Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Програмна система тестування студентів
з можливістю імпорту тестових завдань з інших форматів
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДПІПЗ.170103.01.03.ПЗ

Виконав студент IV курсу група ІПЗ-17-1 Дзюрбан / Підпис / Е. С. Дзюрбан / Ініціали, прізвище

Керівник канд. техн. наук, доцент / Науковий ступінь, звання / Підпис / Г. І. Радельчук / Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент / Підпис / Г. І. Радельчук / Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення / Підпис / Л. П. Бедратюк / Ініціали, прізвище

15 06 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
5 . 02 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Дзюрбану Едуарду Станіславовичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів
Керівник проєкту (роботи) Радельчук Галина Іванівна
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

кандидат технічних наук, доцент
Затверджена наказом ректора університету від 05.02.2021 р. № 11
2. Строк подання студентом проєкту (роботи) на кафедру 01.06.2021 р.
3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмної системи, програмна реалізація, тестування програми
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 11 шт)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Радельчук Г. І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання « 05 » лютого 2021р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2020	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2021	
3 Проектування програмного забезпечення	01.02 – 28.02.2021	
4 Програмна реалізація	01.03 – 10.04.2021	
5 Тестування програмного забезпечення	11.04 – 30.04.2021	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2021	
7 Попередній захист ДП	Травень 2021 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки)	26.05 – 30.05.2021	
9 Підготовка до захисту та захист ДП	з 01.06.2021	

Студент _____ Е. С. Дзюрбан
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Г. І. Радельчук
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів.

Автор проекту: Дзюрбан Едуард Станіславович.

Керівник проекту: Радельчук Галина Іванівна.

Пояснювальна записка: 67 с., 33 рис., 4 табл., 3 дод., 17 джерел.

Графічна частина: 14 презентаційних слайдів.

ПРОГРАМНА СИСТЕМА, ТЕСТУВАННЯ, ТЕСТ, ОЦІНКА, .NET Core, C#, MS SQL, MVC.

Метою проекту є створення програмної системи тестування студентів з можливістю імпорту тестових завдань з інших форматів.

У дипломному проекті проведено аналіз предметної області, наявного схожого програмного забезпечення, виділено їх переваги та недоліки. Виконано порівняння різних архітектурних підходів до створення програмних систем та вибір найоптимальнішого для вирішення поставлених задач. Спроектовано серверну та клієнтську частину додатку.

Для розробки програмної системи використано платформу .NET Core з мовою програмування C#. Для створення інтерфейсу користувача використано фреймворк Angular з бібліотекою MaterialAngular. У ролі системи керування базою даних було використано MS SQL.

У результаті проектування здійснена програмна реалізація програмної системи тестування студентів з можливістю імпорту тестових завдань з інших форматів.

01.06.2021 р.
Дата

Дзюрбан
Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПШПЗ.170103.01.03.ПЗ	Пояснювальна записка	67		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	14		

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.170103.01.03.ВД		
Виконав		Дзюрбан Е.С..	Дзюрбан	14.06	Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів. Відомість документів	Літ.	Арк. 1 / Акрушів 1
Керівник		Радельчук Г.І.		14.06			
Н. Контр.		Радельчук Г.І.		15.06		ХНУ, ІПЗ-17-1	
Зав. Каф.		Бедратюк Л.П.		15.06			

ЗМІСТ

ВСТУП

Світові тенденції у оцінюванні знань учнів та студентів все більше схиляються до об'єктивнішого методу – тестування. Оскільки при такому виді контролю знань виключений вплив суб'єктивних чинників, таких як обізнаність екзаменатора про поточну успішність студента чи облік його поведінки на учбових заняттях. Також важливим фактором є те, що результат достовірний і надійний – тестова оцінка є однозначною, та добре відображає об'єм та рівень засвоєння матеріалу.

Враховуючи епідеміологічну ситуації в Україні та світі, навчальні заклади масово переходять на дистанційну форму навчання, а отже і контроль знань відбувається дистанційно. Викладачі змушені шукати різні способи комунікації із учнями, та, що найголовніше – способи контролю та оцінки їх знань. Адже тепер немає можливості зібрати їх усіх у одній аудиторії та написати тест на листі паперу.

Підсумовуючи сказане, можна зробити висновок, що подана тема на сьогодні є, як ніколи актуальною, а її вирішення – затребуваним. З відкритих засобів, які є доволі популярними у викладачів та дозволяють так чи інакше створити щось на кшталт тестів для перевірки знань, немає таких, які були б максимально зручними та ефективними. Деякі з них не обладнані автоматизованою перевіркою правильності відповіді, інші мають доволі незручний інтерфейс створення тестів, треті ж являють собою комп'ютерну програму, для користування якою потрібна її інсталяція. Тому пропонується реалізувати програмну систему тестування та перевірки знань, яка б була максимально простою для користувачів, і в той же момент – високоефективною.

Прогрес не стоїть на місці, а розвиток веб-технологій суттєво вплинув на уявлення людей про роботу з комп'ютером та інформаційними системами. Сьогодні все більше відмовляються від комп'ютерних програм, які потрібно інсталювати, а надають перевагу веб-застосункам, для доступу до яких потрібен

лише браузер, котрий за замовчуванням є на кожному комп’ютері. Саме тому реалізована програмна система буде у вигляді веб-додатку, для користуванням яким користувачу потрібен лише доступ до мережі Інтернет.

Проте, до цього моменту, на різноманітних платформах користувачами уже створена і наявна доволі велика кількість тестів, перенесення яких вручну у нову систему була б марудною і довгою справою. Саме тому, реалізована система буде мати можливість імпорту тестів з найпопулярніших форматів, щоб пришвидшити і облегшити роботу користувачам.

Метою проекту є створення програмної системи, яка б дозволяла швидко та зручно створювати нові тести, або ж імпортувати їх з інших форматів; доступ до якої був би швидким та легким, а інтерфейс інтуїтивно зрозумілим та простим.

Основними завданнями при виконанні дипломного проекту є:

- визначення та аналіз особливостей предметної області, її суті та основних проблем;
- аналіз наявних на ринку програмних засобів та платформ відповідно предметної області;
- проектування та реалізація програмної системи, яка б відповідала основним вимогам та вирішувала основні проблеми;
- тестування реалізованої системи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Веб-застосунок для тестування студентів – це програмна система, яка дозволяє викладачам швидко та зручно створювати тести та давати на проходження його студентам.

Тестування, як таке – високоефективний та об’єктивний спосіб оцінки та контролю знань студентів. В той же час він є доволі простим у реалізації та виконанні. Проходження тестування займає короткий час, його можна проводити як індивідуально, так і для груп студентів. Результати тестування є однозначними, тому їх можна систематизувати та вивести статистику по них.

До недавнього часу більша частина тестування проводилася безпосередньо вживу, на аркушах паперу. Проте такий підхід є малоефективним, так як кожен результат доводиться перевіряти окремо, а процес систематизації результатів не є автоматизованим, не говорячи вже про виведення будь-якої статистики.

Тому на допомогу нам приходять інформаційні технології, які дозволяють автоматизувати та ще більше спростити цей процес. Вони дозволять оцінку та контроль знань студентів вивести на новий рівень. А в умовах нинішньої епідеміологічної ситуації, коли дистанційне навчання – невід’ємна частина освітніх процесів у всьому світі, це вирішить низку проблем та дозволить максимально спростити і облегшити сам процес перевірки знань.

Основні проблеми пов’язані з перевіркою знань студентів:

- проблема зручного механізму донесення завдань тестування та збору результатів (відповідей) студентів;
- проблема великої витрати часу на перевірку отриманих результатів та визначення тестової оцінки;
- проблема систематизації результатів за виведення статистики.

Створювана програмна система призвана вирішувати дані проблеми, значно спрощуючи процес перевірки знань студентів, в той же час роблячи його більш ефективним та менш затратним по часу.

Для опису програмної системи буде використано спрощену DFD та функціональну діаграми. Вони відображатимуть основні потоки даних та процеси в системі.

На рисунку 1.1 зображено діаграму потоків даних верхнього рівня. Як бачимо, дані у систему тестування надходять від двох об’єктів – студента та викладача, і в результаті використовуються самою системою засобом запису їх в базу даних.

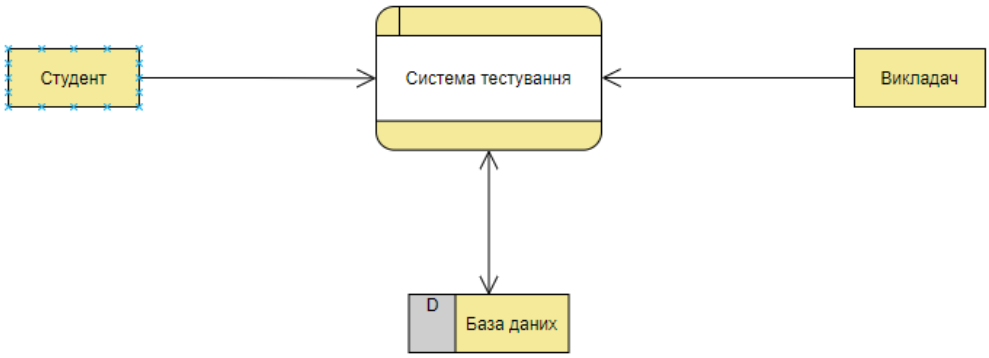


Рисунок 1.1 – DFD діаграма верхнього рівня

На наступних рисунках буде зображено діаграми потоків даних та функціональні діаграми для викладача (рисунки 1.2 та 1.5) та студента (рисунки 1.3 та 1.6) відповідно.

Як бачимо, викладач буде проходити процес авторизації та матиме можливість створювати тести, або ж переглядати оцінки по пройденим тестам. Створені тести будуть записані у базу даних та доступні для проходження студентам. Для перегляду результатів тестування студентів із бази даних для викладача будуть вивантажуватися оцінки.

Студент у свою чергу також проходитиме процес авторизації, після чого зможе вибрати тест для проходження із доступних, матиме можливість ввести

свої відповіді. Після введення всіх відповідей та підтвердження тест відправляється на перевірку. Система автоматично перевірить та виставить оцінку за пройдений тест. Після цього, результати проходження тесту будуть записані у базу даних.

