щоб зробити оновлення до нової версії фреймворка максимально простим та швидким, для цього навіть існує спеціальна утиліта яка автоматизує процес. Екосистема Angular складається з різноманітної групи з понад 1,7 мільйонів розробників, авторів бібліотек і творців контенту. Остання версія Angular на момент написання роботи – Angular 13 вийшла у листопаді 2021 року [27].

2.2 Загальна архітектура платформи

Для побудови гнучкої та ефективної клієнт-серверної системи, найкраще обрати трирівневий варіант клієнт-серверної архітектури.

Клієнт-серверна архітектура – це модель, в якій сервер займається розміщення, доставкою та керуванням більшістю ресурсів і послуг, які споживає клієнт. Архітектура клієнт-сервер відома під назвою модель мережевих обчислень або мережа клієнт-сервер, оскільки всі запити та послуги проходять через мережу до серверів, на рис. 2.1 показано структурну схему клієнт-серверної системи [28].

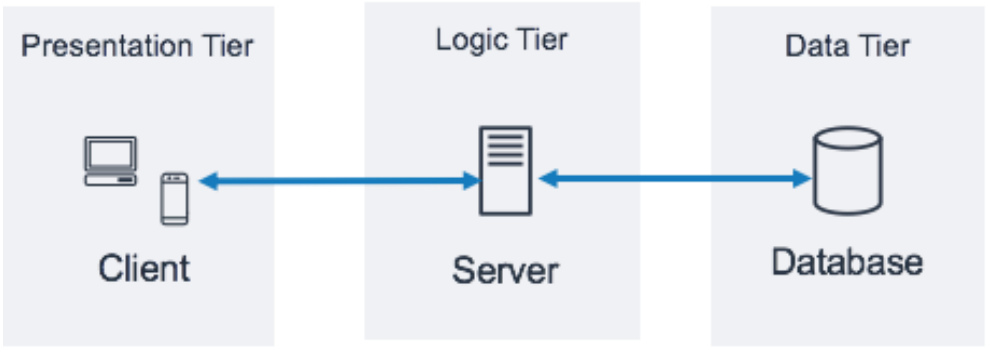

Рисунок 2.1 – Схема архітектури клієнт-серверної системи

Складові клієнт-серверної архітектури:

- сервер: частина програмного забезпечення, яка отримує та обробляє запити від клієнтів.
- балансувальник навантаження: відповідає за розподіл вхідного мережевого трафіку між групою серверів внутрішньої частини для оптимізації використання ресурсів.
- протоколи мережевого рівня, такі як TCP/IP.
- потік даних є односпрямованим і утворює цикл. Зазвичай він ініціюється клієнтом, який запитує певні дані, а сервер обробляє запит і надсилає певні дані назад клієнту через протокол. Клієнти не можуть безпосередньо спілкуватися один з одним. Типовий топологічний потік даних виглядає наступним чином:
- клієнт запитує дані від сервера.
- балансувальник навантаження направляє запит на відповідний сервер.
- сервер займається обробкою запитів клієнта.
- сервер запитує відповідну базу даних для деяких даних.
- база даних повертає запитані дані назад на сервер.
- сервер обробляє дані та надсилає дані назад клієнту.
- цей процес повторюється.

Проблеми які вирішує дана архітектура:

- клієнт-серверна архітектура є найбільш корисною для програм, в яких необхідний поділ між клієнтом і сервером; він призначений для систем з високою сумісністю. Стиль клієнт-серверної архітектури допомагає програмам покращити продуктивність у масштабованості.
- в системах, які потребують розділення функціональних можливостей, дизайн архітектури клієнт-сервер є найбільш поширеним. Перевірка запиту та введення можуть оброблятися з боку клієнта, поки балансувальник навантаження направляє запит на сервер для належної

обробки. Сервер відповідатиме за обробку запиту клієнта та повернення результату за допомогою правильного протоколу. Ці рівні (клієнт і сервер) виконують завдання незалежно, і вони корисні для абстрагування функціональних можливостей; наприклад, клієнту не потрібно знати, як сервер обробляє аутентифікацію користувача або перевірку запитів.

- з поділом функціональних можливостей кожен рівень функціонує більш ефективно у великих масштабах. Сучасні методи були розроблені в клієнт-серверні архітектурі для вирішення проблем масштабованості, таких як балансування навантаження, шардінг і розділення. Ці методи забезпечують підвищення продуктивності для кількох запитів на серверній стороні архітектури і будуть корисні для програм, які працюють з кількома запитами/користувачами.
- цей стиль архітектури надзвичайно гнучкий і може бути адаптований до користувача та набору проблем.
- його також можна поєднувати з іншими типами архітектури на стороні клієнта або сервера.
- внутрішні зміни на одному рівні не спливають на зміни на іншому [29].

2.3 Архітектура серверного додатку

Слоїста архітетура була представлена Джеффрі Палермо, щоб забезпечити кращий спосіб створення додатків для забезпечення кращого забезпечення тестування і легкості підтримки та надійності. Багатошарова архітектура вирішує проблеми, з якими стикаються 3- і n-рівневі архітектури, а також надає рішення для поширених проблем. Слої архітектури взаємодіють один з одним за допомогою інтерфейсів [30].

Відповідно до традиційної архітектури, рівень інтерфейсу інтерфейсу взаємодіє з бізнес-логікою, а бізнес-логіка взаємодіє з рівнем даних, і всі шари

перемішані і сильно залежать один від одного. У 3-рівневій і n-рівневій архітектурі жоден з рівнів не є незалежним; цей факт викликає розділення проблем. Такі системи дуже важко зрозуміти та підтримувати. Недоліком цієї традиційної архітектури є непотрібне зчеплення.

Багатошарова архітектура вирішила цю проблему, розділивши шари від ядра до інфраструктури. Ця архітектура, безсумнівно, заснована на основі об'єктно-орієнтованого програмування. У основі слоїстої архітектури знаходиться модель домену, яка представляє бізнес-моделі та бізнес-процеси. Навколо шару домену розташовані інші шари з більшою кількістю поведінки. На рис. 2.2 зображено шари багатошарової архітектури [31].

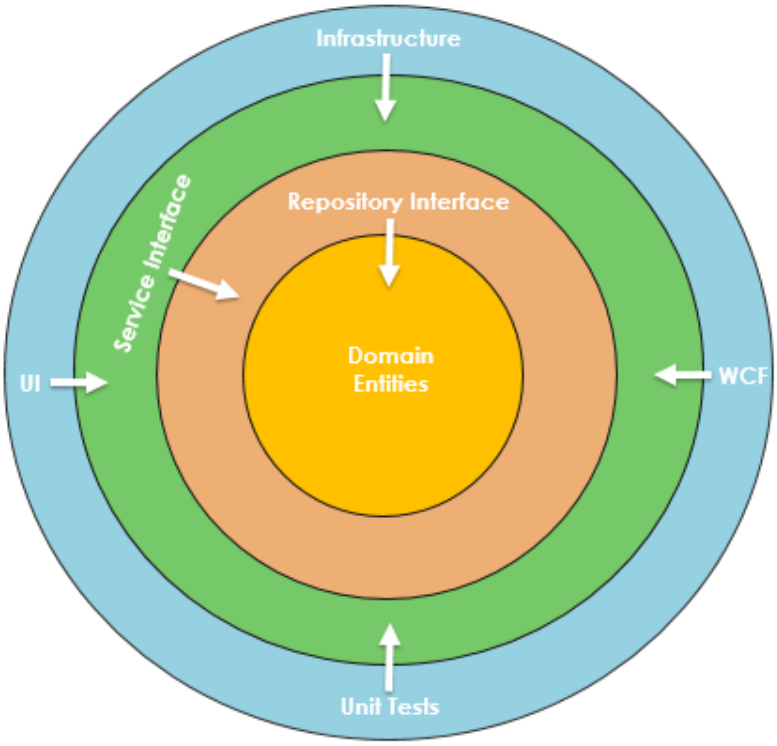

Рисунок 2.2 – Шари багатошарової архітектури

Дана архітектура використовує концепцію слоїв, але вона відрізняються від 3-рівневої та n-рівневої архітектурни. Що представляє і має містити кожен із цих шарів:

- доменний слой. У центрі архітектури існує шар домену; цей шар представляє бізнес-процеси і бізнес-моделі. Ідея полягає в тому, щоб усі об’єкти домену були в цьому ядрі. Окрім об’єктів домену, є допускаються інтерфейси домену. Ці об’єкти домену не мають мати жодних залежностей. Об’єкти домену також максимально прості, без системного коду чи складних залежностей;
- рівень збереження даних. Цей рівень створює абстракцію між сутностями домену та бізнес-логікою програми. На цьому рівні зазвичай додаються інтерфейси, які забезпечують збереження об’єктів та поведінку вилучення, як правило, за допомогою бази даних. Цей рівень складається з шаблону доступу до даних, який є більш слабо пов’язаним підходом до доступу до даних. Створюється загальний репозиторій і додаються запити, щоб отримати дані з джерела, зіставляти дані з джерела даних у бізнес-сутності, і зберігати зміни в бізнес-сутності в джерелі даних;
- рівень бізнес логіки. Цей рівень використовується для зв’язку між рівнем UI та рівнем сховища. Рівень служби також може містити бізнес-логіку для сутності. На цьому рівні сервісні інтерфейси розділяються від його реалізації, забезпечуючи про слабке зчеплення та розділення відповідальності;
- рівень відображення. Це зовнішній рівень, який відображає периферійні сутності, такі як інтерфейс користувача та тести. Для веб-додатків він представляє веб-API або проект модульного тестування. Цей рівень має реалізацію принципу ін’єкції залежностей, так що програма створює слабко зчеплену структуру і може спілкуватися з внутрішнім рівнем через інтерфейси;

Нижче наведено переваги впровадження слоїстої архітектури:

- шари слоїстої архітектури підключаються через інтерфейси. Реалізація інтерфейсу може змінюватися під час роботи програми.
- архітектура програми побудована на основі моделі домену.

- усі зовнішні залежності, такі як доступ до бази даних і виклики служб, представлені на зовнішніх рівнях.
- немає залежностей внутрішнього шару із зовнішніми шарами.
- гнучка, стійка та портативна архітектура.
- немає необхідності створювати спільні проекти.
- можна швидко протестувати, оскільки ядро програми ні від чого не залежить [32].

2.4 Архітектура клієнтського додатку

Архітектура програм Angular базується на деяких базових концепціях. Основне з чого складається проект на Angular – це компоненти, які в свою чергу організовані в NgModules. NgModules збирає пов’язані компоненти, та інший схожий по функціоналу кож; додатки Angular складаються з набору NgModules. Завжди як мінімум повинен існувати хоча б один кореневий модуль, який дозволяє завантажувати інші функціональні модулі.

Компоненти визначають представлення, які є наборами елементів екрану, які Angular може вибирати та змінювати відповідно до логіки та даних вашої програми.

Компоненти звертаються до сервісів, які надають конкретні функції, не пов’язані безпосередньо з представленнями. Сервіси можуть бути введені в компоненти як залежності, що робить код модульним, багатовикористовуваним та ефективним.

Модулі, компоненти та сервіси – це класи, в яких використовуються декоратори. Ці декоратори позначають тип класу і надають метадані, які повідомляють Angular, щоб той міг їх використовувати.

Метадані для класу сервісу надають інформацію, необхідну Angular, щоб зробити її доступною для компонентів за допомогою ін'єкції залежностей (DI).

Компоненти програми представляють собою графічні елементи, які розташовані ієрархічно. Angular надає модуль Router, щоб допомогти визначити шляхи навігації між представленнями.

Архітектура додатку на Angular складається з таких основних елементів: модулі, компоненти, шаблони, директиви, сервіси, маршрутизація тощо, на рис. 2.3 зображено діаграму яка зв’язки між концепціями [33].