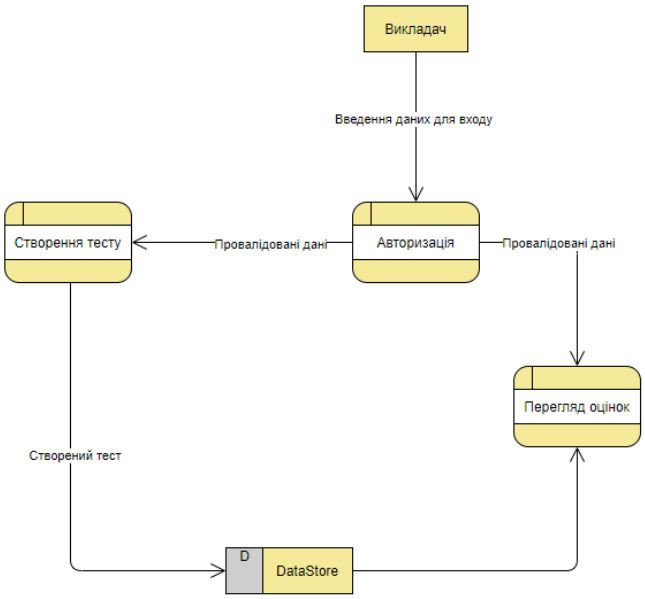


Рисунок 1.2 – DFD діаграма системи з боку викладача

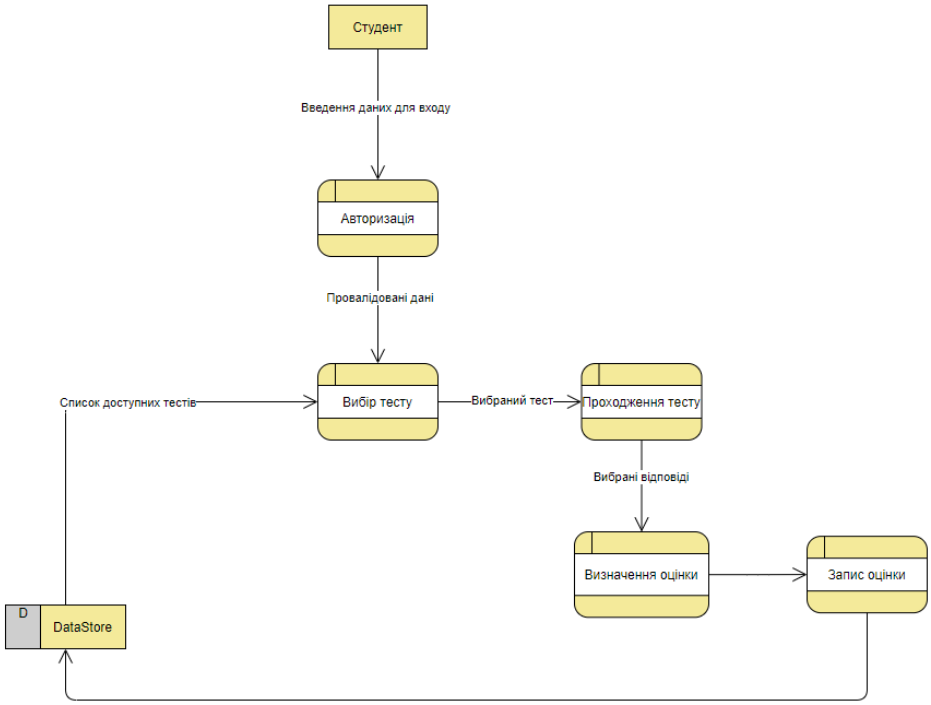


Рисунок 1.3 – DFD діаграма системи з боку студента

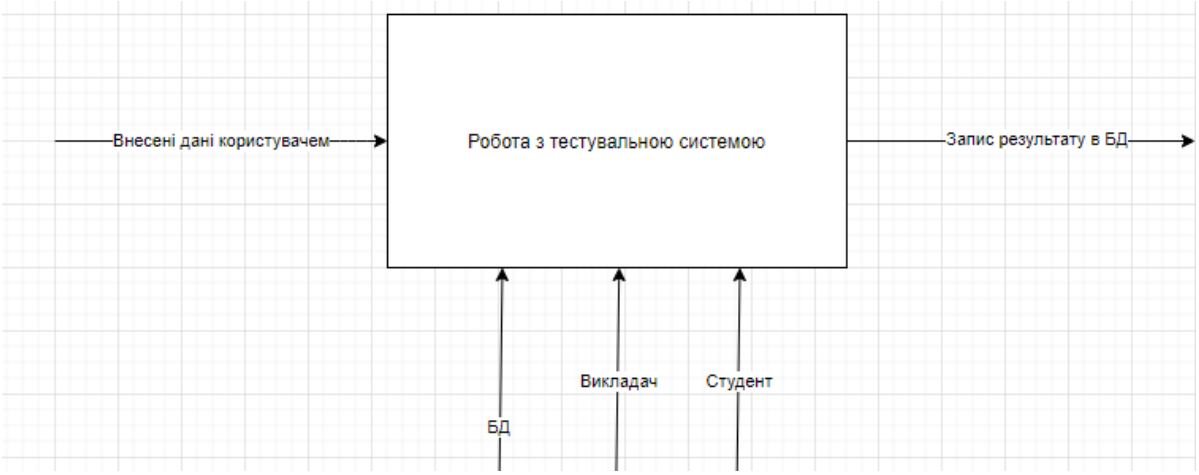


Рисунок 1.4 – Функціональна діаграма верхнього рівня

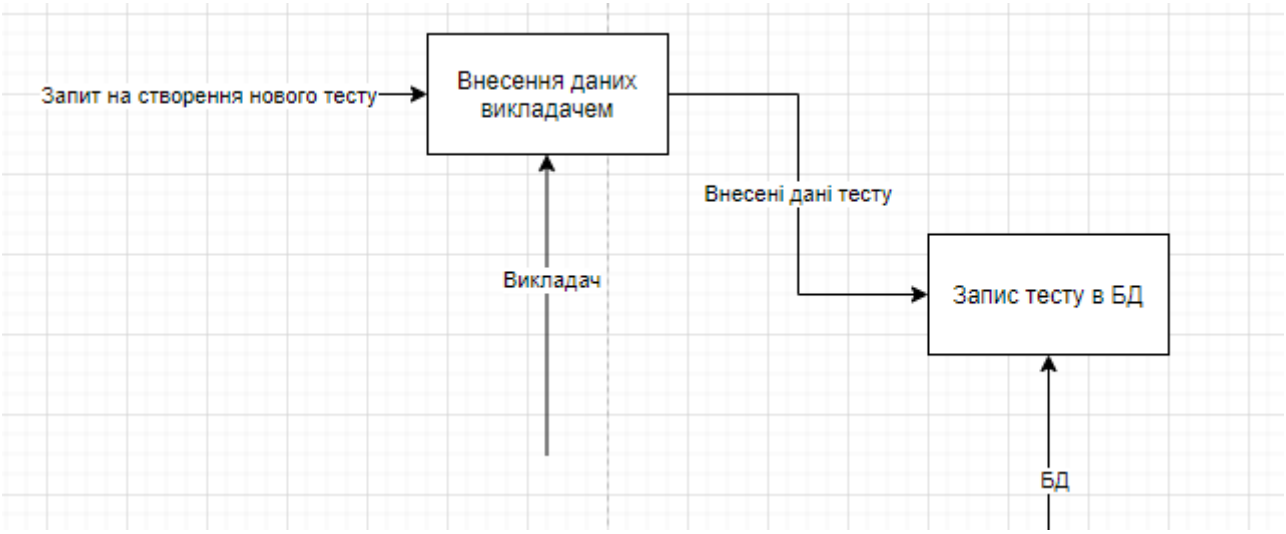


Рисунок 1.5 – Декомпозована діаграма для викладача

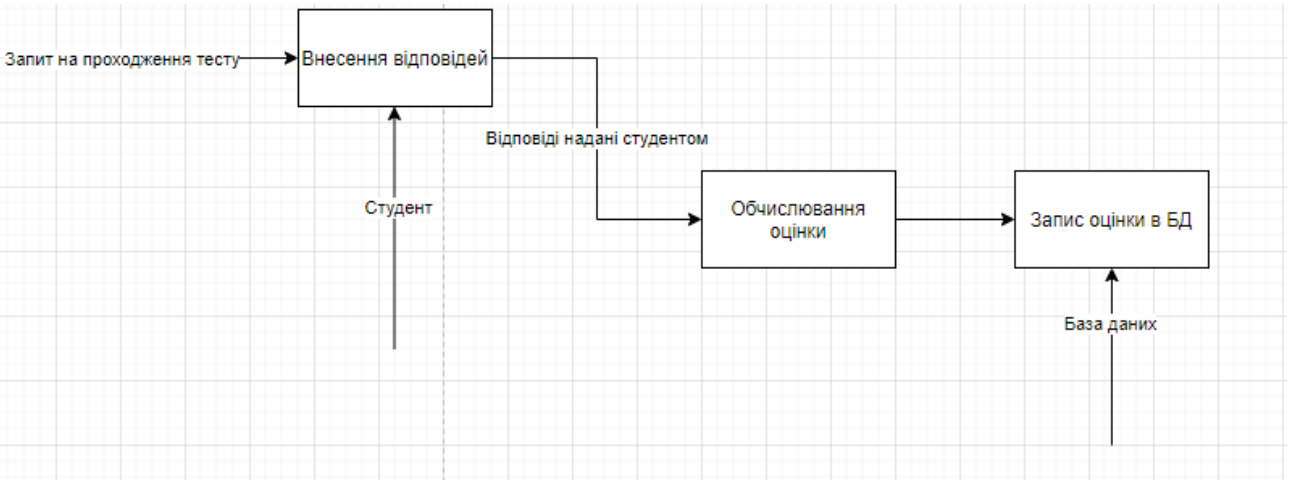


Рисунок 1.6 – Декомпозована діаграма для викладача

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

В області відкритих платформ для створення тестів доволі мало проектів, які б відповідали критеріям – швидко, доступно та ефективно. Найпопулярнішими та найчастіше використовуваними є Google Форми (рисунок 1.7). Це веб-застосунок (тому доступ до нього легкий і швидкий, варто лише авторизуватися через аккаунт Google), частина пакету інструментарію Google Drive.

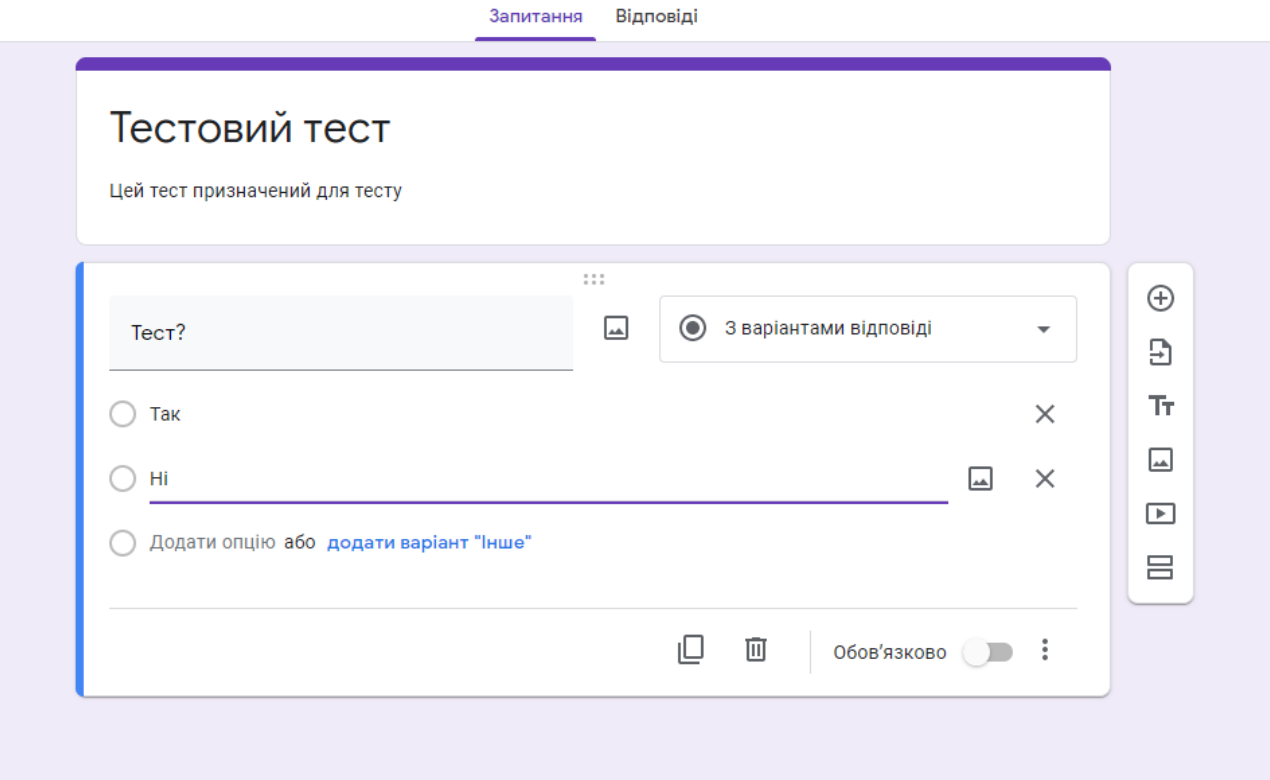


Рисунок 1.7 – Конструктор тесту (опитування) Google Forms

Додаток має досить приємний дизайн, зручний інтерфейс, а також великі можливості для додавання медіафайлів (таких як фото, відео, посилання тощо) до запитань. Також є в наявності зручний інструмент систематизації результатів тестування та виведення статистики. Основним недоліком додатку є те, що ця система погано підходить для тестування студентів, а більше для опитування,

оскільки тут відсутній функціонал перевірки відповідей та визначення результату (оцінки).

Наступна за популярноітю програма – MyTestX, яка є десктопним застосунком. Вона має досить широкий функціонал, глибокі можливості по виведенню статистичних даних та їх аналізу. Є досить потужним інструментом, що є і перевагою, і недоліком водночас. Для викладачів не завжди потрібен такий великий набір інформації, глибокого статистичного аналізу тощо. Це все дуже нагромаджує інтерфейс програми (рисунок 1.8) і робить його важким для сприйняття новими користувачами.

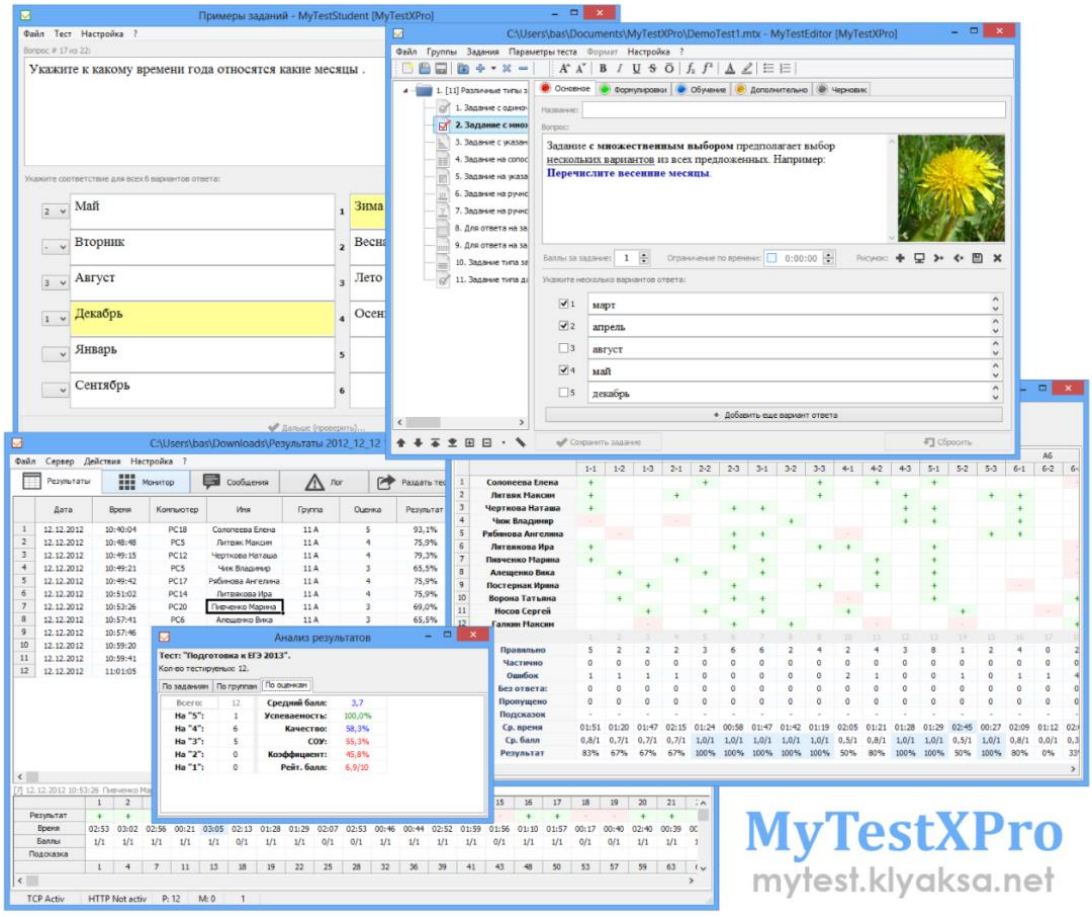


Рисунок 1.8 – Інтерфейс програми MyTestX

Та, мабуть, найбільшим недоліком цього комплексу є те, що це все ж десктопний додаток, для користування яким потрібна його інсталяція, та ще й на кожному комп’ютері, з якого буде проводитися тестування.

1.3 Визначення вимог до програмного забезпечення

Для визначення вимог до майбутнього програмного забезпечення прийнято використовувати уніфіковану мову моделювання – UML. Вона дозволяє візуально представити структуру майбутньої системи та її об’єкти.

Щоб представити функціональне призначення системи, було побудовано діаграму варіантів використання (рисунок 1.9). Вона складається з користувачів системи (акторів) та варіантами використання (дій) які вони можуть робити. У таблиці 1.1 буде описано акторів системи, у таблиці 1.2 – варіанти використання.

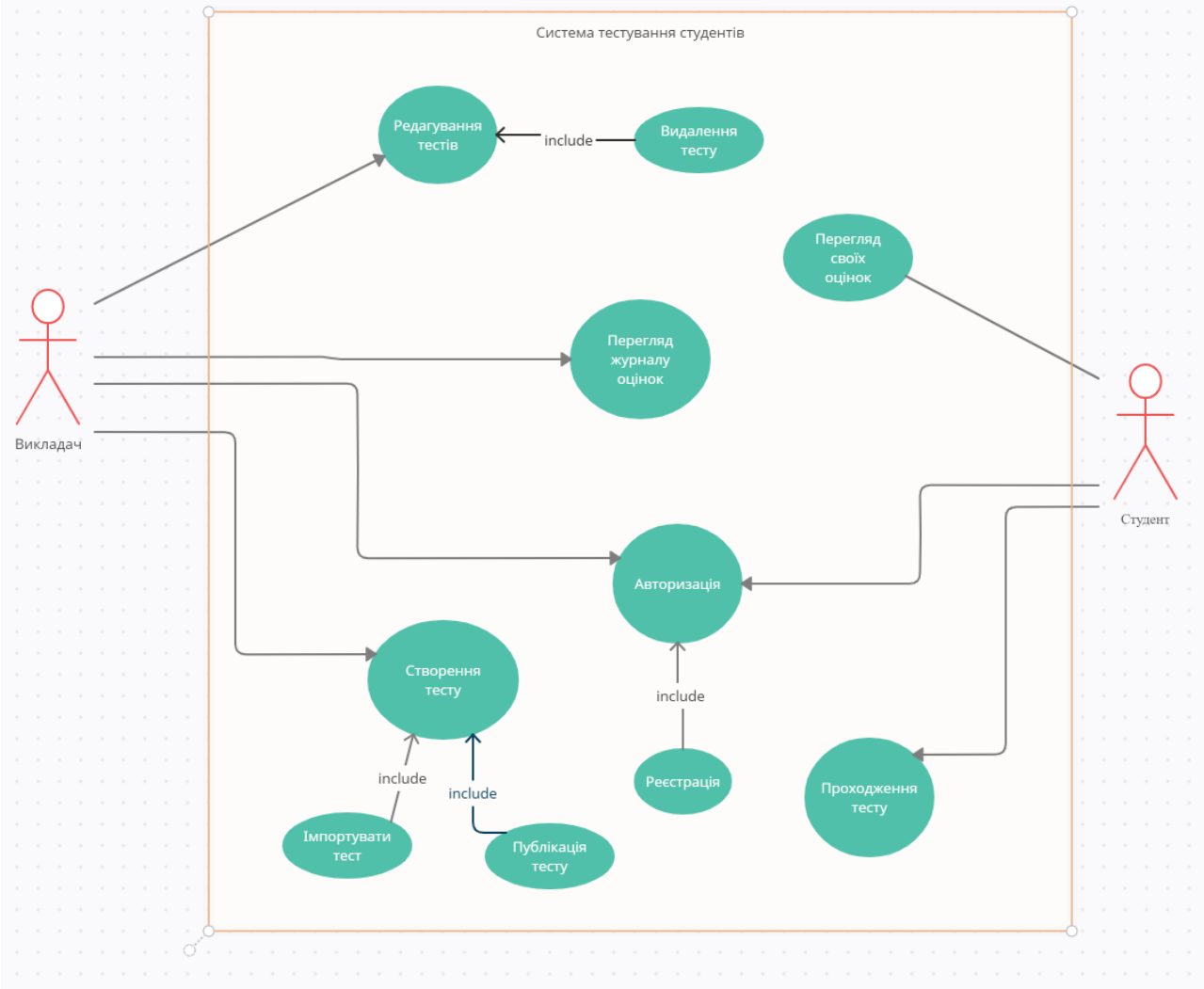


Рисунок 1.9 – Діаграма варіантів використання

Таблиця 1.1 – Опис акторів системи

Актор	Короткий опис
Викладач	Має авторизуватись. Якщо користувач ще не зареєстрований – має зареєструватися. Після чого може створювати, публікувати, редагувати та видаляти тести. Може імпортувати тест з іншого формату та в результаті отримати готовий тест без самостійного конструювання. Може переглядати журнал оцінок (результати проходження тестів)
Студент	Має авторизуватись. Якщо користувач ще не зареєстрований – має зареєструватися. Може проходити тести, які викладач опублікував. Також має можливість переглядати свої оцінки за пройдені тести.

Таблиця 1.2 – Опис варіантів використання

Актор	Найменування ВВ	Опис ВВ
Викладач/студент	Реєстрація	Користувач повинен бути зареєстрований, щоб користуватися системою
	Авторизація	Для початку роботи з системою потрібно авторизуватися
Викладач	Створення тесту	Викладач створює новий тест та може його зразу ж опублікувати для проходження студентами. Є можливість імпорту готового тесту з іншого формату.
	Редагування тесту	Викладач може вносити правки в тест або ж видаляти тест повністю
	Перегляд журналу оцінок	Викладач переглядає журнал оцінок (результати проходження тестів)
Студент	Проходження тесту	Студент може проходити опубліковані викладачем тести
	Перегляд своїх оцінок	Студент переглядає свої оцінки за пройдені тести

Технічне завдання на розробку програми подано у додатку А.

Отже, у даному розділі було проаналізовано основні проблеми оцінювання якості знань студентів, які виникають під час дистанційного навчання, а також запропоновано вирішення її – створення програмної системи для швидкого та зручного тестування.

Попри те, було проаналізовано інше наявне на ринку програмне забезпечення, виділено їх основні плюси та мінуси.

Вимоги до програмної системи були висвітлені у вигляді діаграми варіантів використання та витікали також з аналізу наявного на ринку програмного забезпечення.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та вибір архітектури веб-додатку

Оскільки створюване програмне забезпечення буде у вигляді веб-додатку, доцільним буде розглянути для його проектування таку архітектуру як клієнт-серверну. Тим паче, у веб-додатку мають бути присутні інтерактивні елементи, що також переважує терези у сторону клієнт-серверної архітектури.

Клієнт-серверна архітектура – це така модель інформаційної мережі, в якій кілька компонентів працюють у строго визначених ролях для спілкування між собою. Сервер розміщує, постачає та управляє більшістю сервісів та послуг, що споживаються клієнтом. Цей тип архітектури поширюваних сервісів має один або кілька клієнтських комп'ютерів, підключених до центрального сервера через мережу або Інтернет (рисунок 2.1). Вона набула своєї популярності завдяки динамічному розвитку мережі Інтернет та зосередженню значної частини інформації в базах даних на серверах.

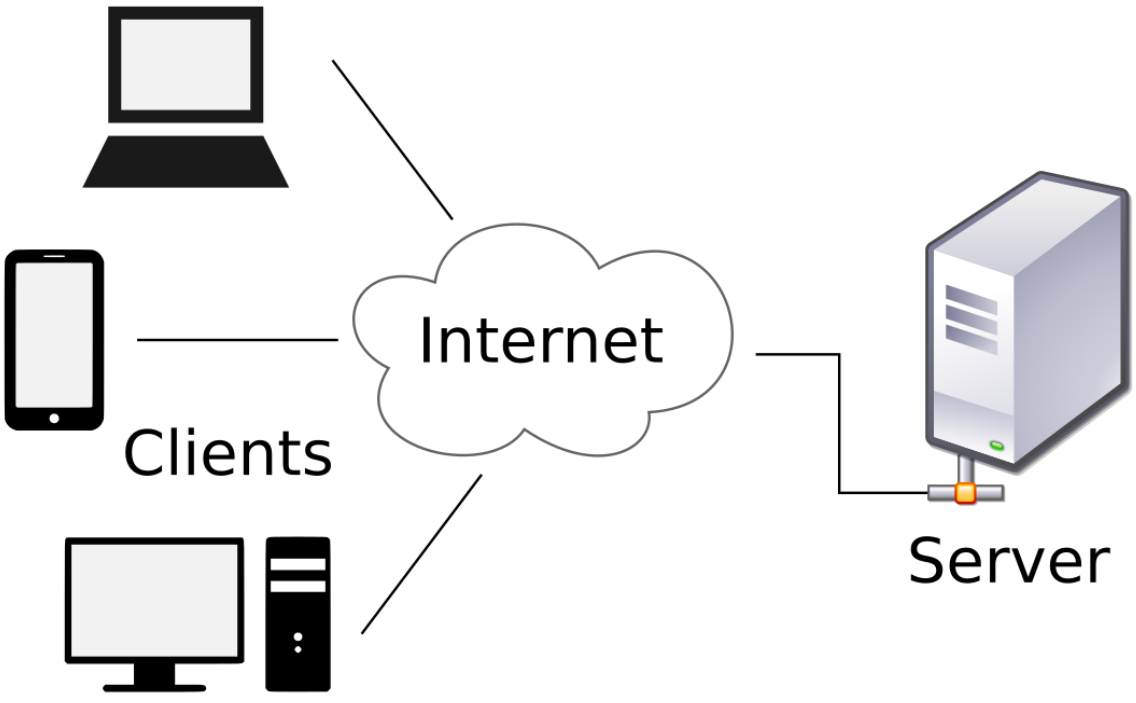


Рисунок 2.1 – Загальна схема архітектури клієнт-сервер

Клієнт-серверна архітектура також відома як модель «мережевих обчислень», оскільки всі послуги та запити виконуються за допомогою мережі. Її робота подібна до розподілених обчислювальних систем, оскільки всі вузли виконують свої власні функції. Це спільна обчислювальна мережа, де різні віддалено підключені пристрої надсилають кілька запитів до централізованої системи (сервера або хост-машини). Клієнтська частина пропонує зручний інтерфейс, який дозволяє користувачам відправляти запити на сервер і як результат отримувати відповідь у зрозумілому для нього вигляді.

Оскільки така архітектура передбачає розділення програми на дві частини – клієнтську і серверну, то постає питання розподілу функцій між ними.

У комп'ютерній термінології термін «клієнт» означає певне програмне чи апаратне забезпечення, яке взаємодіє із сервером, щоб дозволити користувачеві отримувати дані про дії, що здійснюються системою.

Клієнт є найважливішим компонентом системної архітектури. Простим прикладом клієнта є звичайний веб-браузер, який може передавати веб-запити на веб-сервер, отримуючи у відповідь вміст необхідної веб-сторінки. Усі клієнти в архітектурі клієнт-сервер умовно можна розділити на два підтипи: товсті та тонкі.

Товстий клієнт – це клієнт, який виконує операції, які вимагає користувач, незалежно від основного сервера. Основний сервер у цьому типі системної архітектури може бути використаний як спеціальна база даних, яка обробляється і доставляє дані на ПК користувача. Товстий клієнт – це таке програмне забезпечення, яке виконує всі потрібні функції по обчисленнях та обробці даних на стороні клієнта.

До переваг використання товстого клієнта можна віднести:

- краща продуктивність, порівняно із тонким клієнтом;
- наявність багатокористувацького режиму;
- можливість використання програми в автономному режимі;
- швидкий відклик програми;
- низька залежність від складних серверів.

Недоліки ж товстих клієнтів такі:

− всі пристрої, які працюють потребують постійної технічної підтримки;

− при оновленні програмної системи, це потрібно провести на кожному комп'ютері, який використовує дане ПЗ, окремо;

− такі програми зазвичай заточені під платформи, для яких вони розроблялися, з цього витікає що вони платформозалежні.

Тонкий клієнт – це такий тип клієнта, який передає всі дані для обробки на сервер, сам при цьому витрачаючи ресурси лише на відображення результатів. Всі обчислювальні ресурси такого клієнта повністю обмежені і повинні бути достатніми для запуску необхідного мережевого програмного забезпечення за допомогою веб-інтерфейсу, наприклад. Одним із найпоширеніших прикладів цього типу клієнтів є ПК з попередньо встановленим веб-браузером, який використовується для роботи з веб-додатками.

Переваги моделі тонкого клієнта:

− низька потреба в апаратному обслуговуванні та підтримці;

− низький ризик збоїв у роботі;

− низькі технічні вимоги до обладнання.

Недоліки:

− при виникненні неполадок на сервері це вплине на всіх підключених користувачів;

− відсутня можливість працювати в автономному режимі;

− якщо у такій системі починає взаємодіяти велика кількість даних, то продуктивність основного сервера може значно просідати.

Тож, основна відмінність між товстим і тонким клієнтом полягає у шляху обробки даних. Товсті клієнти працюють з інформацією використовуючи власне апаратне та програмне забезпечення, а тонкі клієнти використовують програмне забезпечення та потужності основного сервера для обробки даних. У свою чергу тонкі клієнти забезпечують систему необхідним графічним інтерфейсом, щоб дати можливість користувачеві взаємодіяти із цією системою.

Оскільки розроблюваний додаток – це система для тестування студентів, то найкращим вибором буде модель тонкого клієнта, адже вона дає змогу не прив’язуватися до платформи, а доступ робить можливим з будь якого пристрою підключеного до мережі Інтернет.

Коли створення програмного забезпечення тільки зароджувалося, розробники створювали його без жодної архітектури.. Спочатку це здавалося зручним: ніяких витрат, пов'язаних з проектуванням та пришвидшене прототипування. Проте із ускладненням ПЗ його гнучкість і керованість ставали все складнішими і складнішими, а впровадження нового функціоналу було все дорожчим і дорожчим. Це призводило до того, що розвиток проекту за межі визначених спочатку рамок було дуже складним.

За роки розвитку програмного забезпечення та підходів до його створення розробники придумали і розробили надійні підходи, щоб уникнути проблем, пов’язаних із проектуванням без архітектури.

Для створення веб-додатків найбільш популярними стали два підходи – це мікросервісна та монолітна архітектури. Зараз розглянемо їх та порівняємо.

Монолітна архітектура – це традиційний підхід до розробки програмного забезпечення, при якому вся функція системи базується на одному додатку як єдиному, автономному блоці і всі функціональні можливості проекту існують в одній кодовій базі. У такому додатку всі функції виконуються та обслуговуються в одному місці. Звичайно, додаток має свою внутрішню структуру, що складається з бази даних, клієнтського інтерфейсу, бізнес-логіки, але вона все ще залишається неподільною одиницею (рисунок 2.2).

Монолітна архітектура вважається традиційним способом побудови додатків. Зазвичай таке рішення включає в себе інтерфейс користувача на стороні клієнта, додаток на стороні сервера та базу даних. Він уніфікований, і всі функції керуються та обслуговуються в одному місці.

У тісно пов'язаній архітектурі кожен компонент та пов'язані з ним компоненти повинні бути присутніми для виконання або компіляції коду.