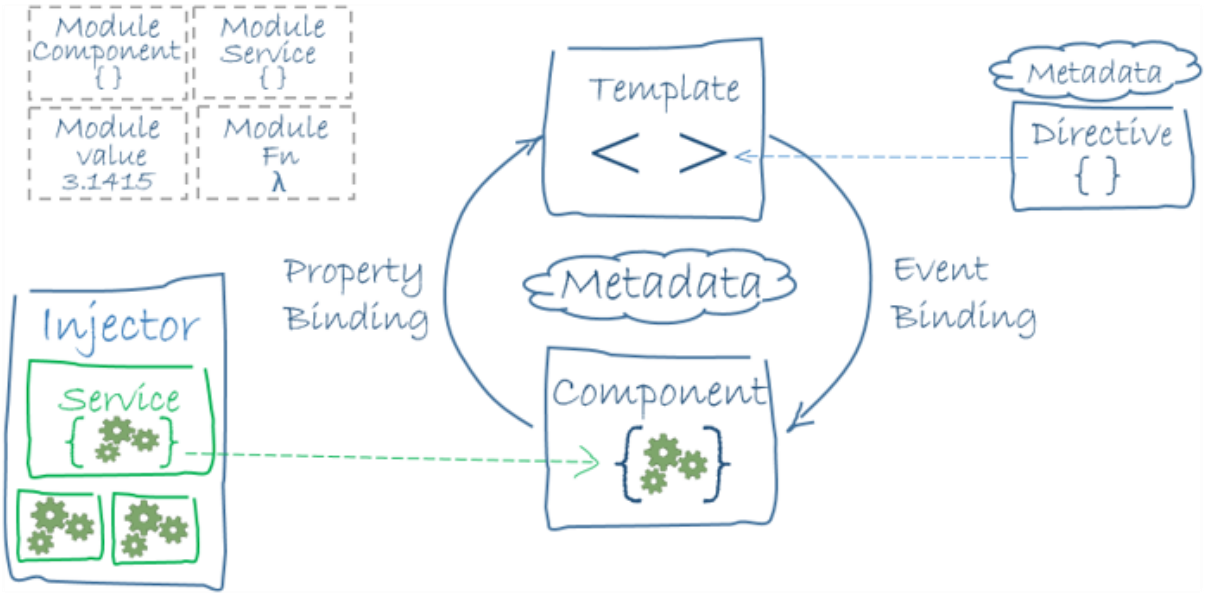

Рисунок 2.3 – Діаграми зв’язків між основними концепціями додатку спректованого з допомогою Angular

Модулі. Angular NgModules відрізняється від модулів JavaScript (ES6) і доповнює їх. NgModule оголошує контекст компіляції для групи компонентів, що використовуються в домені програми, робочому процесі або тісно пов'язаному наборі функцій. NgModule може асоціювати свої компоненти з кодом (наприклад, сервісами), щоб утворити функціональні одиниці. Кожна програма Angular містить кореневий модуль, який зазвичай має назву AppModule, він забезпечує механізм запуску програми. Додаток зазвичай містить складається з функціональних модулів. Як і модулі JavaScript, NgModules можуть імпортувати функції з інших NgModules і дозволяти вам експортувати власні функції та використовувати інші NgModules. Наприклад, щоб використовувати службу маршрутизатора у своїй програмі, потрібно

скористатися маршрутизатором NgModule. Організація коду в окремі функціональні модулі допомагає керувати розробкою складних додатків і розробляти їх для повторного використання. Крім того, ця техніка дозволяє вам скористатися перевагами відкладеного завантаження, тобто завантаження модулів на вимогу, щоб мінімізувати кількість коду, який потрібно завантажити під час запуску.

Компоненти. Кожна додаток на містить має принаймні хоча б один компонент, який називається кореневим компонентом, який з'єднує ієрархію компонентів з об'єктною моделлю документа сторінки (DOM). Кожен компонент визначає клас, який містить дані програми та логіку, і пов'язаний з ним шаблон HTML, який визначає графічний елемент, який відображатиметься в браузері. Декоратор @Component() визначає клас безпосередньо під ним як компонент і надає шаблон і пов'язані метадані, специфічні для компонента.

Шаблони, директиви та прив'язка даних. Шаблон поєднує HTML з розміткою Angular, яка може змінювати елементи HTML перед їх відображенням. Директиви шаблонів забезпечують програмну логіку, а розмітка зв'язування з'єднує дані програми та DOM. Існує два типи прив'язки даних:

- прив'язка подій дозволяє вашій програмі реагувати на введення користувача в цільовому середовищі, оновлюючи дані програми.
- прив'язка властивостей дозволяє інтерполювати значення, обчислені з даних програми, у HTML.

Перед відображенням представлення Angular оцінює директиви та вирішує синтаксис прив'язки в шаблоні, щоб змінити елементи HTML і DOM відповідно до даних і логіки вашої програми. Angular підтримує двостороннє прив'язування даних, що означає, що зміни в DOM, такі як вибір користувача, також відображаються в даних програми.

Сервіси та ін'єкція залежностей. Створюється класи сервісів для даних або логіки, які не пов’язані з певним представленням і є спільними між компонентами. Визначений класу сервісу визначається безпосередньо декоратором @Injectable. Декоратор надає метадані, які дозволяють іншим провайдерам вводити залежності у цей клас. Ін'єкція залежностей дозволяє підтримувати класи компонентів впорядкованими та ефективними. Вони не отримують дані від сервера, не валідують дані, введені користувачем вони делегують такі завдання сервісам.

Angular Router NgModule надає функцію, яка дозволяє визначати шлях навігації між різними станами програми та переглядати ієрархію маршрутизації в браузері маршрутизатор замість сторінки відображає шлях перегляду, подібний до URL-адреси. Коли користувач виконує операцію, наприклад, натискає посилання, яке завантажує нову сторінку в браузер, маршрутизатор перехопить поведінку браузера та покаже або приховає ієрархію перегляду [34].

2.5 Висновки до другого розділу

В даному розділі досліджено та розглянуто клієнт-серверну архітектуру, для серверного додатку обрано слоїсту архітектру та компонентний підхід з інверсією залежностей. Вибір цих архітектурних підходів довзоляє легко добавляти новий функціонал, та змінювати старий при зміні вимог. Також обрано засоби, та середовища, які забезпечують оптимальну продуктивність розробника та високі швидкодію системи. Описано особливості та переваги кожного обраного рішення.

3 РЕАЛІЗАЦІЯ ПЛАТФОРМИ ДЛЯ ПЕРЕВІРКИ ЗНАНЬ ШЛЯХОМ ТЕСТУВАННЯ

3.1 Структура бази даних

Якість проектування бази даних може проводити роботу з нею. З добре спроектованою базою даних легше працювати, легше писати запити, добавляти нові поля і таблиці.

Для якісного проектування бази даних існують різні методики, різні послідовності кроків чи етапів, які багато в чому схожі. І загалом ми можна виділити такі етапи:

- виділення сутностей та його атрибутів, які зберігатимуться у базі даних, і формування з них таблиць. Атомізація складних атрибутів на більш прості;
- визначення унікальних ідентифікаторів (первинних ключів) об'єктів, що зберігаються у рядках таблиці;
- визначення відносин між таблицями за допомогою зовнішніх ключів;
- нормалізація бази даних.

У першому етапі відбувається виділення сутностей. Сутність (entity) є типом об'єктів, які повинні зберігатися в базі даних. Кожна таблиця у базі даних має представляти одну сутність. Як правило, сутності відповідають об'єктам із реального світу [35].

Кожна сутність визначає набір атрибутів. Атрибут є властивістю, яка описує деяку характеристику об'єкта.

Кожен стовпець повинен зберігати один атрибут сутності. А кожен рядок представляє окремий об'єкт чи екземпляр сутності.

Нормалізація – це метод проектування бази даних, що дозволяє привести базу даних до мінімальної надмірності. Надмірність усувається, зазвичай,

рахунок декомпозиції зв’язків (таблиць), тобто розбиття однієї таблиці на кілька. База даних вважається нормалізованою, якщо вона знаходиться як мінімум у третій нормальній формі (3NF). На практиці нормалізація до третьої нормальної форми (3NF) є звичайною, стандартною практикою, так як 3NF усуває достатню кількість аномалій, при цьому продуктивність бази даних, а також зручність її використання не знижується, що не можна сказати про всі наступні форми [36].

Ситуації, у яких потрібно нормалізувати базу даних до четвертої нормальної форми (4NF), у на практиці зустрічаються досить рідко. Якщо говорити про всі наступні нормальні форми (5NF, DKNF, 6NF), то на практиці важко навіть уявити ситуації, за яких потрібно нормалізувати базу даних до цих форм. Іншими словами, 5NF, DKNF, 6NF – це переважно теоретичні нормальні форми, трохи відсторонені від реального світу.

Надмірність даних – це коли одні й самі дані зберігаються у БД в кількох місцях, саме це і призводить до аномалій. Так як в цьому випадку необхідно додавати, змінювати або видаляти ті самі дані в декількох місцях. Наприклад, якщо не виконати операцію в якомусь одному місці, то виникає ситуація, коли одні дані не відповідають начебто таким же даними в іншому місці [37].

Для роботи з користувачами використовуються таблиця users. Вона містить такі поля:

id – ключове поле потрібне для забезпечення унікальності записів, та швидкого отримання запису;

createdAt, updatedAt – поле яке автоматично генерується, позначає відповідно час коли запис було створено та відредаговано;

email – унікальне текстове поле яке зберігає електронну пошту користувача, обо’язкове для заповнення;

firstName – текстове поле яке зберігає ім’я користувача, обов’язкове для заповнення;

lastName – текстове поле яке зберігає прізвище користувача, обов’язкове для заповнення;

password – текстове поле яке зберігає пароль користувача в зашифрованому вигляді, обов’язкове для заповнення;

role – поле перелічуваного типу, допустимі значення: “student”, “admin”, значення за замовчуванням “student”.

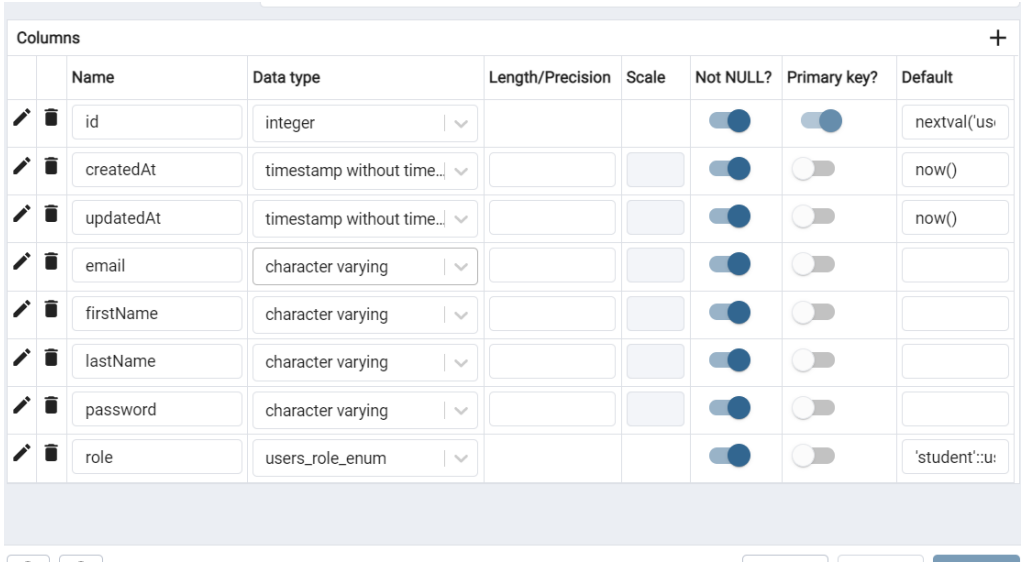

Рисунок 3.1 – Структура таблиці users в pgAdmin.

Для зберігання інформації про предмети використовується таблиця subjects. Вона містить такі поля:

id – ключове поле потрібне для забезпечення унікальності записів, та швидкого отримання запису;

name – текстове поле що містить назву предмету, обов’язкова для заповнння;

description – текстове поле що опис предмту, обов’язкова для заповнення;

authorId – зовнішній ключ який посиляється на таблицю users, з зв’язком багато-до-одного, та позначає автора предмета.