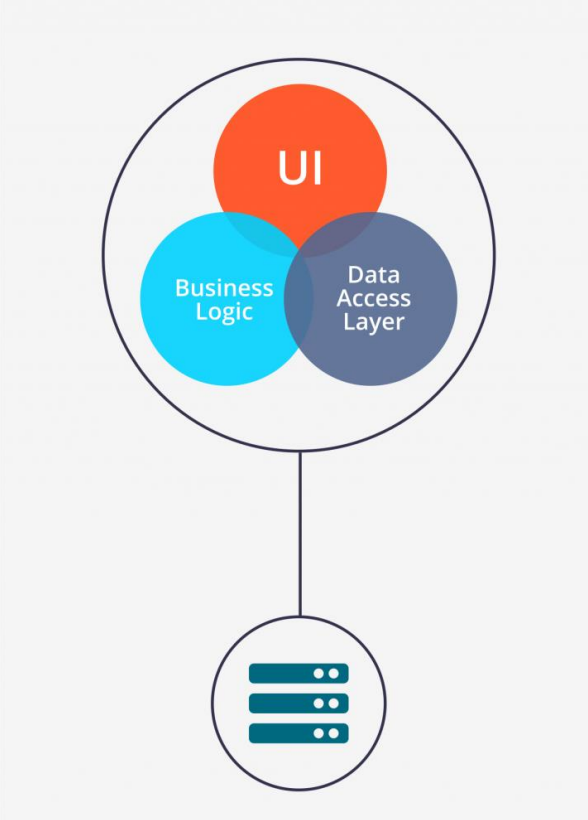


Рисунок 2.2 – Монолітна архітектура

Зазвичай монолітні додатки побудовані на багатошаровій архітектурі. Компоненти всередині багатошарової архітектури організовані у вигляді горизонтальних шарів. Кожен шар виконує певну роль у програмі (наприклад, логіка представлення або бізнес-логіка). Хоча шаблон багаторівневої архітектури не визначає кількість та типи шарів, які повинні існувати в шаблоні, більшість таких додатків складається з трьох стандартних шарів: представлення, бізнес-логіки та шар доступу до даних (рисунок 2.3).

Варто взяти до уваги, що кожен з шарів в архітектурі позначений як закритий. Це дуже важлива концепція в багатошаровій архітектурі. Закритий шар означає, що, коли запит переходить від шару до шару, він повинен пройти шар прямо над ним, щоб дістатися до наступного шару нижче цього. Наприклад, запит, що надходить із рівня представлення, повинен спочатку пройти бізнес-рівень, а потім – рівень доступу до даних, перш ніж нарешті потрапити у рівень бази даних.

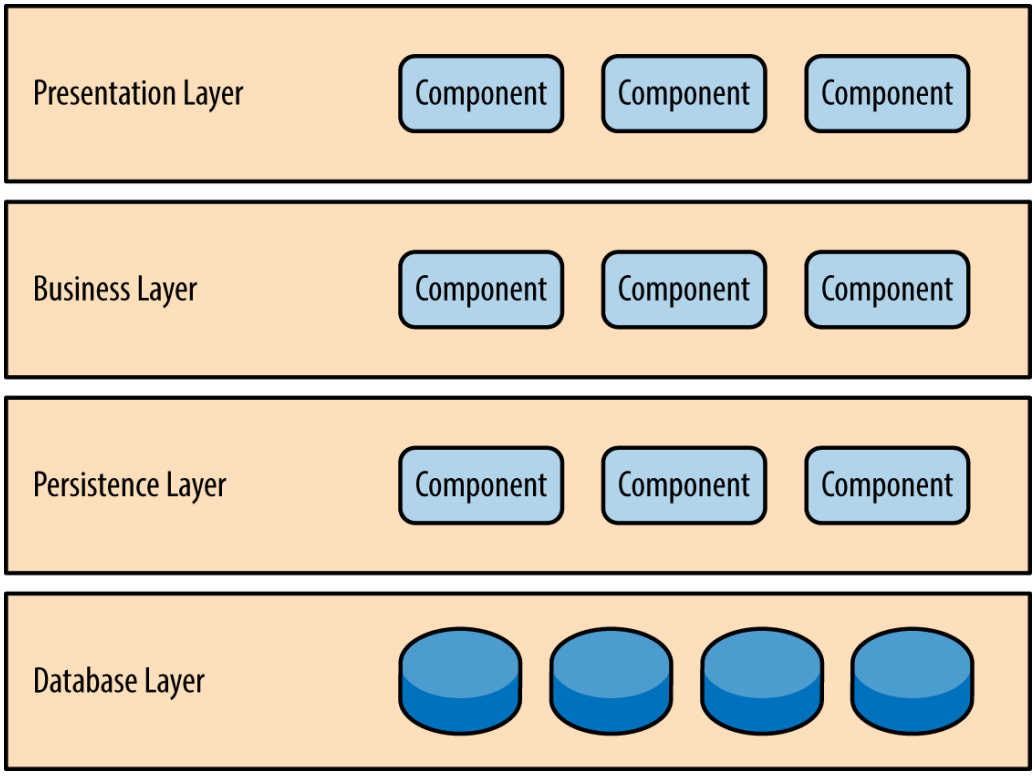


Рисунок 2.3 – Багаторівнева архітектура монолітного додатку

Кожен шар багаторівневої архітектури має свою роль і функціональну відповідальність. Наприклад, рівень представлення відповідає за обробку користувацького інтерфейсу та логіку спілкування користувача із системою. Шар бізнес-логіки відповідає за обробку даних відповідно до бізнес-завдань. Кожен шар в такій архітектурі утворює абстракцію навколо завдань, які він має вирішувати щоб виконувати свою частину функціоналу системи. Наприклад, рівень представлення не повинен знати і турбуватися про те, як отримати і обробити дані від користувача, він відповідає лише за відображення цих даних для користувача у певному форматі. Так само, рівень бізнес-логіки не повинен турбуватися про те, як форматувати дані для відображення їх клієнту чи навіть звідки ці дані взяти. Він повинен лише взяти дані з рівня доступу до даних, виконати власний функціонал по обробці даних, закладених бізнес-завданнями, та віддати результати рівню представлення для відображення їх клієнтові.

Однією з найпотужніших переваг багатошарової архітектури є розподіл проблем між компонентами. Компоненти в межах певного шару мають справу

лише з логікою і завданнями, які стосуються лише цього рівня. Наприклад, компоненти рівня представлення відповідають лише за логіку відображення даних клієнтові, тоді як компоненти, що перебувають на рівні бізнес-логіки, відповідають лише за обробку цих даних та обчислень. Такий тип класифікації компонентів дозволяє легко побудувати ефективну модель ролей та відповідальностей у архітектурі. А це у свою чергу значно спрощує розробку, тестування та підтримку програмного продукту.

Монолітна архітектура додатку має цілу низку недоліків.

З часом монолітний додаток стає занадто великим, і це тягне за собою великі проблеми з управління ним. Для внесення навіть невеликих змін у додаток потрібна буде його повна перебудова та розгортання. А зі збільшенням розміру програми також збільшується час її запуску та розгортання.

Новому розробнику дуже важко буде приєднатися до проекту і зрозуміти логіку всього монолітного додатку, навіть якщо розробник буде відповідати тільки за деяку його функціональність.

У випадку, якщо лише окрема частина додатку підпадає під велике навантаження чи трафік, щоб уникнути перевантажень прийдеться розгортати всю програму на нових серверах, адже вона повністю пов’язана. Це максимально неефективно та забирає зайві ресурси. Отже, горизонтальне масштабування неможливе в монолітних додатках.

Дуже важким є впровадження нових технологій для певної функціональності додатку, адже це вплине на весь додаток.

Також великою проблемою є те, що збій у одній частині додатку може порушити роботу усього монолітного додатку.

Проте, монолітна архітектура має і декілька переваг.

Такі додатки прості у розробці у порівнянні із мікросервісними, для яких потрібні висококваліфіковані фахівці, щоб визначити і розробити сервіси.

Монолітні додатки простіше розгорнути, оскільки розгортається лише один єдиний програмний пакет, навідміну від мікросервісного додатку, де розгортати доведеться кожен сервіс окремо.

Також у монолітних додатків значно нижча проблема мережевої затримки та безпеки у порівнянні з іншими архітектурами.

Мікросервісна архітектура – це такий стиль архітектурної розробки, в якому додаток складається з менших сервісів, що взаємодіють між собою (рисунок 2.4). Кожен такий сервіс є автономним і повинен виконувати лише одну свою бізнес-функцію. Ця архітектура використовує API для передачі інформації від одного сервісу до іншого, наприклад запитів користувачів або потоку даних .

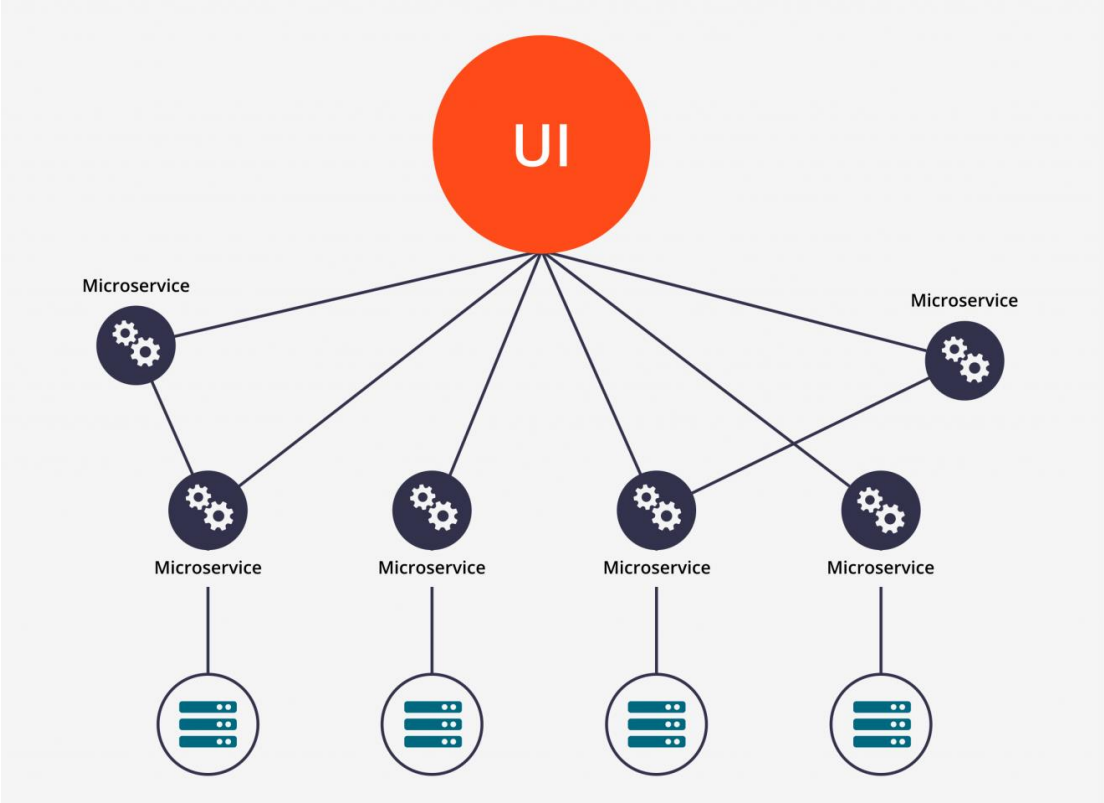


Рисунок 2.4 – Мікросервісна архітектура

Зараз розглянемо переваги використання мікросервісної архітектури для створення веб-додатків.

Мікросервісною архітектурою значно легше керувати, у порівнянні із монолітною, оскільки вона набагато менша.

Якщо нам потрібно оновити чи змінити якийсь функціонал у додатку, це зачепить лише один мікросервіс, який відповідає за нього.

Усі мікросервіси є автономними та розгортаються незалежно. Також їх розгортання займає набагато менше часу.

Новому розробнику дуже легко влитися у проект і приступити до розробки, адже йому потрібно зрозуміти принцип дії лише одного мікросервісу, який забезпечує функціонал, над яким йому прийдеться працювати, а не всю систему цілком.

У випадку, коли один мікросервіс підпадає під велике навантаження чи трафік, для вирішення цієї проблеми варто масштабувати лише цей сервіс. Отже, мікросервісна архітектура підтримує горизонтальне масштабування.

У залежності від поставлених бізнес-задач, кожен мікросервіс може використовувати ту технологію, яка найкраще підходить для їх вирішення.

Якщо певний мікросервіс виходить з ладу через помилку чи збій, це не впливатиме на працездатність інших сервісів, і системи в цілому. Тому інші мікросервіси зможуть продовжувати виконувати свої функції для користувачів.

Використання мікросервісної архітектури також має певні недоліки, які розглянемо нижче.

Так як така система є розподіленою, вона набагато складніша ніж монолітна програма. Складність її зростає зі збільшенням кількості сервісів.

Для розробки мікросервісного додатку розробники мають бути висококваліфікованими для правильного розподілу функціоналу по мікросервісам та організації їх правильної взаємодії.

Незалежне розгортання мікросервісів є досить складним процесом. Також мікросервіси менш захищені порівняно з монолітними додатками через взаємозв'язок між ними по мережі. Відладка таких додатків є досить важкою, оскільки елемент керування перетікає від мікросервісу до мікросервісу, тому визначити де і чому саме сталася помилка стає важко.

Отже, беручи до уваги переваги та недоліки кожної з архітектур, для створюваного додатку було обрано монолітну архітектуру, адже це дозволить швидко створити MVP (Minimum viable product) – мінімально життєздатний продукт, який тим не менш буде виконувати всі поставлені завдання.

2.2 Опис структури даних та моделі бази даних

Модель даної предметної області можна представити таким неформальним текстом**:**

- користувач реєструється в системі;
- є два види користувачів – студент та викладач;
- викладач створює тести, редагує їх та видаляє;
- студент проходить тести;
- користувачі мають можливість перегляду оцінок результатів по пройденим тестам.

В ході додаткового аналізу даних, які потрібно врахувати, було виявлено що:

- для кожного користувача потрібно зберігати його прізвище ім’я, логін для авторизації, пароль;
- тест має лише один варіант правильної відповіді;
- кожен тест має свій власний ідентифікатор;
- тест може бути у одній з трьох систем оцінювання;

Проаналізувавши цю інформацію, можемо виділити такі сутності та атрибути в них:

- сутність Користувач з атрибутами: унікальний номер; ім’я користувача; прізвище; логін; пароль; роль користувача;
- сутність Тест з атрибутами: унікальний номер; назва тесту; кількість спроб; ідентифікатор; номер викладача, який створив тест; система оцінювання; дата створення тесту;
- сутність Питання з атрибутами: унікальний номер; текст питання; номер тесту, до якого належить питання; номер правильної відповіді;
- сутність відповідь з атрибутами: унікальний номер; номер питання, до якого належить відповідь; текст відповіді;
- сутність Пройдений Тест з атрибутами: унікальний номер, номер тесту, номер студента, який пройшов тест, оцінка, спроба, дата здачі тесту.

Відповідно до описаної моделі сутність-зв'язок було спроектовано базу даних, схема якої подано на рисунку 2.5.

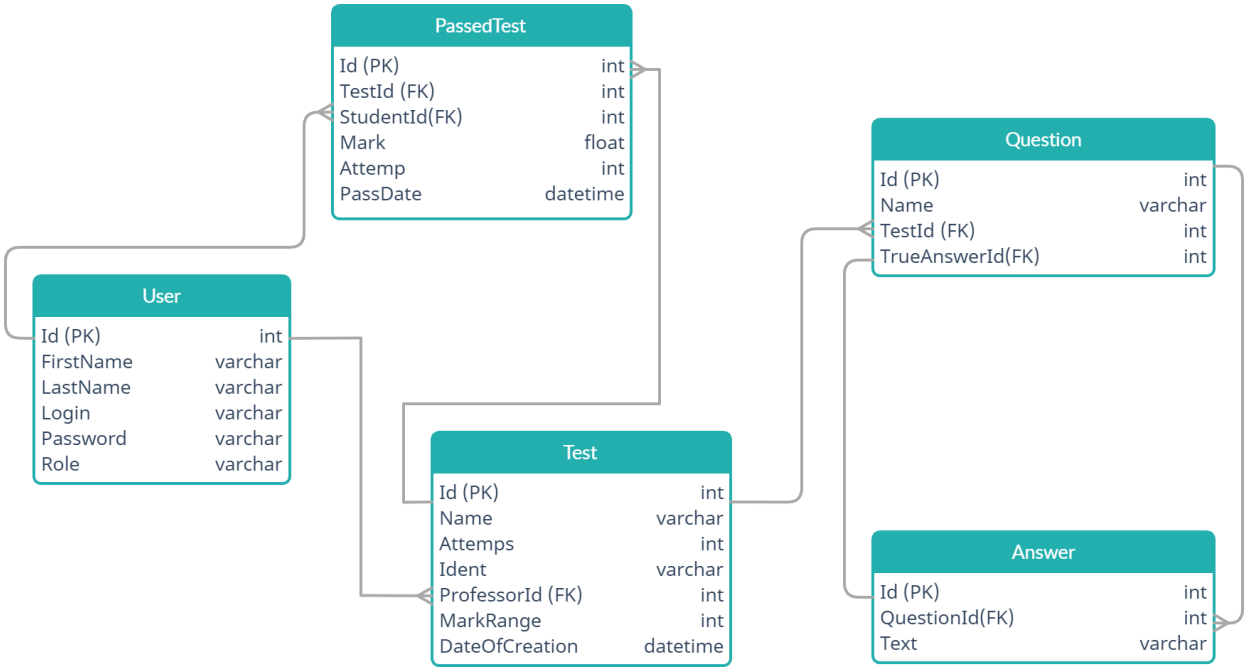


Рисунок 2.5 – Схема спроектованої бази даних

2.3 Проектування серверної частини веб-додатка

Практично стандартом для проектування монолітних додатків є архітектурний шаблон MVC (Model-View-Controller).

Модель-Вид-Контролер (MVC) – це архітектурний шаблон, який розділяє додаток на три основні логічні компоненти: модель, представлення та контролер (рисунок 2.6). Кожен із цих компонентів створений для обробки певної частини додатку. MVC є найбільш часто використовуваним для створення масштабованих та розширюваних проектів. Він відокремлює бізнес-логіку додатку від рівня представлення даних користувачеві. Згідно цієї моделі, кожен елемент повинен існувати в додатку, але не бути тісно пов'язаним з іншим, або ж бути малопов'язаним. Ідея, яка лежить в основі цього шаблону – потрібно чітко розділяти відповідальність за різний функціонал у додатку.

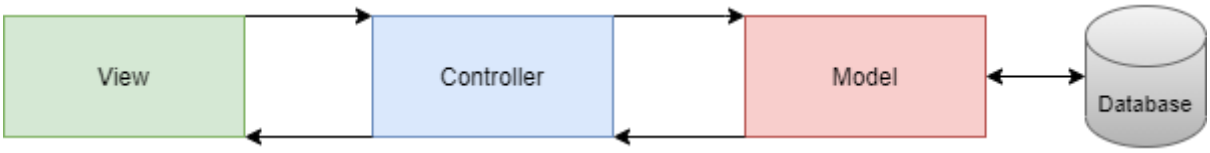


Рисунок 2.6 – Схема архітектурного шаблону MVC

Компонент Модель відповідальний за обробку всіх даних, з якими працює користувач системи. Це включає в себе як дані, які перетікають між Контролером на Представленням, так і будь які дані з бізнес-логіки додатку.

Компонент Представлення використовується для керування логікою інтерфейсу користувача. Представлення генерується відповідно до даних, які переходять до нього із Моделі.

Компонент Контролер виступає своєрідним інтерфейсом між компонентами Представлення та Моделі. Він призначений для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою компонента Моделі та взаємодії з Представленням для виведення кінцевого результату користувачеві.

Відповідно до цього шаблону та бізнес-задач, які має виконувати додаток було спроектовано чотири контролери.

Контролер AccountController, який буде відповідати за обробку запитів по реєстрації користувачів, видачі токену, авторизації.

Контролер TestController, який відповідатиме за перевірку зданого тесту, видачу результатів по пройденим тестам, імпорту та експорту вже готових тестів з JSON та XML форматів, присвоєння тесту індентифікатору

Контролер ProfessorController, який буде відповідати за створення тестів, їх редагування та видалення.

Контролер StudentController, який відповідатиме за видачу результатів проходження тесту, завантаження тесту по ідентифікатору чи посиланню.

Модель даних буде ідентична тій, що описана у розділі 2.2, оскільки у додатку буде використовуватися така технологія як ORM, яка дозволить зв’язати базу даних з об’єктно-орієнтованими моделями у додатку.

2.4 Проектування клієнтської частини веб-додатка

Правильне і грамотне проектування користувацького інтерфейсу є дуже важливою частиною створення будь якого додатку. Адже інтерфейс – це перше, що зустрічає користувача, і це те, з чим йому доведеться взаємодіяти постійно. Тому правильна його побудова є невід'ємною частиною для створення приємного враження про програмний продукт.

Оскільки уже є побудована діаграма варіантів використання, отже можна приступити до створення інтерфейсу користувача відповідно до неї.

Першим що зустрічає користувача – це сторінка авторизації та реєстрації (рисунок 2.7). На ній міститься форма із двома вкладками. Перша вкладка для авторизації, друга – для реєстрації.

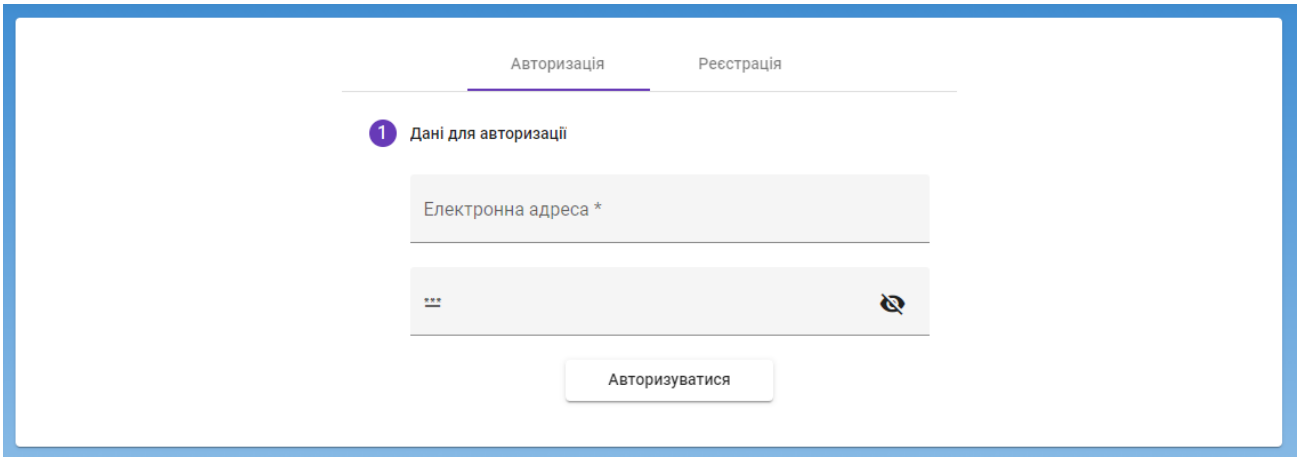


Рисунок 2.7 – Форма авторизації та реєстрації

У вкладці для авторизації присутні два поля – введення електронної адреси та паролю. У полі введення паролю присутня кнопка показати/приховати пароль для інформаційної безпеки.

Вкладка для реєстрації складається з трьох блоків (рисунок 2.8). Перший блок – персональні дані. У ньому користувач вводить своє прізвище та ім'я. Другий блок – для вибору ролі: викладач чи студентю У третьому блоці користувач вводить дані для авторизації – електронну пошту та пароль.

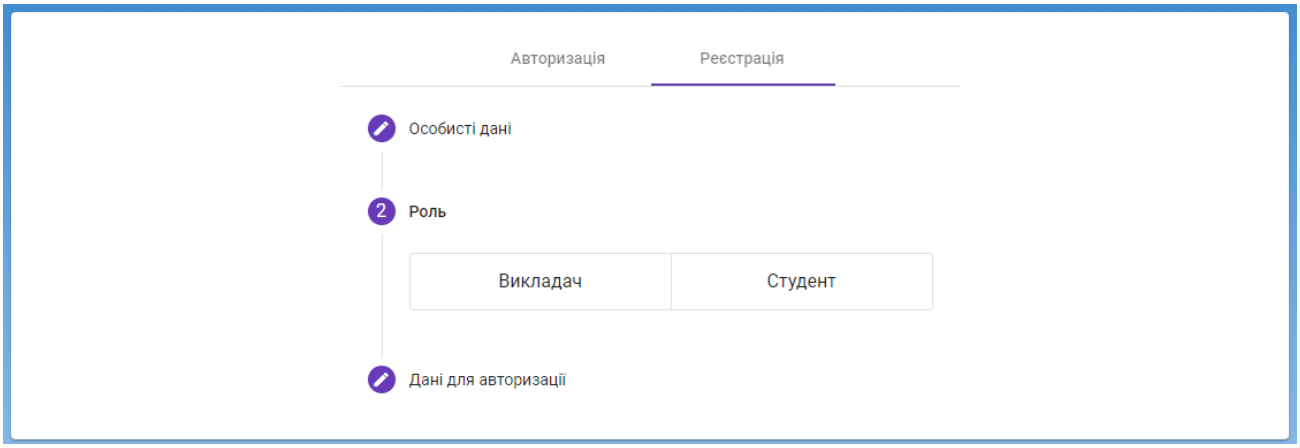


Рисунок 2.8 – Вкладка форми для реєстрації нового користувача

Далі, в залежності від обраної ролі, інтерфейс буде частково відрізнятися у викладача та студента. Почергово розглянемо кожен варіант.

Якщо авторизований користувач з роллю викладача, у верхній панелі йому будуть доступні додаткові функціональні кнопки (рисунок 2.9). Та що зліва – кнопка для створення нового тесту, та що справа – для перегляду результатів проходження створених тестів. У користувача з роллю студента ці кнопки будуть відсутні, в той час як решта інтерфейсу буде ідентична.

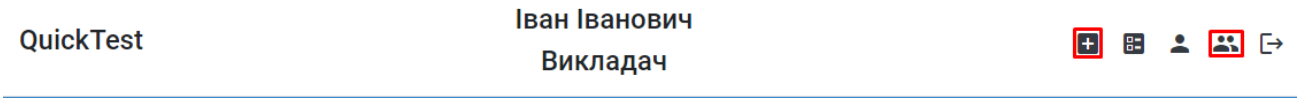


Рисунок 2.9 – Функціональна панель

Як бачимо з рисунку 2.9, посередині панелі буде виводитися прізвище та ім’я користувача, а нижче – його роль: викладач або студент. Зліва на панелі виводиться назва програми.

При натиску на першу функціональну кнопку, викладачеві буде видана форма для створення тесту (рисунок 2.10). На ній будуть поля для введення назви тесту, кількості спроб для проходження тесту, вибір системи оцінювання (5-ти, 12-ти чи 100 бальна система) та кнопка для імпорту вже готового тесту. Якщо користувач натисне на неї, йому буде відкрите вікно провідника для вибору файлу, з якого потрібно завантажити готовий тест.

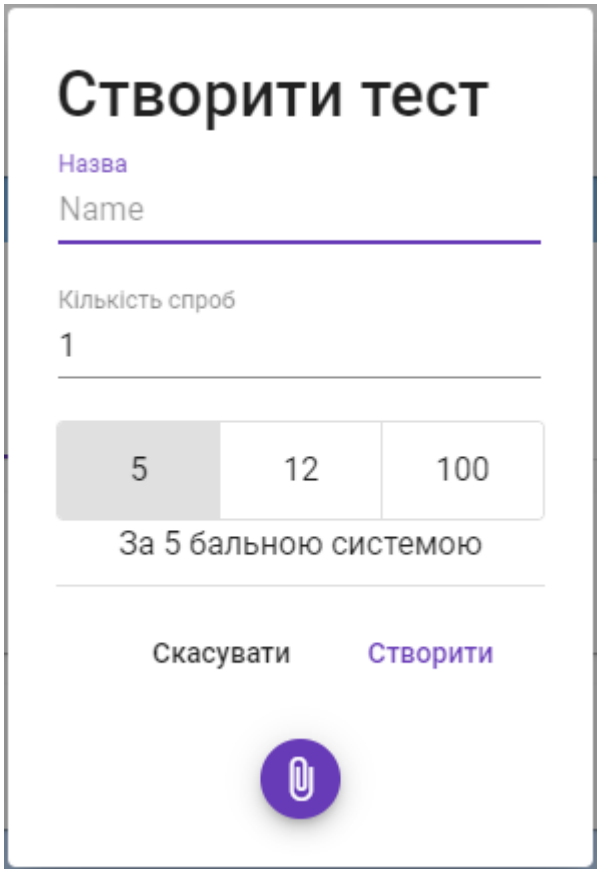


Рисунок 2.10 – Форма для створення нового тесту

Викладач може переглядати створені ним тести. Вони будуть розміщені у вигляді форм (рисунок 2.11). Кожна така форма міститиме дві вкладки. Перша вкладка – інформація про тест, де вказано його назву, посилання для поширення тесту, унікальний ідентифікатор тесту. Також там розміщені кнопки для імпорту тесту у формати JSON та XML.

Друга вкладка призначена для редагування тесту (рисунок 2.12). У ній викладач може додавати нові питання, спочатку ввівши у відповідне поле назву питання. До кожного питання можна додавати певну кількість відповідей за допомогою відповідної кнопки у вигляді «+1». Вже додані варіанти відповіді можна редагувати та зберігати, натискаючи на кнопку з іконкою «дискета». Також додані варіанти відповіді можна видаляти. За це відповідатиме кнопка із знаком «-». Кожне питання також можна видаляти натиснувши кнопку «Видалити запитання». Після внесеня змін у тест, потрібно буде натиснути кнопку «Зберегти зміни» для збереження змін.