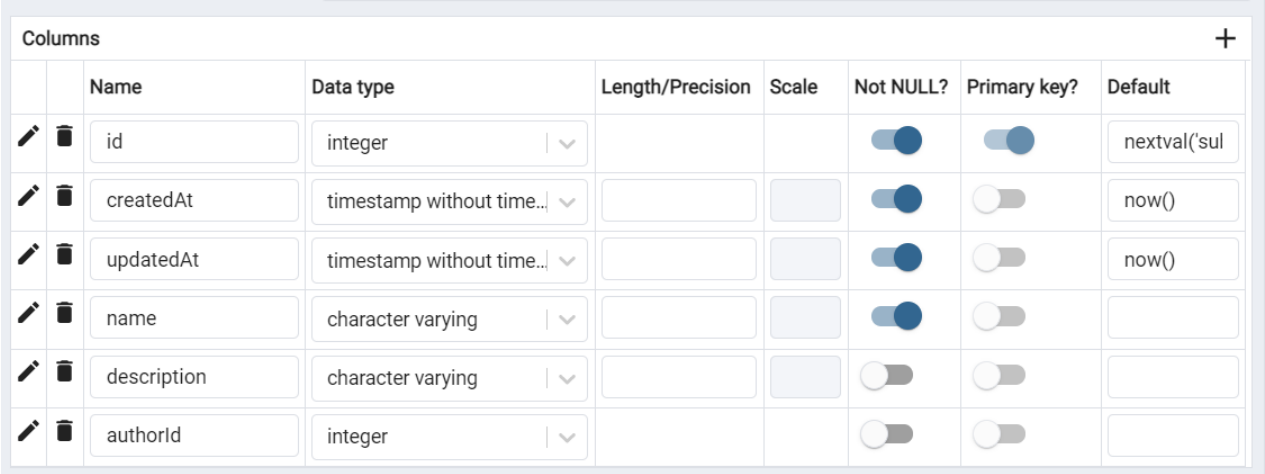

Рисунок 3.2 – Структура таблиці subjects в pgAdmin

Для зберігання інформації про тести використовується таблиця quizzes. Вона містить такі поля:

id – ключове поле потрібне для забезпечення унікальності записів, та швидкого отримання запису;

title – текстове поле яке містить коротку інформацію про тест, обов’язкове для заповнння;

description – текстове поле що опис тесту, обов’язкове для заповнення;

authorId – зовнішній ключ який посиляється на таблицю quizzes, з зв’язком багато-до-одного, та позначає автора тесту.

subjectId – зовнішній ключ який посиляється на таблицю quizzes, з зв’язком багато-до-одного, та позначає предмет до якого відноситься тест.

type – поле перелічуваного типу, допустимі значення: “steps”, “whole”, значення за замовчуванням “whole ”.

maxAttempts – поле цілочисельного типу, значення за замовчуванням 3. Позначає скільки разів користувач може здати тест.

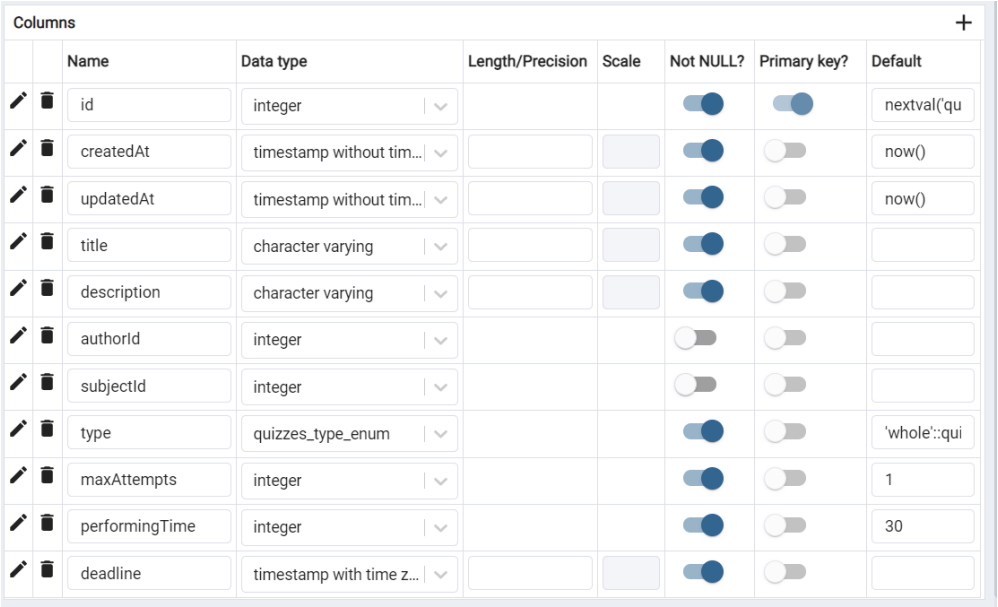

Рисунок 3.3 – Структура таблиці quizzes в pgAdmin

Для зберігання інформації про питання використовується таблиця questions. Вона містить такі поля:

id – ключове поле потрібне для забезпечення унікальності записів, та швидкого отримання запису;

title – текстове поле яке містить текст питання, обов’язкове для заповнння;

quizId – зовнішній ключ який посиляється на таблицю quizzes, з зв’язком багато-до-одного, та позначає тест до якого відноситься питання.

answers – поле типу массиву, містить запитання варіанти відповідей;

correctIndexes – поле типу массиву, який містить індекс чи індекси правильних відповідей.

type – поле перелічуваного типу, допустимі значення: “single”, “multiple”, значення за замовчуванням “single ”.

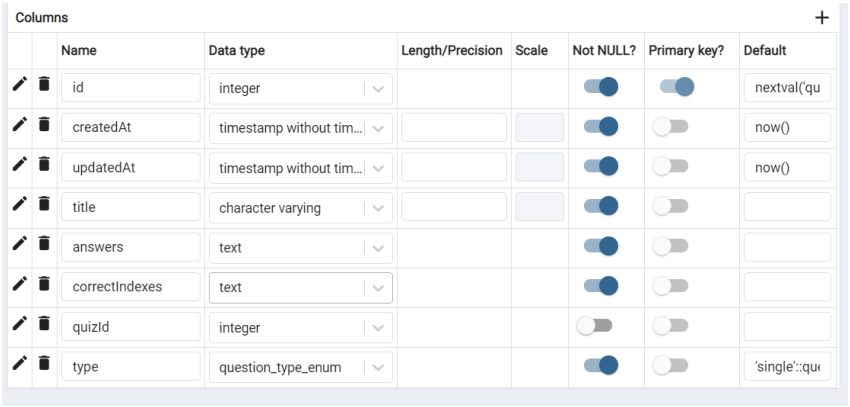

Рисунок. 3.4 – Структура таблиці questions в pgAdmin.

Для зберігання інформації про спроби здачу тесту використовується таблиця attempts. Вона містить такі поля:

id – ключове поле потрібне для забезпечення унікальності записів, та швидкого отримання запису;

quizId – зовнішній ключ який посиляється на таблицю quizzes, з зв’язком багато-до-одного, та позначає тест до якого відноситься спроба.

userId – зовнішній ключ який посиляється на таблицю users, з зв’язком багато-до-одного, та позначає користувача який здавав тест.

userId – зовнішній ключ який посиляється на таблицю users, з зв’язком багато-до-одного, та позначає користувача який здавав тест.

fineshed – булеве поле, яке позначає чи спроба вже завершилась.

score – поле цілочисельного типу, яка позначає оцінку на яку склав користувач.

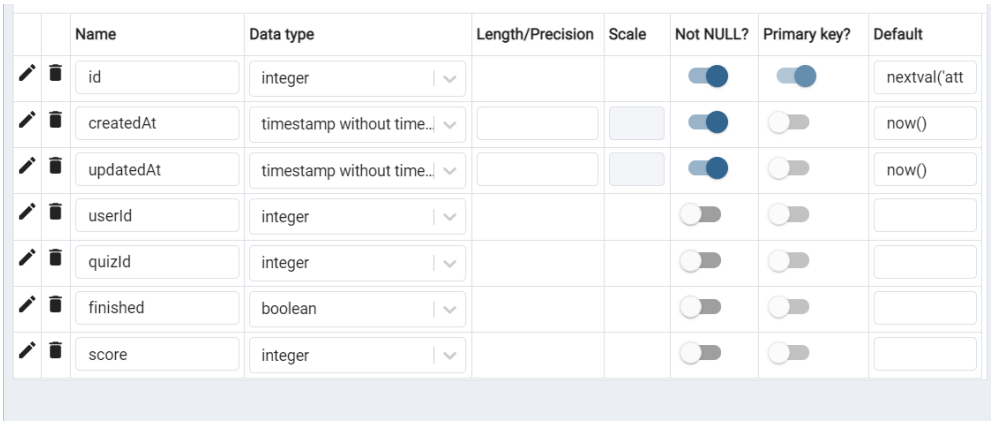

Рисунок 3.5 – Структура таблиці attempts в pgAdmin.

Таблиця subjects_students_users реалізує зв’язок багато до багатьох та відображає студентів які підписалися на предмет.

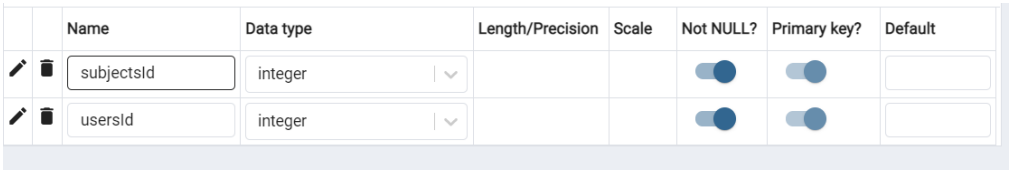

Рисунок 3.6 – Структура таблиці subjects_students_users в pgAdmin.

EER-діаграму бази даних в цілому можна побачити на рис. 3.7.

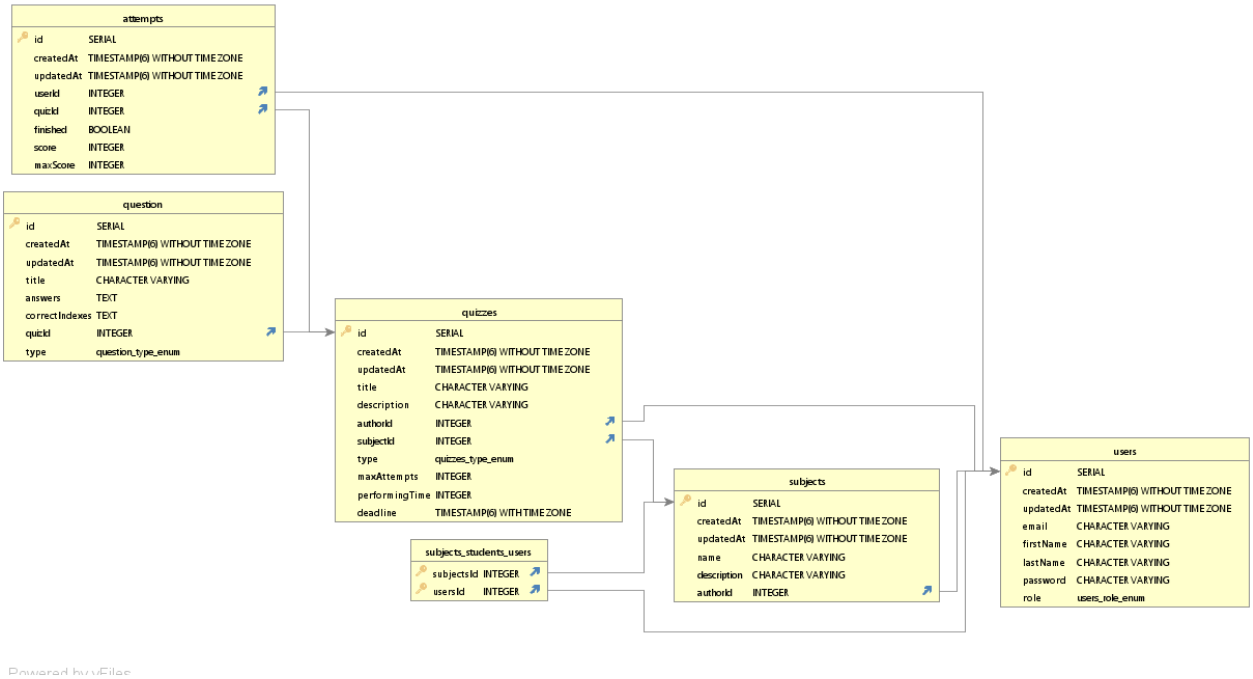

Рисунок 3.7 – EER-діаграма бази даних.

3.2 Реалізація модулів програмної логіки

Для розмітки було обрано бібліотеку ng-zorro, її було обрано через те що це бібліотека компонентів Angular UI корпоративного рівня на основі Ant Design, всі компоненти з відкритим вихідним кодом і безкоштовні для використання за ліцензією MIT. Її особливості:

- більше 60 високоякісних компонентів Angular із коробки;

- написана з використанням TypeScript та повністю типізованими компонентами;
- підтримка режиму OnPush, висока продуктивність;
- гнучке налаштування теми в кожному компоненті;
- підтримка інтернаціоналізації для десятка мов [38].

Будь-яка платформа містить функціонал авторизації. Платформа для перевірки знань використовує JWT для аутентифікації та авторизації. Його особливістю є те що дані про сесії не зберігаються на сервері, а унікальний ідентифікатор і метедані про користувача зашифровуються і зберігаються на стороні користувача сервер розшифровує та валідує токен, і на основі цього визначає чи авторизований користувач[39]. Для забезпечення безпеки було вирішено зберігати токен в http-only куках, це дозволяє захистити платформу від міжсайтового скриптингу. HttpOnly Cookie – це тег, який додається до серверної відповіді для браузера, який недозволяє доступ до даних скриптам які знаходяться на клієнті. Він забезпечує шлюз, який запобігає доступу до спеціалізованих файлів cookie будь-чого, крім сервера [40].