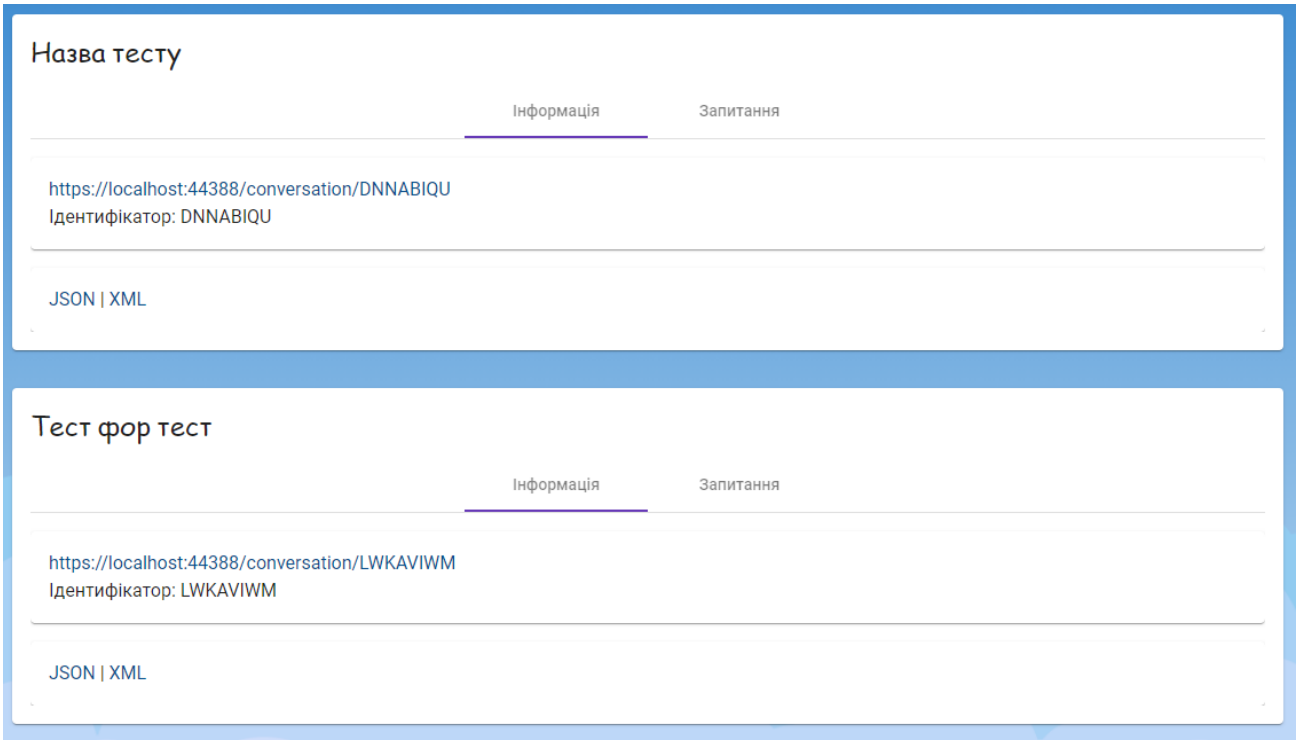


Рисунок 2.11 – Створені викладачем тести

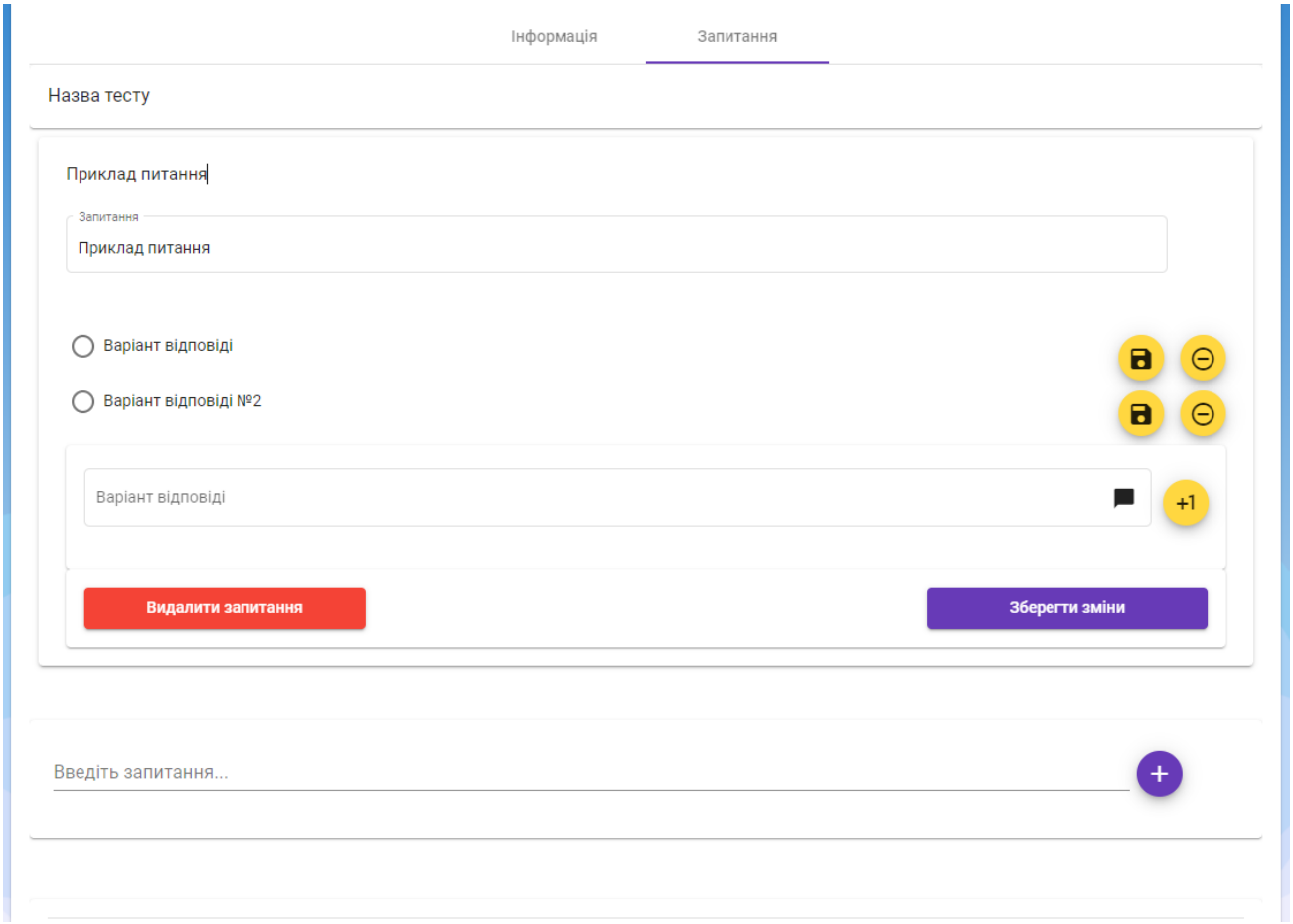


Рисунок 2.12 – Вкладка форми для редагування тесту

Форма для проходження тесту буде представлена на рисунку 2.13. На ній буде виведено назву тесту, автора тесту, систему оцінювання та поточну спробу. Вибрана відповідь буде позначена заповненим кружком.

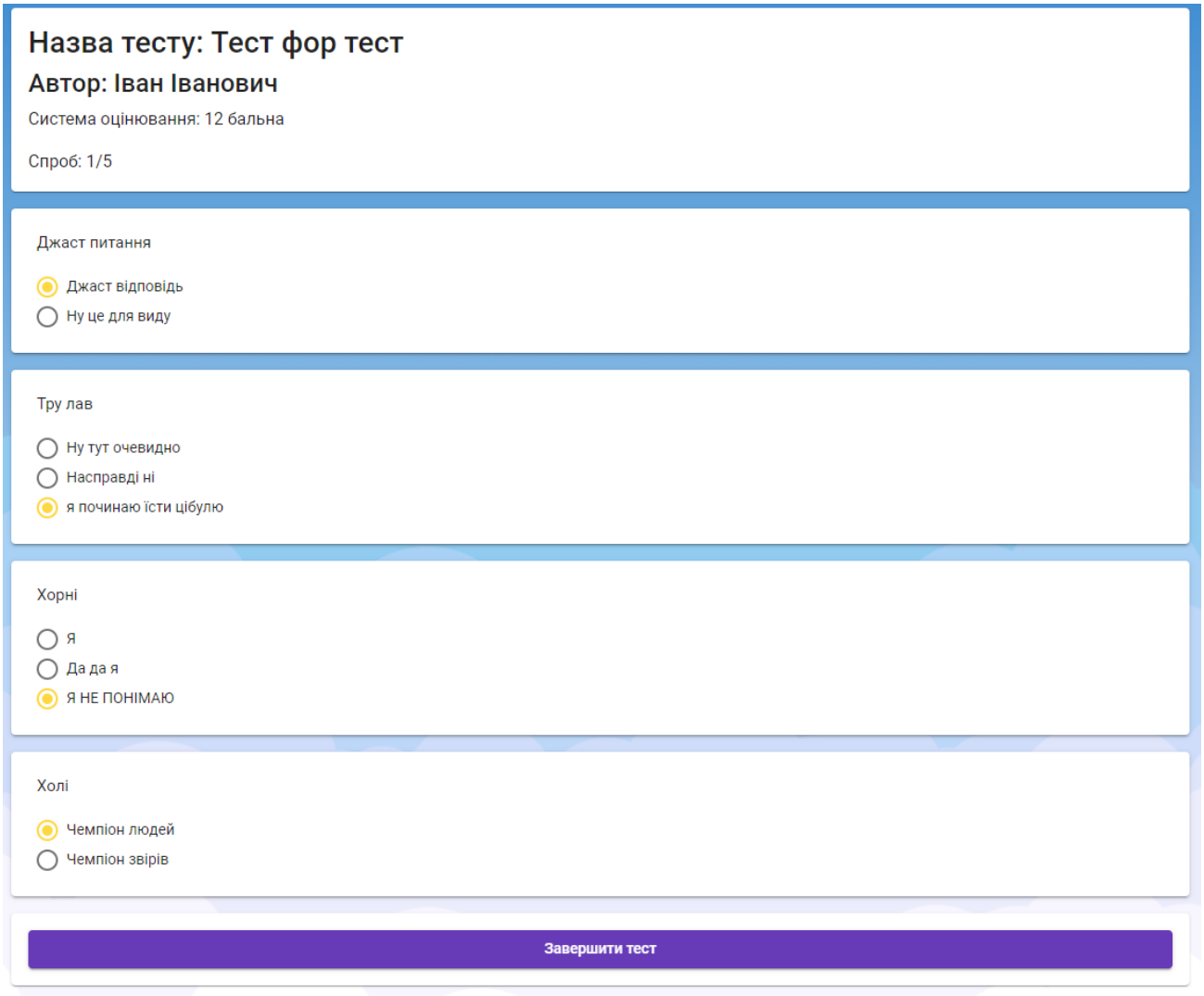


Рисунок 2.13 – Форма проходження тесту

2.6 Аналіз та вибір технологій і методів реалізації веб-додатка

На сьогоднішній момент мова програмування C# є однією з найпотужніших, та водночас відносною просто. Вона дуже швидко і стрімко розвивається, має величезне комп'юніті та постійну підтримку. Це об'єктно-орієнтована мова зі строгою статичною типізацією Проте говорячи про неї, варто згадати і платформу, на якій вона базується – Microsoft .NET.

Microsoft .NET – це так звана програмна платформа. Вона складається з двох частин. Основою частиною є виконуюче середовище Common Language Runtime (CLR), яке може виконувати як звичайні програми, так і серверні додатки. Друга, проте не менш важлива частина, це бібліотека класів Framework Class Library (FCL), що містить в собі безліч компонентів для роботи з базами даних, мережею, введенням та виведенням, файлами, інтерфейсом користувача тощо. Це дозволяє розробнику не займатися низькорівневим програмуванням, а використовувати вже готові класи.

Власне .NET складається з багатьох різних компонентів, призначених для вирішення різних задач. Проте нас цікавить лише ASP.NET – частина технології .NET, яка використовується для написання потужних клієнт-серверних інтернет додатків. Вона дозволяє створювати динамічні сторінки HTML. Вона містить безліч готових елементів управління, використовуючи які можна швидко створювати інтерактивні веб-додатки.

Деякі особливості ASP.NET:

− скомпільований код виконується швидше, а більшість помилок вдається відловити ще на стадії розробки;

− постійно розширюваний набір елементів управління і бібліотек класів, що дозволяє прискорити розробку;

− присутній функціонал для кешування всієї сторінки, її частин або даних, що використовуються на сторінці;

− розширювані моделі подій, обробки запитів і серверних елементів управління;

− Можливе створення веб-додатків, які реалізують шаблон Model-View-Controller (ASP.NET MVC Framework).

Для реалізації взаємодії із базою даних було обрано використовувати фреймворк Entity Framework. Це інструмент, який спрощує зіставлення об'єктів в програмному забезпеченні з таблицями і стовпцями реляційної бази даних. Entity Framework (EF) - це ORM-фреймворк з відкритим вихідним кодом для ASP.NET, який є частиною .NET Framework. ORM обробляє створення з'єднань

з базою даних і виконання команд, а також результати запитів і автоматичне надання цих результатів в якості об'єктів програми. ORM також допомагає відслідковувати зміни об'єктів додатку і може зберігати ці зміни в базі даних.

Для реалізації клієнтської частини веб-додатку було обрано фреймворк Angular з використанням бібліотеки Material Angular.

Angular - це платформа для розробки для створення ефективних та складних односторінкових додатків. Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому.

Angular надає великий вибір сторонніх інтеграцій, які можна легко додати до фреймворку. Angular пропонує швидший час завантаження та підвищений рівень безпеки, використовуючи чудову концепцію, відому під назвою «попередній компілятор». Angular компілює HTML та TypeScript у JavaScript під час розробки, а це означає, що весь код компілюється до того, як браузер завантажує веб-додаток.

Angular використовує звичайний DOM. Умовно, десять оновлень будуть зроблені на одній і тій же HTML-сторінці. Замість того, щоб завантажувати нову сторінку, Angular оновить всю деревоподібну структуру тегів HTML на наявній сторінці, що дозволяє оновлювати вміст сторінки без її перевантаження.

Angular Material - це бібліотека компонентів інтерфейсу для розробників Angular. Компоненти Angular Material допомагають створювати привабливі, послідовні та функціональні веб-сторінки та веб-програми, дотримуючись сучасних принципів веб-дизайну.

Отже, у даному розділі було проаналізовано та порівняно основні архітектурні рішення для побудови веб-додатку, та вибрано Також було проаналізовано найпопулярніші архітектури для побудови додатків, і визначено що найкращим рішенням буде монолітна архітектура додатку із використанням патерну проектування MVC. Насамкінець, було проведено проектування клієнтської та серверної частини веб-додатку та описані засоби його реалізації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка бази даних

Для реалізації бази даних у програмі було використано Entity Framework – технологію на базі .NET для роботи з даними. Він надає декілька можливостей взаємодії з базою даних. Серед них було вибрано спосіб code first – це коли розробник створює клас моделі даних, які будуть зберігатися в БД, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

Відповідно до спроектованої у розділі 2.2 бази даних було написано 5 класів моделі даних, які репрезентують спроектовані сутності. Нижче буде наведено код класів моделі.

Клас сутності Тест:

```
public class Test
{
    public int Id {get; set;}
    public string Name { get; set; }
    public int? HowManyAttempt { get; set; }
    public string Identyficator { get; set; }
    public int ProfessorId { get; set; }
    public int MarkRange { get; set; }
    public string DateOfCreation { get; set; }
}
```

Клас сутності User:

```
public class User
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
    public string Role { get; set; }
}
```

Клас сутності PassedTest:

```
public class PassedTest
{
    public int Id { get; set; }
    public int TestId { get; set; }
    public int StudentId { get; set; }
    public float Mark { get; set; }
    public int? Attempt { get; set; }
    public string PassDate { get; set; }
}
```

Клас сутності Question:

```
public class Question
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int TestId { get; set; }
    public int TrueAnswerId { get; set; }
}
```

Клас сутності Answer:

```
public class Answer
{
    public int Id { get; set; }
    public int QuestionId { get; set; }
    public string Text { get; set; }
}
```

Відповідно до написаних класів моделі Entity Framework створив базу даних із відповідними сутностями та атрибутами. Для підключення до цієї бази даних було створено клас DBContext, який наслідує пакетний клас DbContext і надає контекст бази даних для доступу до неї, виконання запитів тощо. Рядок підключення до бази даних:

```
optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Databas
e=QuickTest;Trusted_Connection=True;");
```

Таким чином було дуже просто створено базу даних за заданою моделлю та надано простий інтерфейс для взаємодії з нею.

3.2 Розробка програмних модулів

У якості архітектури додатку було обрано патерн Model-View-Controller, а засіб реалізації – ASP.NET. Ця технологія включає в себе фреймворк ASP.NET MVC – готовий шаблон для створення веб-застосунку за патерном Model-View-Controller. Він допомагає створювати проект за вже готовим шаблоном. Для створення такого проекту у середовищі Visual Studio 2019 було обрано відповідний шаблон проекту (рисунок 3.1). У цьому проекті буде уже створена структура відповідно патерну MVC. Створено окремі директорії для кожного компоненту шаблону – контролерів, моделей та представлень. Буде згенеровано стандартний HomeController, представлення у вигляді сторінки привітання ASP.NET. Також до проекту за замовчуванням додається файл Web.config, який служить для задання конфігурацій додатку. У директорію Content за замовчуванням завантажуються елементи бібліотеки Bootstrap з каскадними таблицями та готовими стилями. Оскільки для клієнтської частини ми обрали фреймворк Angular, дана директорія буде очищена.

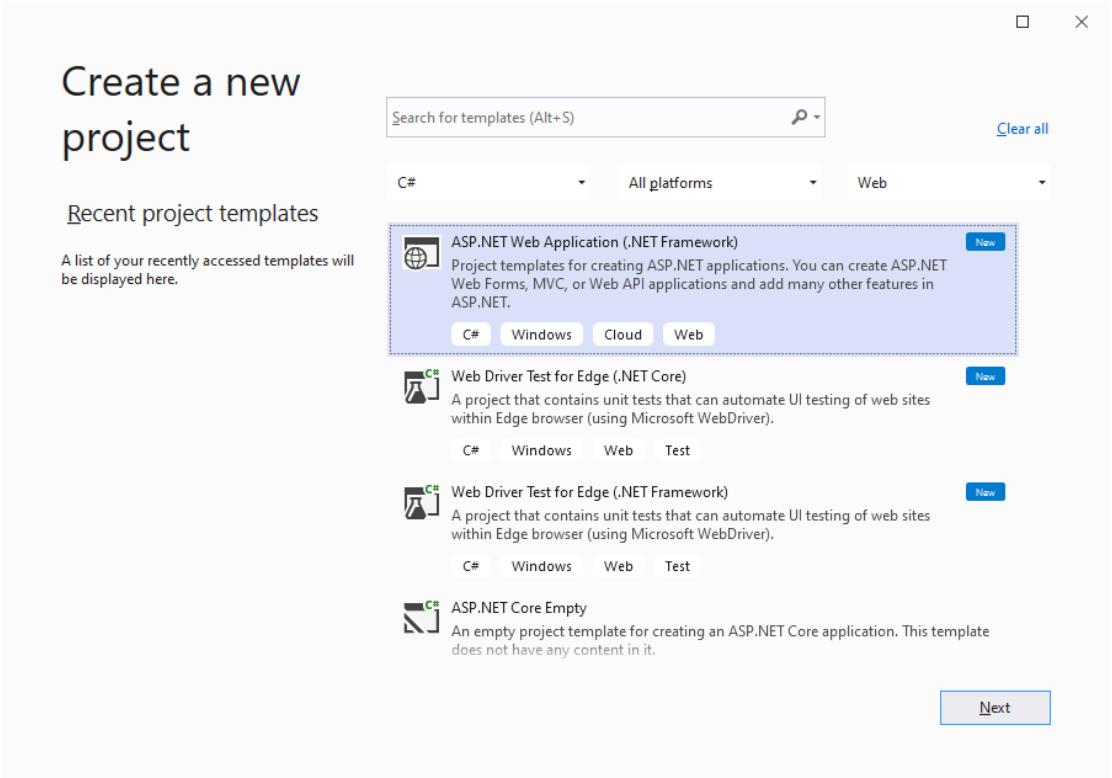


Рисунок 3.1 – Створення проекту за шаблоном ASP.NET MVC Framework

Далі перейдемо до розробки контролерів додатку. Як було спроектовано раніше, їх буде чотири: AccountController, TestController, ProfessorController, StudentController. Почергово опишемо основний функціонал кожного з них.

AccountController відповідає за обробку всіх операцій пов’язаних з акаунтами користувачів.

За реєстрацію нового користувача у ньому відповідає метод Register. На вхід він приймає всі дані користувача. Перед реєстрацією він перевіряє, чи не є вже користувач з такою електронною поштою зареєстрований. Якщо такий користувач вже є – він повертає клієнту результат у вигляді 400 коду помилки HTTP з відповідною приміткою. Якщо ж такого користувача ще немає, він створює його об’єкт і записує у базу даних. В такому випадку на клієнт повертається відповідь із статусом 200. Код методу Register:

```
[HttpGet("Register")]
public IActionResult Register(string login, string password, string
firstName, string lastName, string role)
{
    var persons = _db.People.FirstOrDefault(x => x.Login == login);
    if (persons != null) return BadRequest(new { errorText = "Email
already registered" });
    var user = new People
    {
        Login = login,
        Password = password,
        FirstName = firstName,
        LastName = lastName,
        Role = role
    };
    _db.People.Add(user);
    _db.SaveChanges();
    return Ok();
}
```

Для авторизації і аутентифікацій користувача було прийнято рішення використовувати JWT-токен. JWT (або JSON Web Token) являє собою веб-стандарт, який визначає спосіб передачі даних про користувача в форматі JSON у зашифрованому вигляді. JWT-токен складається з трьох частин: Header –

об'єкт JSON, який містить інформацію про тип токена і алгоритм його шифрування; Payload – об'єкт JSON, який містить дані, потрібні для авторизації користувача; Signature – рядок, який створюється за допомогою секретного коду, а також Header і Payload. Цей рядок служить для верифікації токена.

Додаток використовує JWT для перевірки аутентифікації користувача в такий спосіб, як описано нижче.

Спершу користувач заходить на сервер аутентифікації за допомогою аутентифікаційного ключа (у нашому випадку це пара логін/пароль). Потім сервер аутентифікації створює JWT і відправляє його користувачеві. Коли користувач робить запит до API додатка, він додає до нього отриманий раніше JWT. Коли користувач робить API запит, додаток може перевірити за поданим із запитом JWT чи є користувач тим, за кого себе видає. У цій схемі сервер додатку налаштований так, що зможе перевірити, чи є вхідний JWT саме тим, що був створений сервером аутентифікації.

Код методів для видачі токену:

```
[HttpPost("token")]
public IActionResult Token(string username, string password)
{
    People person = _db.People.FirstOrDefault(x => x.Login == 
username && x.Password == password);
    var identity = GetIdentity(username, password, person);
    if (identity == null)
    {
        ModelState.AddModelError("errorMessage", "Invalid username 
or password.");
        return BadRequest(ModelState);
    }
    var now = DateTime.UtcNow;
    // створюємо JWT-токен
    var jwt = new JwtSecurityToken(
            issuer: AuthOptions.ISSUER,
            audience: AuthOptions.AUDIENCE,
            notBefore: now,
            claims: identity.Claims,
            expires: 
now.Add(TimeSpan.FromMinutes(AuthOptions.LIFETIME)),
            signingCredentials: new 
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(), 
SecurityAlgorithms.HmacSha256));
```

```
    var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);
    var response = new
    {
        access_token = encodedJwt,
        email = identity.Name,
        username = $"{person.FirstName} {person.LastName}",
        role = person.Role
    };
    return Json(response);
}
```

Код методу для авторизації та аутентифікації:

```
private ClaimsIdentity GetIdentity(string username, string
password, People people)
{
    if (people != null)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimsIdentity.DefaultNameClaimType,
people.Login),
            new Claim(ClaimsIdentity.DefaultRoleClaimType,
people.Role),
        };
        ClaimsIdentity claimsIdentity =
        new ClaimsIdentity(claims, "Token",
ClaimsIdentity.DefaultNameClaimType,
            ClaimsIdentity.DefaultRoleClaimType);
        return claimsIdentity;
    }
    // якщо користувача не знайдено
    return null;
}
```

Конфігурація для видачі токену задається у файлі AuthOptions.cs, нижче буде приведено його лістинг:

```
public class AuthOptions
{
    public const string ISSUER = "QuickTest"; // видавець токену
    public const string AUDIENCE = "QuickTest"; // споживач токену
    const string KEY = "QuickTestSecretKey";   // ключ для
шифрування
    public const int LIFETIME = 525948; // час життя токену
    public static SymmetricSecurityKey GetSymmetricSecurityKey()
    {
```

```
        return new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));
    }
}
```

У контролері ProfessorController визначені та реалізовані методи для створення та редагування тестів. Нижче буде приведено один з них – метод для створення тесту.

```
[Authorize(Roles = "professor")]
[HttpPost("CreateConversation")]
public async Task<IActionResult> CreateConversation(string name,
int? howManyAttempt, int markRange)
{
    var rndGen = new RandomGen();
    Conversations conversations;
    var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
    string identyficator;
    do
    {
        identyficator = rndGen.RandomString(8);
        conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
    } while (conversations != null);

    var conversation = new Conversations
    {
        Name = name,
        ProfessorId = professor.Id,
        ProfessorName = $"{professor.FirstName}
{professor.LastName}",
        HowManyAttempt = howManyAttempt,
        MarkRange = markRange,
        DateOfCreation = DateTime.Now.ToString("MM/dd/yyyy HH:mm"),
        Identyficator = identyficator
    };
    _db.Conversations.Add(conversation);
    _db.SaveChanges();
    var conversationFromDb = _db.Conversations.FirstOrDefault(x =>
x.Name == name && x.HowManyAttempt == howManyAttempt && x.MarkRange
== markRange);
    await _signalRClient.connection.SendAsync("Conversations",
"add", conversationFromDb);
    return Ok();
}
```

Повний лістинг класу ProfessorController буде представлено у додатку. Ще наведемо лістинг методу для отримання всіх тестів створених викладачем.

```
[Authorize(Roles = "professor")]
[HttpPost("GetConversationByProfessorId")]
public List<Conversations> GetConversationsByProfessorId()
{
    var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
    if (professor == null) return null;
    var conversations = _db.Conversations.Where(x => x.ProfessorId
== professor.Id).ToList();
    if (conversations == null)
    {
        return null;
    }
    List<Question> questions;
    foreach (var conversation in conversations)
    {
        questions = _db.Question.Where(x => x.ConversationId ==
conversation.Id).ToList();
        foreach (var question in questions)
        {
            var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
            question.AnswerOptions = answerOptions;
        }
        conversation.Question = questions;
    }
    return conversations;
}
```

Далі опишемо основний функціонал контролера StudentController. Основні його функції – це обробка запиту на отримання тесту через ідентифікатор та перегляд своїх результатів проходження тестів.

Метод GetConversationByIdentyficator на вхід отримує ідентифікатор тесту. Після цього перевіряє у базі даних наявність тесту з таким ідентифікатором. Якщо такий тест знайдено, відбувається перевірка на кількість дозволених спроб. Якщо користувач вже використав всі спроби проходження тесту, йому повертається відповідь з відповідним повідомленням. Якщо ж у користувача є ще доступні спроби, йому повертається тест на проходження.

```
[Authorize(Roles = "professor,student")]
```

```
[HttpPost("GetConversationByIdentyficator")]
public Conversations? GetConversationByIdentyficator(string
identyficator)
{
    var student = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
    var conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
    if (conversations == null)
    {
        return null;
    }
    var passedConversationss = _db.PassedConversations.Where(x =>
x.ConversationId == conversations.Id && x.StudentId ==
student.Id).ToList();
    if (passedConversationss != null)
    {
        foreach (var passedConversation in passedConversationss)
        {
            if (passedConversation.Attempt >=
conversations.HowManyAttempt)
            {
                return null;
            }
        }
    }
    var questions = _db.Question.Where(x => x.ConversationId ==
conversations.Id).ToList();
    foreach (var question in questions)
    {
        var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
        question.AnswerOptions = answerOptions;
    }
    conversations.Question = questions;
    conversations.PassedConversations =
_db.PassedConversations.Where(x => x.ConversationId ==
conversations.Id && x.StudentId == student.Id).ToList();
    return conversations;
}
```

Другий важливий метод із класу StudentController це PassConversation, який отримує результати проходження тесту студента із представлення (вибрані відповіді студентом), перевіряє та обчислює оцінку, після чого проводить запис результатів у базу даних.

```
[Authorize]
[HttpPost("PassTest")]
```

```
public int? PassConversation(PassedTest passedTest)
{
    var student = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
    var test = _db.Test.FirstOrDefault(x => x.Id == passedTest.Id);
    var attempt = _db.TestResult.Where(x => x.TestId == testId &&
x.StudentId == student.Id)?.Count() ?? 0;
    var trueAwards = 0;
    for (int i = 0; i < passedTest.AnswersId.Length; i++)
    {
        if(passedTest.AnswersId[i] ==
test.Question[i].TrueAnswerId)
            trueAwards++;
    }
    var mark = ((trueAwards * Test.MarkRange) / length);
    if(markRange != 5) mark.toFixed(2);
    var newTestResult = new TestResult
    {
        StudentId = student.Id,
        StudentName = $"{student.FirstName} {student.LastName}",
        ProfessorId = test.ProfessorId,
        TestId = test.Id,
        Attempt = attempt + 1,
        Mark = mark,
        MarkRange = test.MarkRange,
        TestName = test.Name,
        PassDate = DateTime.Now.ToString("MM/dd/yyyy HH:mm"),
    };
    _db.TestResult.Add(newTestResult);
    _db.SaveChanges();
    return attempt;
}
```

Наступним буде контролер TestController. Він відповідає за функціонал експорту тесту у JSON та XML, отримання результатів проходження тестів студентами для викладача.