На серверній стороні для авторизація використовується бібліотека Passport.js. Бібліотека гнучка і проста в налаштуваннях, при необхідності можна легко добавити нові способи авторизації, наприклад за допомогою: Google, Facebook, Twitter, Linkedin тощо [41]. В лістингу 3.2.1 можна побачити валідацію JWT-токена в фреймворку Nest.js.

Лістинг 3.1 – Валідація JWT-токена

```
@Injectable()
export class JwtCookieStrategy extends
PassportStrategy(Strategy) {
  constructor(
    @InjectRepository(UserEntity) private userRepository:
Repository<UserEntity>,
  ) {
    super({
```

```
      jwtFromRequest: (req) => {
        return parseCookie(req).token;
      },
      secretOrKey: process.env.SECRET,
    });
  }
  async validate(payload: AuthPayload) {
    const { id } = payload;
    const user = await this. userRepository.find({ where: {
id } });
    if (!user) {
      throw new UnauthorizedException();
    }
    return user;
  }
}
export function parseCookie(req): { [key: string]: string }
{
  const { cookie } = req.headers;
  if (!cookie) return;
  const result = {};
  const items = cookie.split(';');
  for (const item of items) {
    const parts = item.split('=');
    const key = parts[0].trim();
    const val = parts[1] || '';
    result[key] = val.trim();
  }
  return result;
}
```

На стороні клієнта, з міркувань безпеки, зберігати токен не варто. Але можна зберігати час за який токен втратить актуальність, це потрібно для вчасного оновлення токена та коректної роботи програми. Перевірку на

актуальність токена можна побачити в лістингу 3.2. HTML-розмітку сторінки логіну та реєстрації можна побачити в додатку Б.

Лістинг. 3.2.– Перевірка на актуальність токена

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private router: Router) {
  }
  canActivate(route: ActivatedRouteSnapshot, state:
RouterStateSnapshot): Observable<boolean> | boolean {
    const expiresIn = localStorage.getItem('expiresIn'); //
&& this.validateExpiration();
    const isAllowed = !! expiresIn && new
Date(expiresIn).getTime() > Date.now();
    if (!isAllowed) {
      this.router.navigate(['auth', 'login']);
    }
    return isAllowed;
  }
  private validateExpiration(): boolean {
    const expiresIn = +localStorage.getItem('expiresIn');
    return Date.now() <= expiresIn;
  }
}
```

В системі необхідно розділяти користувачами за правами, передбачено два типи користувачів: адміністратора і студента. Адміністратор може створювати, видаляти, та редагувати предмети та тести. Користувач може здавати тести, переглядати їх результати, статистику пов’язану зі своїми тестами. На сервері для перевірки ролі користувача використовуються декоратори та концепцію guards, які надає фреймворк Nest.js. Вони роблять код більш лаконічним, декларативним та дозволяють усувати дублікати.

Працює це наступним чином на метод контроллера навішується декоратор з метаданими (в цьому випадку роль яка може здійснити дію) та в класі RolesGuarg перевіряється відповідність користувача до необхідної ролі [42]. Guard – це концепція яка є спільна для Angular та NestJS, клас повинен ,бути анотований декоратором @Injectable(), та реалізувати інтерфейс CanActivate. Guard має єдину відповідальність, він визначає, чи буде даний запит оброблений контролером, залежно від певних умов (наприклад, ролі, авторизації) [43]. Код класу RolesGuard показаний в лістингу 3.3.

Лістинг. 3.3.– Перевірка на актуальність токена

```
@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}
  canActivate(context: ExecutionContext): boolean {
    const roles = this.reflector.get<string>('role',
context.getHandler());
    if (!roles) {
      return true;
    }
    const request = context.switchToHttp().getRequest();
    const user = request.user;
    return this.matchRoles(roles, user.role);
  }
  matchRoles(expectedRole, actualRole){
    return expectedRole.includes(actualRole);
  }
}
```

Після логіну користувач бачить список доступних предметів. Після підписки на них від може складати тести. При створенні предмета, спочатку на клієнті відбувається первинна перевірка на правильну заповненість полів, потім сервер повторює перевірку на свої стороні (задля безпеки) і записує дані

в базу даних. Також як вже було сказано перед тим як здавати тест користувач повинен підписатися на предмет.

Лістинг. 3.4. – Код для створення предмета, та підписки/відписки користувача на предмет

```
async createItem(subjectData: any) {
    const data = { ...subjectData, author:
subjectData.author } as SubjectEntity;
    const subject = await this.subjectRepo.create(data);
    await subject?.save();
    return subject;
  }
  async subscribe(id: number, user: UserEntity) {
    const subject = await this.subjectRepo.findOne({ where:
{ id }, relations: ['students'] });
    if (subject.students.map(item =>
item.id).includes(user.id))
      throw new ForbiddenException('You are already
subscribed to to this subject');
    subject.students.push(user);
    await subject.save();
    return subject;
  }
  async unsubscribe(id: number, user: UserEntity) {
    const subject = await this.subjectRepo.findOne({ where:
{ id }, relations: ['students'] });
    subject.students = subject.students.filter(item =>
item.id !== user.id);
    await subject.save();
    return subject;
}
```

Весь код SubjectService як і всіх інших показаний у додатку В.

Логіка створення, редагування, та видалення тестів схожа з предметами. Тільки тест містить додаткові поля які використовуються в бізнес логіці, накриклад: кількість спроб, кінцевий термін здачі, час за який можна здати тест.

Створення тесту на сервері показано у лістингу 3.5.

Лістинг 3.5 – Створення тесту на серверві.

```
async createItem(data: QuizDto): Promise<QuizDto> {
    const subject = await this.subjectRepository.findOne({
where: { id: data.subjectId } });
    const { questions, ...quizDTO } = data;
    const quiz = this.quizRepo.create({ ...quizDTO, author:
data.author });
    quiz.questions = [];
    quiz.subject = subject;
    for (const questionDTO of questions) {
      const question =
this.questionRepo.create(questionDTO);
      await question.save();
      quiz.questions.push(question);
    }
    await quiz.save();
    return quiz as any;
  }
```

При складанні тесту спочатку перевіряється чи користувач не перевищив допустиму кількість спроб та чи кінцеву дату складання тесту менша ніж поточна. Якщо в вхідними даними все гаразд, то в базі даних створюється запис спроби (attepmt), і користувач повинен завершити тест за заданий термін. При завершенні тесту поточній спробі задається статус “завершено” перевіряється час за який тест був зданий та рахується і користувачу показується результат. Є декілька видів запитань: з однієї

правильної відповіддю, та декількома. Код обрахунку тесту показаний у лістингу 3.6.

Лістинг 3.6 – Код обрахунку тесту

```
async calculate(body) {
    const { id, answers } = body;
    const attempt = await this.getAttempt(id);
    const quiz = attempt.quiz;
    const deadline = (new Date(attempt.createdAt).getTime()
      new Date(attempt.createdAt).getTimezoneOffset()) +
(quiz.performingTime * 60 * 1000);
    const currentTime = new Date().getTime() + (new
Date().getTimezoneOffset() * 60 * 1000);
    if (currentTime > deadline) {
      throw new ForbiddenException('Time for this attempt is
already out', `Deadline was ${ new Date(deadline) },
      now is ${ new Date() }`);
    if (attempt.finished) {
      throw new ForbiddenException('Attempt is already
finished.');
    }
    const score = quiz.questions.reduce((score,
currentValue, index) => {
      if (currentValue.type === QuestionType.Single) {
        const answer = answers[index];
        return currentValue.correctIndexes.map(item =>
+item).includes(answer) ? score + 1 : score;
      } else {
        return
_.isEqual(currentValue.correctIndexes.map(item => +item),
answers[index]) ? score + 1 : score;
      }
    }, 0);
    attempt.finished = true;
```

```
    attempt.score = score;
    attempt.maxScore = quiz.questions.length;
    await attempt.save();
    return { score, total: quiz.questions.length };
  }
```

В платформі передбачено декілька видів статистики проходження тестів: за користувачем, за тестом, за предметом та у всіх системі. Це допомогає якнайповніше зібрати і проаналізувати дані, знайти теми в яких у користувачів є прогалини. Код обрахунку загальної статистики показаний у лістингу 3.7. Код який пов’язаний з обрахунками інших видів статистики можна побачити у додатку В.

Лістинг 3.7 – Код обрахунку статистики

```
 async fullStats() {
    const attempts = await this.attemptRepo.query('SELECT *
FROM attempts GROUP BY quiz_id');
    const allScore = attempts.reduce((total, item) => {
      return total + item.score / item.maxScore;
    }, 0);
    const quizStats = allScore / attempts.length;
    const subjectAttempts = await
this.attemptRepo.query('SELECT * FROM attempts GROUP BY
subject_id');
    const subjectScore = subjectAttempts.reduce((total,
item) => {
      return total + item.score / item.maxScore;
    }, 0);
    const subjectStats = subjectScore /
subjectAttempts.length;
    const userAttempts = await
this.attemptRepo.query('SELECT * FROM attempts GROUP BY
user_id');
```

```
    const userScore = subjectAttempts.reduce((total, item)
=> {
      return total + item.score / item.maxScore;
    }, 0);
    const userStats = userScore / userAttempts.length;
    return {
      userStats,
      subjectStats,
      quizStats,
    };
  }
```

Для перевірки на неактивність викоритовується клас ActivityTimer. Він працює за наступним алгоритмом, відслідковується події браузера на втрату та отримання фокусу, якщо є відбувається втрата фокусу, то в якості activityTime записується різниця між останнім отриманням фокусу та теперерішнью датою, якщо відбувається отримання фокусу то в якості inactivityTime записується різниця між останньою втратою фокусу та теперерішнью датою. Потім видається оцінка “неактивності” користувача як частка неактивного часу та суми активного та неактивного, код реалізації класу ActivityTimer можна побачити у лістингу 3.8.

Лістинг 3.8 – Код класу ActivityTimer

```
export class ActivityTimer {
  private blurTime = 0;
  private activityTime = 0;
  private startDate;
  start(): void {
    this.startDate = new Date();
  }
  startBlur(): void {
    this.activityTime = Date.now() -
this.startDate.getTIme();
```

```
    this.startDate = new Date();
  }
  startFocus(): void {
    this.blurTime = Date.now() - this.startDate.getTIme();
    this.startDate = new Date();
  }
  getInactivityRate(): number {
    return this.blurTime / (this.activityTime +
this.blurTime);
  }
}
```

3.3 Графічний інтерфейс платформи

3.3.1 Інтерфейс користувача

Будь-який додаток починається з авторизація, зазвичай при реєстрації портрібно ввести електронну пошту та пароль, іноді доступна реєстрація через соцмережі (Facebook, Instagram, Google, Linkedin тощо). В даній плафтформі доступна реєстрація через електронну пошту, в майбутньому планується добавити також реєстрацію через соцмережі. На рис. 3.8 показано форму логіну а на сторінці 3.9 сторінку реєстрації.

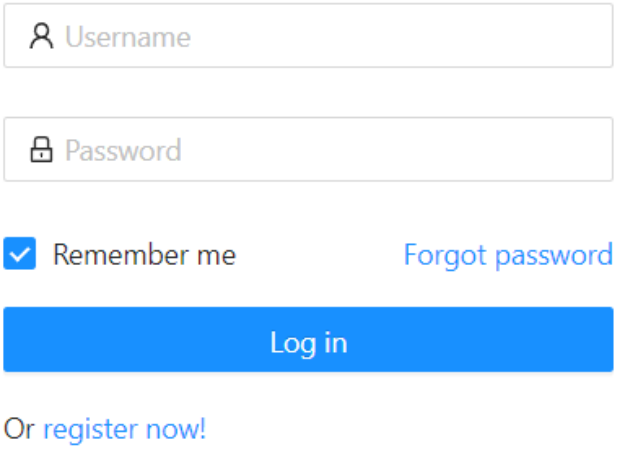

Рисунок 3.8 – Форма входу користувача в систему

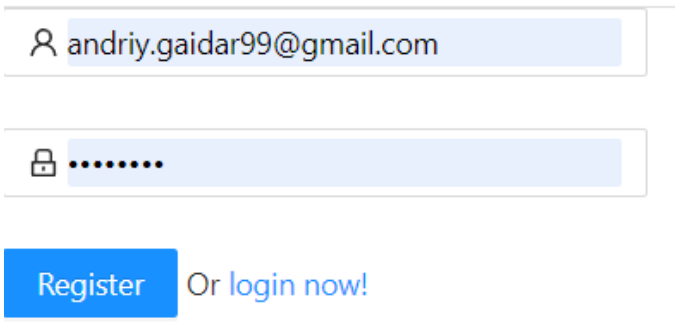

Рисунок 3.9 – Форма реєстрації нового користувача

Після того як користувач зареєстувався він бачить список предметів, для того аби здати тести йому необхідно спочатку підписатися на предмет. На рис. 3.10 показано список з предметами на які можна підписатися/відписатися.

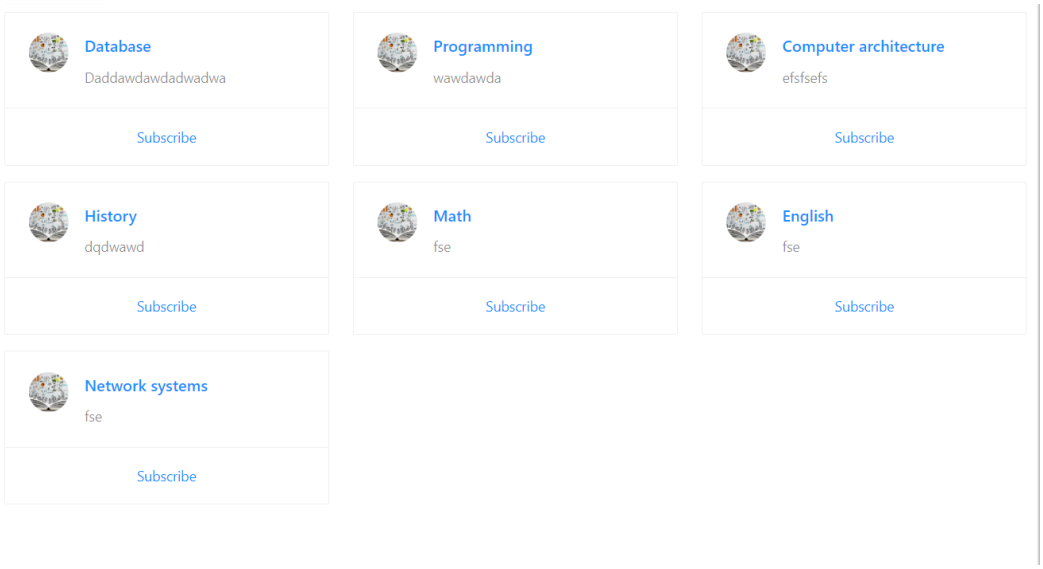

Рисунок 3.10 – Список предметів з можливістю підписки/відписки

Після того як користувач підписався на предмет, він може перейти на сторінку предмету переглянути інфформацію про нього та переглянути зробити пошук тестів, показано на рис. 3.11.

В тесті дозволено до 100 запитань, користувач повинен завершити тест до кінцевої дати, та не перевищуючи час виконання, які визначив адміністратор. Є декілька видів тестів: з однією правильною, відповіддю та

декількома. При приходженні також враховуються час неактивності (час коли користувач не був на сторінці здачі тесту), приймати чи не приймати результати тесту вирішує користувач, але рекомендується брати до розгляду тести з періодом неактивності до 45%.

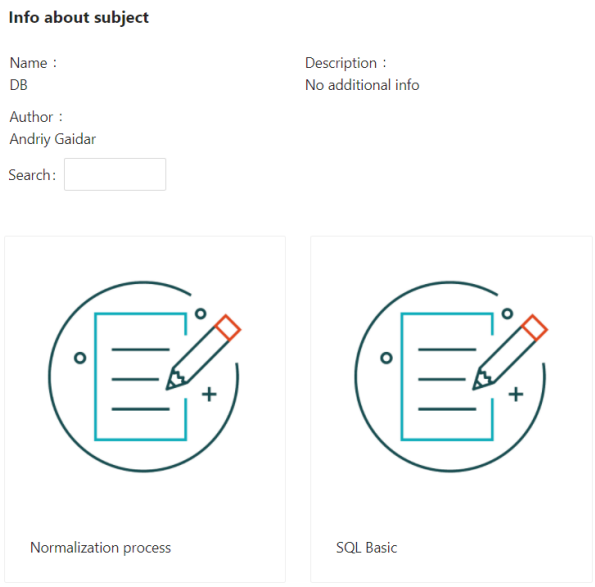

Рисунок 3.11 – Список тестів та загальна інформація про предмет

На рис. 3.12 показано інтерфейс здачі тесту, а на рис. 3.13 результату тесту.

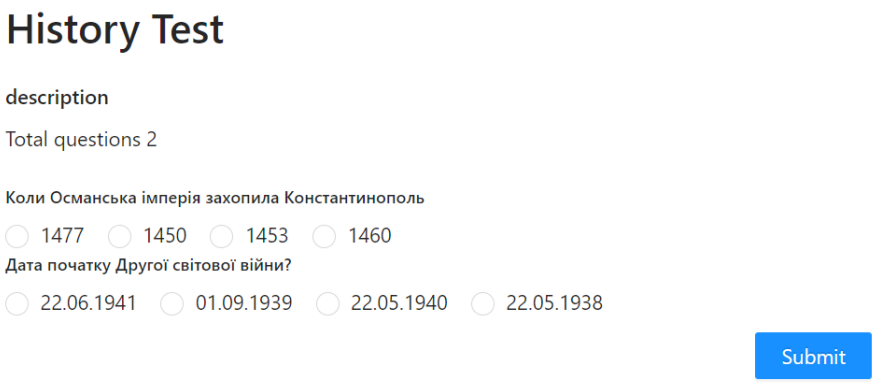

Рисунок 3.12 – Інтерфейс проходження тесту користувачем

Рисунок 3.13 – Інтерфейс з результатами тесту

Також в системі є функціонал профілю користувача (показано на рис. 3.14), воно дозволяє користувачу вказувати інформації про себе: ім’я, електронна пошта, короткі відомості про себе, посилання на персональний сайт або профіль в соцмережі, та місце проживання. В подальшому на основі інфрормації про користувача можна буде добавити статистику по населеному пункту, року народження тощо.

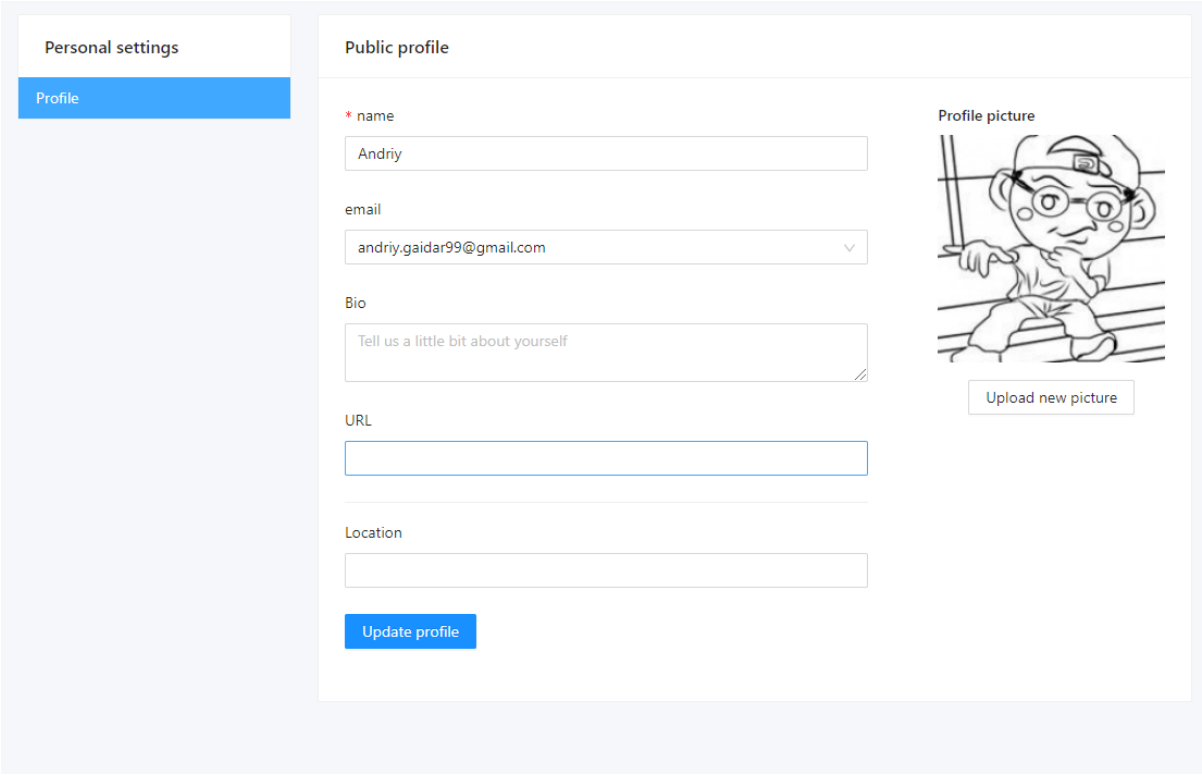

Рисунок 3.14 – Інтерфейс налаштувань користувача

3.3.2 Інтерфейс адміністратора

Адміністратор може виконувати всі ті самі дії що і користувач в тому числі проходити тести. Але на додаток до цього він має права на створення предметів, тестів, перегляду результатів та статистики. При створенні предмету достатньо заповнити назву, та його короткий опис, показано на рис. 3.15.

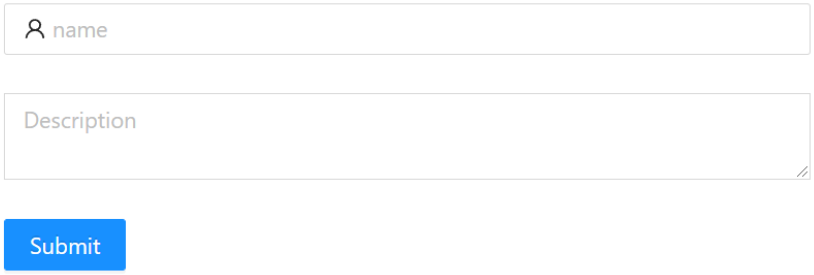

Рисунок 3.15 – Інтерфейс створення предмета

При створенні тесту адміністратор повинен заповнити дані, про кількість можливих спроб якщо поле не заповнене тоді користувачі можуть тест скільки завгодно раз (може бути корисно для тренувальних тестів), про час за який потрібно завершити тест, кінцевий час здачі тесту та інформацію про запитання (текст, тип правильна відповідь, варіанти тощо), показано на рис. 3.16.

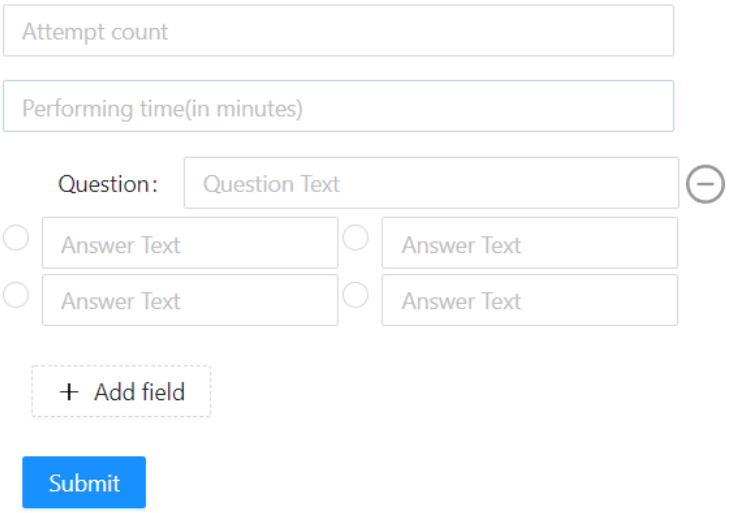

Рисунок. 3.16 – Інтерфейс створення тестів

Адміністратор має змогу побачити детальну статистику щодо здачі тестів (показано на рис. 3.17). Є такі види тестів :за користувачем, за тестом, за предметом та у всіх системі. Детальна статистика в свою чергу довзволяє виявити прогалини у знаннях щодо певних тем, наприклад: результати тестів про основи SQL мали в середньому менше полавини правильних відповідей, тому варто розглянути цю тему на практиці.

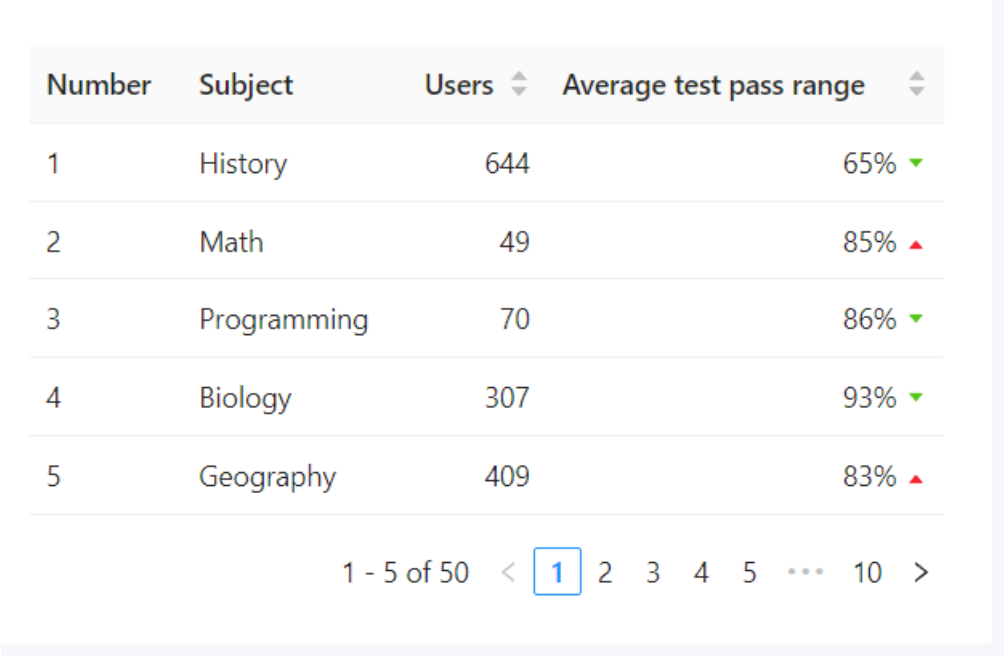

Рисунок 3.17 – Статистика по предметах

3.4 Висновки до третього розділу

В розділі проаналізовано нормалізацію баз даних, особливості нормальних форм, спроектовано та реалізовано базу даних для платформи відповідно до вимог. Реалізовано та продемонстровано функціонал на основі вибраних архітектуних підходів. Показано основні елементи інтерфейсу користувача та адміністратора.

В подальшому планується добавити функціонал авторизації через соціальні мережі (Linkedin, Facebook), “вагу запитання” в тесті, генерацію тесту в pdf-файл тощо.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Обов'язки керівників закладів освіти щодо організація роботи з охорони праці

Оскільки тема дипломної роботи присвячена розробці платформи перевірки знань шляхом тестування, та розробка пов’язана з навчальними закладами, то важливим є дотриманням норм охорони праці та безпеки в надзвичайних ситуаціях в навчальних закладах. На працівників освіти впливають такі самі фактори, що і в інших галузях, але й відповідно професійної діяльності можуть використовувати додаткове обладнання (комп’ютерну техніку, хімічні реагенти), які при неправильно чи надмірному використанні можуть бути шкідливі для здоров’я. Саме тому на керівників навчальних закладів покладають обов’язки передбачені в Положенні про організацію роботи з охорони праці в навчальному закладі. Відповідно до нього голова навчального закладу (ректор, директор, завідувач):

1) забезпечує створення безпечних умов для навчально-виховного процесу відповідно до законодавства про охорону праці та цих нормативних актів, не допускається проведення навчального процесу за наявності шкідливих та небезпечних умов;

2) призначає відповідальних за організацію роботи з охорони праці та безпеки життєдіяльності навчальних закладів, визначає їх обов'язки та забезпечує роботу закладу освіти. система управління охороною праці [44];

3) призначати відповідальними за охорону праці в структурних підрозділах, аудиторіях, лабораторіях, майстернях, спортзалах, тирах тощо згідно з наказами;

4) затверджує посадові інструкції керівників і працівників структурного підрозділу, порушує обов'язкові питання з охорони праці та безпеки життєдіяльності;

5) перед початком навчального року та протягом навчального року регулярно проводити оцінку технічного стану обладнання та устаткування навчального закладу, організовувати профілактичну роботу щодо запобігання травматизму учнів та працівників закладу освіти та зниження захворюваності;

6) вжити заходів щодо приведення інженерно-технічних комунікацій, обладнання та засобів у відповідність чинним нормам, законодавчим і нормативним актам з охорони праці;

7) укласти колективний договір (договір), що включає охорону праці та безпеку життєдіяльності, та забезпечити його виконання;

8) забезпечує дотримання вимог нормативно-правових актів з охорони праці, заходів з охорони праці, безпеки життєдіяльності, передбачених колективним договором (договорами), інструкцій органів державного нагляду за охороною праці, пропозицій виборного органу первинної торгівлі. спілки

9) організовує звітування про попередження травматизму, виконання заходів з охорони праці, колективних договорів (угод) з питань життєдіяльності та безпеки життєдіяльності та видає накази на засіданні педагогічної ради заклад;

10) організовує профілактичну роботу щодо попередження травматизму і зниження захворюваності серед здобувачів освіти та працівників закладів освіти;

11) організовує роботу з розробки складання вступних планів інструктажу та забезпечує проведення всіх видів інструктажів: з охорони праці -відповідно до Типового положення;

12) один раз на 5 років організовує роботу щодо розробки та періодичного перегляду: інструкцій з охорони праці для працівників [45];

13) сприяє громадському контролю за дотриманням законів та нормативних актів з охорони праці;;

14) відповідно до «Порядку забезпечення працівників спеціальним одягом та спецвзуттям» контролювати забезпечення учнів та працівників

навчальних закладів спецодягом, взуттям та іншими засобами індивідуального захисту;

15) контролює підготовку трудових учнівських колективів та студентських об’єднань до створення безпечних і нешкідливих умов праці та відпочинку;

16) сприяє проведенню дозиметричного контролю відповідно до нормативно-правових актів з обов'язковою реєстрацією в спеціальному журналі;

17) забезпечує навчання з питань охорони праці, безпеки життєдіяльності здобувачів освіти та працівників закладів освіти відповідно до законодавства і цього Положення;

18) сприяє виконанню організаційно-технічних заходів упровадження системи стандартів безпеки праці, проведенню атестації робочих місць за умовами праці;

19) організовує проведення обов'язкових попередніх та періодичних медичних оглядів працівників закладів освіти;

20) не дозволяє виконання робіт, які негативно впливають на здобувачів освіти і працівників закладів освіти та стан довкілля;

21) здійснює постійний зв'язок з органами виконавчої влади та громадськими об'єднаннями щодо запобігання травматизму серед здобувачів освіти та працівників закладів освіти;

22) при настанні під час освітнього процесу нещасного випадку вживає заходів, передбачених Положенням про порядок розслідування нещасних випадків [46].

Належне та відповідальне виконання обо’язків голови навчального закладу зменшує шкідливі фактори для фізичного та ментального здоров’я персоналу та учнів (студентів), запобігає нещасним випадкам.

4.2 Пожежна безпека в навчальних закладах

Відповідно до Правил пожежної безпеки України, затверджених постановою Міністерства освіти і науки від 15.08, та правил пожежної безпеки навчальних закладів України забезпечується пожежна безпека організацій і підприємств системи освіти України. 2016 № 974, зареєстрований в Міністерстві юстиції України 8 вересня 2016 року за номером 1229/29359.

Забезпечення пожежної безпеки в організаціях, на підприємствах системи освіти України здійснюється згідно з Відповідно з Правилами пожежної безпеки в Україні та Правилами пожежної безпеки для навчальних закладів та установ системи освіти України, затверджених наказом Міністерства освіти і науки України 15.08.2016 № 974, зареєстрованих в Міністерстві юстиції України 08.09.2016 за № 1229/29359, відбувається забезпечення пожежної безпеки в організаціях [47].

Основним завданням пожежної безпеки в навчальних закладах є захист і порятунок персоналу (дітей) від небезпечних пожежних факторів, які супроводжуються неконтрольованим горінням. При виникненні пожежі дії працівників, залучених до гасіння пожежі, повинні бути спрямовані на забезпечення безпеки особового складу, особливо дітей, а також їх евакуацію та рятування.

Усі заклади та установи перед початком навчального року мають бути затверджені відповідною комісією, у тому числі представниками органів державного нагляду у сфері пожежної безпеки.

Діти у будинках дитячих дошкільних закладів повинні розміщуватися з таким розрахунком, щоб молодші розташовувалися на нижчих поверхах [48].

У багатоповерхових навчальних корпусах та школах-інтернатах класи повинні розміщуватися на нижніх поверхах. У кімнатах з дітьми підлога повинна бути прикріплена до кріплення (за винятком дитячих садків), мати помірну здатність до димоутворення. У дитячих закладах, які працюють

цілодобово, літні дитячі дачі повинні бути оснащені чергуванням персоналу нічної служби. Зал очікування повинен забезпечувати телефонний зв'язок. Черговий повинен забезпечити: особовий склад на пожежі засобами індивідуального захисту органів дихання, комплектом ключів від евакуаційних дверей, переносним ліхтарем, а також знати кількість дітей, які ночують, їх місцезнаходження та зателефонувати до найближчого пожежно-рятувальної частини для передачі інформації.

У загальноосвітніх навчальних закладах (крім закладів для дітей з розумовими і фізичними вадами) можуть створюватися дружини молодих пожежників-рятувальників. У закладах та установах, де учні/першокласники проживають цілодобово, необхідно встановити обов'язки персоналу нічної служби, яка не має права спати під час зміни. Зал очікування повинен забезпечуватися телефонним зв'язком. Черговий персонал повинен окремо оснастити фільтруючими пристроями усіх дітей та обслуговуючий персонал на випадок пожежі, комплектом ключів від евакуаційних виходів і воріт, а також автомобільних під'їздів і установ для в'їзду на територію установи [49]. Не допускається в будівлях установ і в місцях:

- розміщувати людей на горищі та на поверсі (будівлі), не передбачаючи двох евакуаційних виходів;
- перепланувати ділянку без урахування будівельних норм і правил;
- установлювати ґрати та пристрої на вікнах приміщень де перебувають учасники навчально-виховного процесу, а саме: сходових клітках, у коридорах, холах та вестибюлях Якщо ґрати все таки встановлені (кабінет інформатики, інші приміщення з обладнанням, що має матеріальну цінність), вони повинні розсуватися, зніматися або розкриватися, під час перебування в цих приміщеннях вони мають бути відчиненими;
- знімати дверні полотна в отворах, що з'єднують коридори зі сходовими клітками, та двері евакуаційних виходів;

- використовувати для опалення нестандартні (саморобні) нагрівальні пристрої;
- користування прилади для приготування їжі, крім спеціально обладнаних приміщень;
- захаращувати шляхи евакуації;
- встановлення дзеркал та встановлення фальш-дверей на шлях евакуації;
- встановлювати перешкоди на шляху евакуації; пороги, виступи, поворотні двері, розсувні двері, підйомні двері та інші пристрої для евакуації;
- при наявності людей у будівлі проводити електрозварювання та інші види пожежонебезпечних робіт;
- використовувати для освітлення свічки та гасові лампи та ліхтарі;
- використовувати відкритий вогонь для нагрівання труб систем опалення, водопостачання, каналізації тощо (для цього використовується гаряча вода, пару або гарячий пісок);
- зберігати використані обтиральні матеріали на робочому місці, в шафах, зберігати їх у кишенях робочого одягу;
- підключати до джерела живлення електроприлади без нагляду [50].

Навіть в випадку виникнення пожежі виконання вимог дозволяє набагато легше проводити евакуацію та мінімізувати наслідки пожежі.

4.3 Висновки до четвертого розділу

Було проаналізовано та розглянуто обов'язки керівників закладів освіти та організація роботи з охорони праці і безпеки життєдіяльності в закладах освіти, що дало змогу краще мінімізувати негативний вплив робочих факторів на здоров’я людини. Також було розглянуто пожежну безпеку, та заборонені дії в навчальних закладах які можуть призвести до пожежі.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи розроблено платформу для перевірки знань шляхом тестування. В ході роботи:

1. Проведено аналіз та огляд відомих технічних рішень аналогічного призначення.

2. На основі отриманих результатів аналізу відомих рішень обгрунтовано вибір архітектури системи та інструментальних засобів для розробки платформи.

3. Розроблено прототип платформи, зокрема, структуру бази даних, клієнтську та сервеврну її частини.

4. Створено графічний веб-орієнтований інтерфейс користувача.

У розділі «Охорона праці та безпека в надзвичайних ситуаціях» проаналізовано обов'язки керівників закладів освіти та організацію роботи з охорони праці і безпеки життєдіяльності в закладах освіти. Також наведено вимоги та правила пожежної безпеки в навчальних закладах

Кваліфікаційна робота виконана у відповідності до завдання та вихідних даних до роботи.

ПЕРЕЛІК ДЖЕРЕЛ

1. Open Test Architecture [Електронний ресурс] – Режим доступу: URL: https://getopentest.org/docs/architecture.html.
2. Комп’ютерне тестування у системі OpenTEST 2 як форма оцінювання знань та вмінь студентів спеціальності "Інформаційна, бібліотечна та архівна справа" / М. А. Кирилов, Т. М. Трофімук-Кирилова, С. В. Чибирак // Інформаційні технології і засоби навчання. - 2018. - Т. 64, № 2. - с. 138.
3. Використання Google Classroom для організації дистанційного навчання учнів [Електронний ресурс] – Режим доступу: URL: https://zippo.net.ua/data/files/2020/methodical_work/dist_navch_phizik.pdf.
4. Glenn Page “GOOGLE CLASSROOM FOR TEACHERS: The Ultimate Guide to Digital Learning. A step-by-step approach to improve your teaching activities, enhance task management and get started with your online classroom” 2020. – с. 17.
5. Catherine “Korman Google Classroom for Teachers 2020: A Complete Guide to Learn Everything You Need to Know for Your Classroom Management” 2020. – 22 с.
6. Офіційний сайт Moodle [Електронний ресурс] – Режим доступу: URL: https://moodle.org/?lang=uk.
7. Alex Buchner “Moodle 3 Administration, 3rd Edition: An administrator's guide to configuring, securing, customizing, and extending Moodle” 2016. – 12 с.
8. “Moodle Course Design Best Practices: Design and develop outstanding Moodle learning experiences” 2018 – 43 с.
9. Система тестирования «indigo» [Електронний ресурс] – Режим доступу: URL: https://studfile.net/preview/5969210/page:4/.
10. Руководство пользователя по работе с программным продуктом«Система тестирования INDIGO» [Електронний ресурс] – Режим

доступу: URL: http://www.ael.ru/prepodavatelyu-i-sotrudniku/sistema-testirovaniya-indigo/INDIGO%20Manual%202.0%20RC.pdfop

11. ATutor LCMS [Електронний ресурс] – Режим доступу: URL: http://www.atutor.ca/.

12. Михеева, Е. В. Практикум по информационным технологиям в профессиональной деятельности: учебное пособие для средне профессионального образования/ Е.В. Михеева. - Москва: Издательский центр «Академия», 2007. – 137. стр.

13. Захарова, И.Р. Информационные технологии в образовании: учебное пособие/ И.Р. Захарова. - М.: Издательский центр «Академия», 2008. - 192 с.

14. Самсонов В.В., Єрохін А.Л. «Методи та засоби Інтернет-технологій» Х. : СМІТ, 2008. - 263 с.

15. Функциональные возможности современных систем автоматизации контроля качества обучения. сравнительный анализ [Електронний ресурс] – Режим доступу: URL: http://pp.isofts.kiev.ua/ojs1/article/viewFile/142/135.

16. Офіційна документація NodeJS [Електронний ресурс] – Режим доступу: URL: https://nodejs.org/uk/docs/.

17. Что такое Node JS [Електронний ресурс] – Режим доступу: URL: https://metanit.com/web/nodejs/1.1.php.

18. Node.js – Introduction [Електронний ресурс] – Режим доступу: URL: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

19. Офіційна документація NestJS [Електронний ресурс] – Режим доступу: URL: https://nestjs.com/.

20. Getting Started with NestJS [Електронний ресурс] – Режим доступу: URL: https://www.digitalocean.com/community/tutorials/getting-started-with-nestjs

21. JetBrains WebStorm – Опис [Електронний ресурс] —Режим доступу: URL: https://itpro.ua/product/jetbrains-webstorm/?tab=description.

22. Офіційна документація Postgres [Електронний ресурс] – Режим доступу: URL: https://www.postgresql.org.

23. R. O. Obe “PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database” 2017. – 111 с.

24. Офіційна документація pgAdmin [Електронний ресурс] – Режим доступу: URL:https://www.pgadmin.org/

25. Офіційна документація Angular [Електронний ресурс] – Режим доступу: URL: https://angular.io/.

26. Jeremy Wilken “Angular in Action” 2018 – 87 с.

27. Mathieu Nayrolles “Angular Design Patterns: Implement the Gang of Four patterns in your apps with Angular” 2018 – 65 с.

28. Морозов Ю.В., Пастернак І.І. / Модель об’єктної клієнт-серверної взаємодії // Вісник «Комп’ютерні науки та інформаційні технології».- Львів: НУ «Львівська політехніка», 2011.- №719.- С. 164-167.

29. Морозов Ю.В., Пастернак І.І. / Мережні інтерфейси рівня клієнт-сервер // Вісник «Інформаційні системи та мережі».- Львів: НУ «Львівська політехніка», 2012.- №743.- С. 121-131.

30. Understanding Onion Architecture [Електронний ресурс] – Режим доступу URL : https://www.codeguru.com/csharp/csharp/understanding-onion-architecture.html.

31. Stéphane Eyskens “Software Architecture for Busy Developers: Talk and act like a software architect in one weekend” 2021 – 43с.

32. Eric Normand “Grokking Simplicity: Taming complex software with functional thinking” 2021 – 76 с.

33. Introduction to Angular concepts [Електронний ресурс] – Режим доступу URL : https://angular.io/guide/architecture.

34. Mathieu Nayrolles, Rajesh Gunasundaram, Sridhar Rao “Expert Angular” 2017 – 18 с.

35. Рудикова, Л.В. Базы данных. Разработка приложений: учебно пособие/ Л.В. Рудикова. - СПб.: БХВ - Петербург, 2007. - 140 с.

36. Бураков, П.В. Введение в системы баз данных: учебное пособие/ П.В. Бураков. -СПбГУ ИТМО, 2010. - 129 с.

37. Пирогов, В. Ю. Информационные системы и базы данных: организация и проектирование: учебное пособие/ В. Ю. Пирогов. - БХВ - Петербург, 2009. - 243 с.

38. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу: URL: https://jwt.io/introduction.

39. Getting Started with ng-zorro [Електронний ресурс] – Режим доступу: URL: https://ng.ant.design/docs/getting-started/en.

40. What is an HttpOnly Cookie? [Електронний ресурс] – Режим доступу: URL: https://www.cookiepro.com/knowledge/httponly-cookie/

41. Instroduction to PassportJs [Електронний ресурс] – Режим доступу: URL: https://jwt.io/introduction. http://www.passportjs.org/docs/

42. Custom decorators [Електронний ресурс] – Режим доступу: URL: https://docs.nestjs.com/custom-decorators

43. Guards [Електронний ресурс] – Режим доступу: URL: https://docs.nestjs.com/guards

44. Про затвердження Положення про організацію роботи з охорони праці та безпеки життєдіяльності учасників освітнього процесу в установах і закладах освіти [Електронний ресурс] – Режим доступу: URL: https://rada.info/upload/users_files/26323918/e0ff5e06a735b05d8fe3e9af33399bc6.pdf.

45. Про затвердження Типового положення про службу охорони праці [Електронний ресурс] – Режим доступу: URL: https://zakon.rada.gov.ua/laws/show/z1526-04#Text.

46. Положення про розробку інструкцій з охорони праці [Електронний ресурс] – Режим доступу: URL: https://zakon.rada.gov.ua/laws/show/z0226-98#n30.

47. Про затвердження Правил пожежної безпеки для навчальних закладів та установ системи освіти України [Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/z1229-16#Text

48. Пожежна безпека у навчальних закладах [Електронний ресурс] – Режим доступу: URL: https://ns-plus.com.ua/2020/01/08/zabezpechennya-pozhezhnoyi-bezpeky-u-shkolah-ta-dytyachyh-doshkilnyh-zakladah-2/?fbclid=IwAR2HTDXPIkKj2RqWteCki3QZ2x_7bUWddGkx9Z1yR1pYLLzHJ7c3qsVObZY/.

49. Протипожежна безпека в закладах освіти [Електронний ресурс] – Режим доступу: URL:https://rozvytok-osvity.te.ua/123342436-2/

50. Пожежна безпека у закладах освіти: рекомендації https://www.auc.org.ua/novyna/pozhezhna-bezpeka-u-zakladah-osvity-rekomendaciyi-eksperta

ДОДАТКИ

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

ІХ НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

А1

УДК 378.147
А.В. Гайдар, В.А. Готович, канд. техн. наук
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗРОБКА ПЛАТФОРМИ ДЛЯ ПЕРЕВІРКИ ЗНАНЬ ШЛЯХОМ ТЕСТУВАННЯ

UDC 378.147
A.V. Haidar, V.A. Hotovych, Ph.D

DEVELOPMENT OF PLATFORMS FOR VERIFICATION OF KNOWLEDGE THROUGH TESTING

Задача організації та контролю засвоєння знань як результат навчального процесу є не менш важливою як задача організації самого навчального процесу. Існує багато форм контролю, але найчастіше використовуються письмова або усна форми. На жаль, кожна з них не позбавлена недоліків. При проведенні усного опитування витрачається відносно велика кількість часу, пропорційна до кількості проведених опитувань, при виконанні письмових завдань – багато часу витрачається на ручну перевірку [1].

Все частіше для контролю знань, як підсумкового так і поточного, використовується тестування. Однією з його головних і незаперечних переваг є мінімальні витрати часу на отримання достовірних результатів. Для тестування використовуються як традиційні засоби (папір та ручка), так і засоби автоматизації (комп'ютерні системи). Важливими перевагами засобів автоматизації є можливість дуже швидкого отримання результатів а також одночасної перевірки знань багатьох здобувачів [2, 3].

Існує безліч систем тестування, але більшість з них мають ряд недоліків: платна ліцензія, незручний інтерфейс, складність при встановленні програмного забезпечення, відсутня перевірка на "чесність". В доповіді пропонується проект платформи для автоматизації процесу перевірки знань шляхом тестування у вигляді веб-додатку, яка позбавлена вказаних недоліків.

Для створення платформи обрано трирівневу клієнт-серверну (багатошарову) архітектуру та компонентну модель з використання принципу інверсії залежностей. Використано наступні засоби проектування: платформа NodeJS, СУБД PostgresSQL, pgAdmin, середовище IDE WebStorm, фреймворки NestJS та Angular.

Пропонована платформа містить наступний функціонал: реєстрація користувачів, аутентифікація та авторизація, розділення прав доступу користувачів на основі ролей студента та адміністратора, налаштування тестів (створення, редагування та видалення предметів, створення та редагування тестів, можливіть задати максимальну кількість спроб для тесту, вибрати кінцевий термін здачі тесту, час для його проходження). Також реалізовано функціонал перевірки на "неактивність" користувача та збір детальної статистики для адміністратора (загальна по платформі, по певному користувачу, по певному тесту, по навчальному предмету тощо).

На даний момент проводиться тестування по відловлюванню помилок, навантажувальне тестування (кількість запитів за одну секунду).

Література.

1. Михеева, Е. В. Практикум по информационным технологиям в профессиональной деятельности: учебное пособие для средне профессионального образования/ Е.В. Михеева. – Москва : Издательский центр «Академия», 2007. – 137 с.
2. Захарова, И. Р. Информационные технологии в образовании: учебное пособие/ И.Р. Захарова. – М.: Издательский центр «Академия», 2008. – 192 с.
3. Дворецкая А. В. Основные типы компьютерных средств обучения // Педагогические технологии. – 2004. – № 2.

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей
Том І

X Міжнародної науково-практичної конференції молодих учених та студентів
24-25 листопада 2021 року

УКРАЇНА
ТЕРНОПІЛЬ – 2021

СЕКЦІЯ: КОМПЮТЕРНО-ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ЗВ'ЯЗКУ

Б2

УДК 378.147

А. В. Гайдар, В. А. Готович, канд. техн. наук
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ЗАСТОСУВАННЯ КОМП'ЮТЕРНО-ІНФОРМАЦІЙНИХ ЗАСОБІВ В ПРОЦЕСІ НАВЧАННЯ

A.V. Haidar, V. A. Hotovych, Ph.D

APPLICATION OF COMPUTER INFORMATION AIDS IN THE PROCESS OF EDUCATION

Науково-технічний прогрес та розвиток технологій не оминули сектору освіти. Інформаційні технології (ІТ) підвищили доступність та ефективність надання освітніх послуг [1]. До основних переваг застосування ІТ в освіті належать:

1. Спрощений доступ до навчальних матеріалів. За допомогою мережі Інтернет здобувачі мають змогу користуватися необхідними для навчання матеріалами (електронні книги, посібники, засоби для перевірки та контролю знань тощо);
2. Безперервність процесу навчання. Здобувачі можуть навчатися в зручний для них час та в умовах гнучкого графіку;
3. Обмін знаннями та неформальне спілкування в позаурочний час. Різноманітні онлайн-засоби (форуми, блоги, спільноти, чати, вебінари) дозволяють обговорювати навчальний матеріал та отримані завдання, брати участь в різного роду дискусіях, розширювати світогляд, розвиватися як в інтелектуальній так і в культурній сфері;
4. Використання широкого спектру звукового та візуального (мультимедійного) матеріалу, що зумовлює підвищення ефективності та доступності подачі матеріалу від вчителя (викладача) до учня (студента) [2];
5. Технології дистанційного навчання, які дозволяють здобувати освіту віддалено, навіть за кордоном, без необхідності регулярної фізичної присутності здобувача освіти в начальному закладі;
6. Належне ведення обліку, до якого належить контроль за своєчасністю та успішністю виконання здобувачами завдань та облік відвідування занять.

Важливе місце серед ІТ в освіті посідають засоби для контролю знань, в тому числі дистанційного, як інструмент контролю за процесом засвоєння знань (поточний контроль) та підсумкового контролю отриманих знань в результаті проходження навчальної програми (курсу, семестру, модуля).

В даній роботі пропонується проект програмної платформи для перевірки знань шляхом тестування. Дана платформа розроблена з використання технологій: NodeJS, СУБД PostgresSQL, та фреймворками NestJS і Angular. Вона відрізняється від більшості інших: безкоштовною ліцензією, web-інтерфейсом (не потрібно розгортати сервери), зручнішим інтерфейсом, та кращою перевіркою під час складанням тесту.

На основі досліджень доведено, що інформаційні технології мають позитивний вплив на процес навчання, це здешевлює навчання, дозволяє проводити його в ігровій формі тощо. Але виключно дистанційне навчання має ряд недоліків: низька мотивація студентів, відсутністю "живого" спілкування, та недостатнім захистом від списування. Тому рекомендується: поєднувати дистанційне навчання з очним, та використовувати ПЗ з більш суворим контролем під час складання тесту.

Література:

1. Анисимов П.Ф. Новые информационные и образовательные технологии как фактор модернизации учебного заведения // СПО. - 2004. - №6., С. 2
2. Дворецкая А.В. Основные типы компьютерных средств обучения // Педагогические технологии. - 2004. - №2.

ДОДАТОК В

Лістинг В.1 – розмітка сторінки логіну

```
<div class="contailer">
  <form nz-form [formGroup]="validateForm" class="login-form"
(ngSubmit)="submitForm()">
    <nz-form-item>
      <nz-form-control nzErrorTip="Please input your email!">
        <nz-input-group nzPrefixIcon="user">
          <input type="text" nz-input formControlName="email"
placeholder="Username"/>
        </nz-input-group>
      </nz-form-control>
    </nz-form-item>
    <nz-form-item>
      <nz-form-control nzErrorTip="Please input your Password!">
        <nz-input-group nzPrefixIcon="lock">
          <input type="password" nz-input
formControlName="password" placeholder="Password"/>
        </nz-input-group>
      </nz-form-control>
    </nz-form-item>
    <button nz-button class="login-form-button login-form-
margin" [nzType]="'primary'">Log in</button>
    Or <a routerLink="../register"> register now! </a>
  </form>
</div>
```

Лістинг В.2 – розмітка сторінки реєстрації

```
<form nz-form [formGroup]="validateForm" class="login-form"
(ngSubmit)="submitForm()">
  <nz-form-item>
    <nz-form-control nzErrorTip="Please input your email!">
      <nz-input-group nzPrefixIcon="user">
        <input type="text" nz-input formControlName="email"
placeholder="Username" />
      </nz-input-group>
    </nz-form-control>
  </nz-form-item>
  <nz-form-item>
    <nz-form-control nzErrorTip="Please input your Password!">
      <nz-input-group nzPrefixIcon="lock">
        <input type="password" nz-input
formControlName="password" placeholder="Password" />
      </nz-input-group>
    </nz-form-control>
  </nz-form-item>  <button nz-button class="login-form-button
login-form-margin" [nzType]="'primary'">Register</button>  Or <a
routerLink="../login"> login now! </a></form>
```

ДОДАТОК Г

Лістинг Г.1 – програмний код з файлу quiz.service.ts

```
@Injectable()
export class QuizService implements CrudService<QuizDto> {
  constructor(@InjectRepository(QuizEntity) private quizRepo:
Repository<QuizEntity>,
              @InjectRepository(QuestionEntity) private
questionRepo: Repository<QuestionEntity>,
              @InjectRepository(AttemptEntity) private
attemptRepo: Repository<AttemptEntity>,
              @InjectRepository(SubjectEntity) private
subjectRepository: Repository<SubjectEntity>) {
  }

  async createItem(data: QuizDto): Promise<QuizDto> {
    const subject = await this.subjectRepository.findOne({
where: { id: data.subjectId } });
    const { questions, ...quizDTO } = data;
    const quiz = this.quizRepo.create({ ...quizDTO, author:
data.author });
    quiz.questions = [];
    quiz.subject = subject;
    for (const questionDTO of questions) {
      const question = this.questionRepo.create(questionDTO);
      await question.save();
      quiz.questions.push(question);
    }
    await quiz.save();
    return quiz as any;
  }

  async getItems(params: { subjectId: Id }) {
    const data = await this.quizRepo.find(
      {
        where: {
          subject: {
            id: params.subjectId,
          },
        },
        relations: ['subject'],
      },
    );
    return { data } as any;
  }

  async getBySubject(subjectId: number) {
    return await this.quizRepo.find({
      where: {
        subject: {
          id: subjectId,
```

```
        },
      },
      relations: ['subject'],
    });
  }

  async calculate(body) {
    const { id, answers } = body;
    const attempt = await this.getAttempt(id);
    const quiz = attempt.quiz;
    const deadline = (new Date(attempt.createdAt).getTime() -
      new Date(attempt.createdAt).getTimezoneOffset()) +
(quiz.performingTime * 60 * 1000);
    const currentTime = new Date().getTime() + (new
Date().getTimezoneOffset() * 60 * 1000);
    if (currentTime > deadline) {
      throw new ForbiddenException('Time for this attempt is
already out', `Deadline was ${ new Date(deadline) },
      now is ${ new Date() }`);
    }
    if (attempt.finished) {
      throw new ForbiddenException('Attempt is already
finished.');
    }
    const score = quiz.questions.reduce((score, currentValue,
index) => {
      if (currentValue.type === QuestionType.Single) {
        const answer = answers[index];
        return currentValue.correctIndexes.map(item =>
+item).includes(answer) ? score + 1 : score;
      } else {
        return _.isEqual(currentValue.correctIndexes.map(item =>
+item), answers[index]) ? score + 1 : score;
      }
    }, 0);
    attempt.finished = true;
    attempt.score = score;
    attempt.maxScore = quiz.questions.length;
    await attempt.save();
    return { score, total: quiz.questions.length };
  }

  async getOverallStatistics() {
    const attempts = await this.attemptRepo.find({
      where: {
        finished: true,
      },
    });
    const allScore = attempts.reduce((total, item) => {
      return total + item.score / item.maxScore;
    }, 0);
    return allScore / attempts.length;
```

```
  }

  async getStatisticsBySubject(subjectId: SubjectEntity) {
    const attempts = await this.attemptRepo.find({
      where: {
        finished: true,
        'quiz.subject': { id: subjectId },
      },
      relations: 'quizzes',
    } as any);
    const allScore = attempts.reduce((total, item) => {
      return total + item.score / item.maxScore;
    }, 0);
    return allScore / attempts.length;
  }

  async getStatisticsByUser(userId: SubjectEntity) {
    const attempts = await this.attemptRepo.find({ where: {
finished: true, user: userId, }, } as any);
    const allScore = attempts.reduce((total, item) => {
      return total + item.score / item.maxScore;
    }, 0);
    return allScore / attempts.length;
  }
  async fullStats() {
    const attempts = await this.attemptRepo.query('SELECT * FROM
attempts GROUP BY quiz_id');
    const allScore = attempts.reduce((total, item) => {
      return total + item.score / item.maxScore;
    }, 0);
    const quizStats = allScore / attempts.length;
    const subjectAttempts = await this.attemptRepo.query('SELECT
* FROM attempts GROUP BY subject_id');
    const subjectScore = subjectAttempts.reduce((total, item) =>
{
      return total + item.score / item.maxScore;
    }, 0);
    const subjectStats = subjectScore / subjectAttempts.length;
    const userAttempts = await this.attemptRepo.query('SELECT *
FROM attempts GROUP BY user_id');
    const userScore = subjectAttempts.reduce((total, item) => {
      return total + item.score / item.maxScore;
    }, 0);
    const userStats = userScore / userAttempts.length;
    return {
      userStats,
      subjectStats,
      quizStats,
    };
  }
  async getItem(id: number) {
    return await this.quizRepo.findOne({
```

```
      where: {
        id,
      },
    }) as any;
  }
  async startQuiz(id: number, user: UserEntity) {
    const attemptCount = await this.attemptRepo.count({
      where: {
        user: {
          id: user.id,
        },
        quiz: {
          id,
        },
      },
    });

    const quiz = await this.quizRepo.findOne({
      where: {
        id,
      },
    });

      if (attemptCount >= quiz.maxAttempts) {
        throw new ForbiddenException('No more attempts for this
quiz is allowed');
      }

    const attempt: AttemptEntity = await
this.attemptRepo.create();
    attempt.user = user;
    attempt.quiz = quiz;
    await attempt.save();
    return attempt;
  }

  async getAttempt(id) {
    const attempt = await this.attemptRepo.findOne({
      where: { id },
      relations: ['quiz', 'quiz.questions'],
    });
    console.log(attempt);
    return attempt;
  }

  deleteById(id: Id, user: UserDto): Promise<boolean> {
    return Promise.resolve(false);
  }

  updateItem(item: Partial<QuizDto>): Promise<QuizDto> {
    return Promise.resolve(undefined);
  }}
```

Лістинг Г.2 – програмний код з файлу subject.service.ts

```
@Injectable()
export class SubjectsService implements
CrudService<SubjectEntity> {
  constructor(@InjectRepository(SubjectEntity) private
subjectRepo: Repository<SubjectEntity>) {
  }

  async getItems(params = {}) {
    const data = await this.subjectRepo.find();
    return { data };
  }

  async getItem(id: number) {
    return await this.subjectRepo.findOne({ where: { id },
relations: ['author'] });
  }

  async deleteById(id: number, user: UserDto) {
    return await this.subjectRepo.delete({ id }).then(() =>
true).catch(() => false);
  }

  async updateItem(subject: SubjectEntity) {
    return this.subjectRepo.update({ id: subject.id }, subject)
as any;
  }

  getCreatedByMe(user: UserEntity) {
    return this.subjectRepo.find({
      where: {
        author: user,
      },
    });
  }

  getMy(user: UserEntity) {
    return this.subjectRepo.find({
      where: {
        students: user,
      },
      relations: ['users'],
    });
  }

  async createItem(subjectData: any) {
    const data = { ...subjectData, author: subjectData.author
as SubjectEntity;
    const subject = await this.subjectRepo.create(data);
    await subject?.save();
```

```
    return subject;
  }

  async subscribe(id: number, user: UserEntity) {
    const subject = await this.subjectRepo.findOne({ where: { id
}, relations: ['students'] });
    if (subject.students.map(item => item.id).includes(user.id))
      throw new ForbiddenException('You are already subscribed
to to this subject');
    subject.students.push(user);
    await subject.save();
    return subject;
  }

  async unsubscribe(id: number, user: UserEntity) {
    const subject = await this.subjectRepo.findOne({ where: { id
}, relations: ['students'] });
    subject.students = subject.students.filter(item => item.id
!== user.id);
    await subject.save();
    return subject;
  }
}
```

Лістинг Г.3 – програмний код з файлу auth.service.ts

```
@Injectable()
export class AuthService {
  constructor(@InjectRepository(UserEntity) private userRepo:
Repository<UserEntity>,
              private jwtService: JwtService) {
  }

  async register(data: RegisterDTO) {
    try {
      const user = this.userRepo.create(data);
      await user.save();
      return user.toJson();
    } catch (err) {
      if (err.code === '23505')
        throw new ConflictException('Email is already taken');

      throw new InternalServerErrorException();
    }
  }

  async login(data: LoginDTO) {
    try {
      const user = await this.userRepo.findOne({
        where: { email: data.email },
      });
```

```
      if (user && (await user.comparePassword(data.password)), ι
        const token = this.jwtService.sign({ id: user.id });
        const expires = new Date( Date.now() +
(+process.env.EXPIRES_IN) );
        return { user: user.toJson(), token, expires };
      } else
        throw new UnauthorizedException('Invalid credentials');
    } catch (err) {
      throw new UnauthorizedException('Invalid credentials');
    }
  }
}
```