Для експорту тесту у XML використовується серіалізація – перетворення об'єкта або дерева об'єктів в будь-який формат з тим, щоб потім ці об'єкти можна було відновити з цього формату.

```
[HttpGet("GetConversationXmlByIdentyficator")]
public string GetConversationXmlByIdentyficator(string
identyficator)
{
    var conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
```

```
    var questions = _db.Question.Where(x => x.ConversationId ==
conversations.Id).ToList();
    foreach (var question in questions)
    {
        var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
        question.AnswerOptions = answerOptions;
    }
    conversations.Question = questions;
    using(var stringwriter = new System.IO.StringWriter())
{
    var serializer = new XmlSerializer(conversations.GetType());
    serializer.Serialize(stringwriter, conversations);
        string xmlUtf8 = stringwriter.ToString().Replace("utf-16",
"utf-8");
        return xmlUtf8;
    }
}
```

Для експорту тесту у формат JSON використовується вбудований клас Json, який у свою чергу під капотом також використовує серіалізацію для перетворення об’єкту тесту у JSON-структуру.

```
[HttpGet("GetConversationJsonByIdentyficator")]
public JsonResult GetConversationJsonByIdentyficator(string
identyficator)
{
    var conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
    var questions = _db.Question.Where(x => x.ConversationId ==
conversations.Id).ToList();
    foreach (var question in questions)
    {
        var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
        question.AnswerOptions = answerOptions;
    }
    conversations.Question = questions;
    return Json(conversations);
}
```

Клас TestController також містить у собі метод під назвою GetHistoryPassedConversationsForProfessor для отримання результатів проходження створених викладачем тестів.

```
[Authorize(Roles = "professor")]
[HttpPost("GetHistoryPassedConversationsForProfessor")]
```

```
public List<People> GetHistoryPassedConversationsForProfessor()
{
    var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
    var conversations = _db.Conversations.Where(x => x.ProfessorId
== professor.Id).ToList();
    List<People> students = new List<People>();
    var passedConversations = _db.PassedConversations.Where(x =>
x.ProfessorId == professor.Id).ToList();
    foreach (var passed in passedConversations)
    {
        if (students.FirstOrDefault(x => x.Id == passed.StudentId)
== null)
        {
            students.Add(_db.People.FirstOrDefault(x => x.Id ==
passed.StudentId));
        }
    }
    foreach (var passed in passedConversations)
    {
        foreach (var student in students)
        {
            if (passed.StudentId == student.Id)
            {
                student.PassedConversations.Add(passed);
            }
        }
    }
    return students;
}
```

Для створення ідентифікатора тесту використовується випадковий генератор, який міститься у класі RandomGenerator. Він генерує випадковий символ латиниці за кожну ітерацію, кількість яких вказується при виклику, та складає всі символи у один ідентифікатор.

```
public string RandomString(int length)
{
    StringBuilder str_build = new StringBuilder();
    Random rnd = new Random();
    char letter;
    for (int i = 0; i < length; i++)
    {
        double flt = rnd.NextDouble();
        int shift = Convert.ToInt32(Math.Floor(25 * flt));
        letter = Convert.ToChar(shift + 65);
        str_build.Append(letter);
    }
```

```
    return str_build.ToString();
}
```

Для оновлення інформації про пройдені тести на стороні користувача (викладача), у програмі було використано бібліотеку SignalR. Це така бібліотека яка спрощує процес додавання веб-функцій в режимі реального часу до додатків. Веб-функції, що працюють в режимі реального часу, – це можливість миттєво відправити вміст на підключених клієнтів у міру доступності сервера, а не чекати, поки клієнт запросить нові дані. SignalR надає простий API для створення віддалених викликів процедур (RPC) "сервер-клієнт", які викликають функції JavaScript в клієнтських браузерах з коду .NET на стороні сервера.

При роботи з SignalR на стороні сервера необхідно створити спеціальну сутність – хаб (hub). По суті хаб представляє собою клас, наслідуваний від класу Hub, який може обробляти запити.

```
public class WssHub : Hub
{
    public async Task Conversations(string action, Conversations
conversations)
    {
        await Clients.All.SendAsync("conversationsReceived",
action, conversations);
    }
    public async Task AnswerOptions(string action, AnswerOptions
answerOptions)
    {
        await Clients.All.SendAsync("answerOptionsReceived",
action, answerOptions);
    }
    public async Task Question(string action, Question question)
    {
        await Clients.All.SendAsync("questionReceived", action,
question);
    }
    public async Task PassedConversations(string action,
PassedConversations passedConversations)
    {
        await Clients.All.SendAsync("passedConversationsReceived",
action, passedConversations);
    }
```

Проте для того, щоб SignalR запрацював, потрібно його сконфігорувати. Це робиться через клас Startup.

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services
to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
        services.AddHttpsRedirection(options =>
        {
            options.RedirectStatusCode = (int)HttpStatusCode.PermanentRedirect;
            options.HttpsPort = 443;
        });
        services.Configure<ForwardedHeadersOptions>(options =>
        {
            options.ForwardedHeaders =
                ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto;
        });
        // In production, the Angular files will be served from this directory
        services.AddSpaStaticFiles(configuration =>
        {
            configuration.RootPath = "ClientApp/dist";
        });
        services.AddDbContext<DBContext>();
        services.AddScoped<DBContext>();
        services.AddHostedService<MyService>();
        services.AddSingleton<SignalRClient, SignalRClient>();
        services.AddScoped<ProfessorController, ProfessorController>();
        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(options =>
                {
                    options.RequireHttpsMetadata = false;
                    options.TokenValidationParameters = new
TokenValidationParameters
                    {
                        // укзывает, будет ли валидироваться издатель при
валидации токена
                        ValidateIssuer = true,
                        // строка, представляющая издателя
                        ValidIssuer = AuthOptions.ISSUER,
                        // будет ли валидироваться потребитель токена
                        ValidateAudience = true,
                        // установка потребителя токена
                        ValidAudience = AuthOptions.AUDIENCE,
                        // будет ли валидироваться время существования
                        ValidateLifetime = true,
                        // установка ключа безопасности
                        IssuerSigningKey =
AuthOptions.GetSymmetricSecurityKey(),
                        // валидация ключа безопасности
                        ValidateIssuerSigningKey = true,
                    };
                });
        services.AddSignalR();
```

```
        services.AddControllers().AddXmlSerializerFormatters();

    }

    // This method gets called by the runtime. Use this method to configure the
HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseForwardedHeaders();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseForwardedHeaders();
            // The default HSTS value is 30 days. You may want to change this
for production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        if (!env.IsDevelopment())
        {
            app.UseSpaStaticFiles();
        }
        app.UseRouting();
        app.UseDefaultFiles();
        app.UseStaticFiles();
        app.UseAuthentication();
        app.UseAuthorization();
        app.UseForwardedHeaders(new ForwardedHeadersOptions
        {
            ForwardedHeaders = ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto
        });
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller}/{action=Index}/{id?}");
            endpoints.MapHub<WssHub>("/hub");
        });
        app.UseSpa(spa =>
        {
            // To learn more about options for serving an Angular SPA from
ASP.NET Core,
            // see https://go.microsoft.com/fwlink/?linkid=864501

            spa.Options.SourcePath = "ClientApp";

            if (env.IsDevelopment())
            {
                spa.UseAngularCliServer(npmScript: "start");
            }
        });
    }
}
```

Програмний код основних модулів подано у додатку Б.

3.3 Керівництво користувача

Для роботи з програмною системою користувачеві потрібно мати доступ до мережі Інтернет та браузер. Після переходу за відповідною адресою користувача зустріне сторінка авторизації та реєстрації (рисунок 3.2)

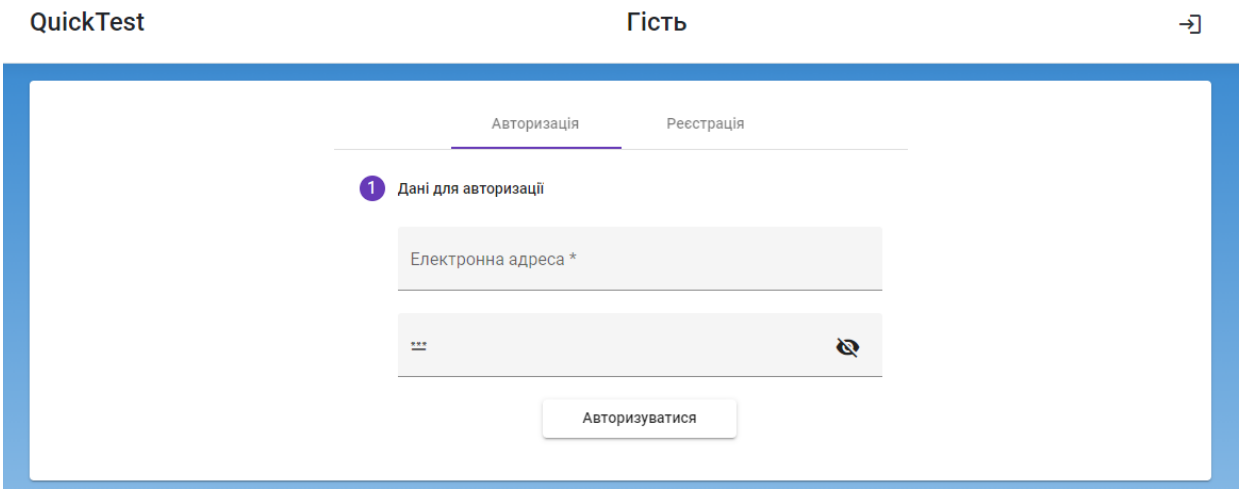


Рисунок 3.2 – сторінка авторизації та реєстрації

Перш ніж авторизуватися, користувач повинен зареєструватися у системі. Реєстрація відбувається на другій вкладці форми (рисунок 3.3). Першим ділом користувач має вказати своє прізвище та ім’я. Після заповненния цих полім, потрібно перейти до наступного пункту – вказання ролі (рисунок 3.4). Далі користувач має вказати адресу електронної пошти та пароль (рисунок 3.5). Поле для введення адреси електронної пошти валідується і приймає лише значення по такому шаблону <текст та/або цифри>@<текст>.<домен>. У разі, якщо введений текст не відповідає шаблону – система сповістить про це користувача. Поле введення паролю має кнопку для прихования чи показу паролю. Це одночасно спрощує введення паролю, якщо він занадто довгий чи складний, а з іншої сторони його можна приховати задля безпеки. Після натискання на кнопку «Зареєструватися» дані відправляються на сервер. Якщо реєстрація відбулася успішно, користувачеві буде видане повідомлення.

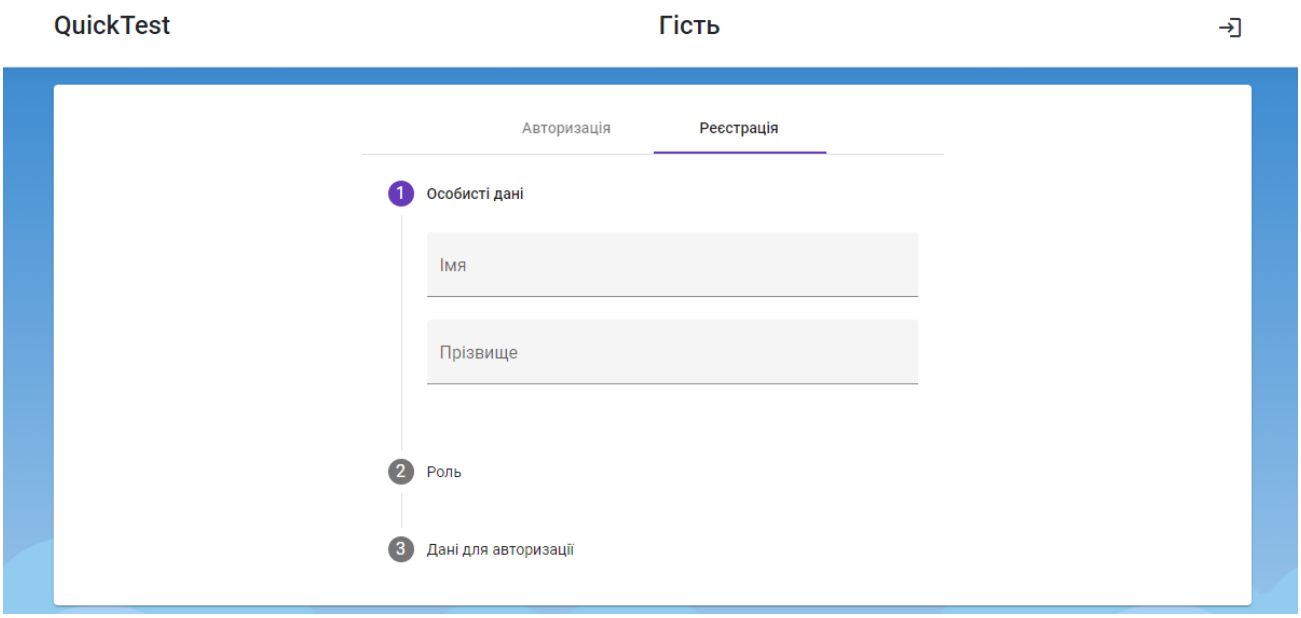


Рисунок 3.3 – Форма реєстрації

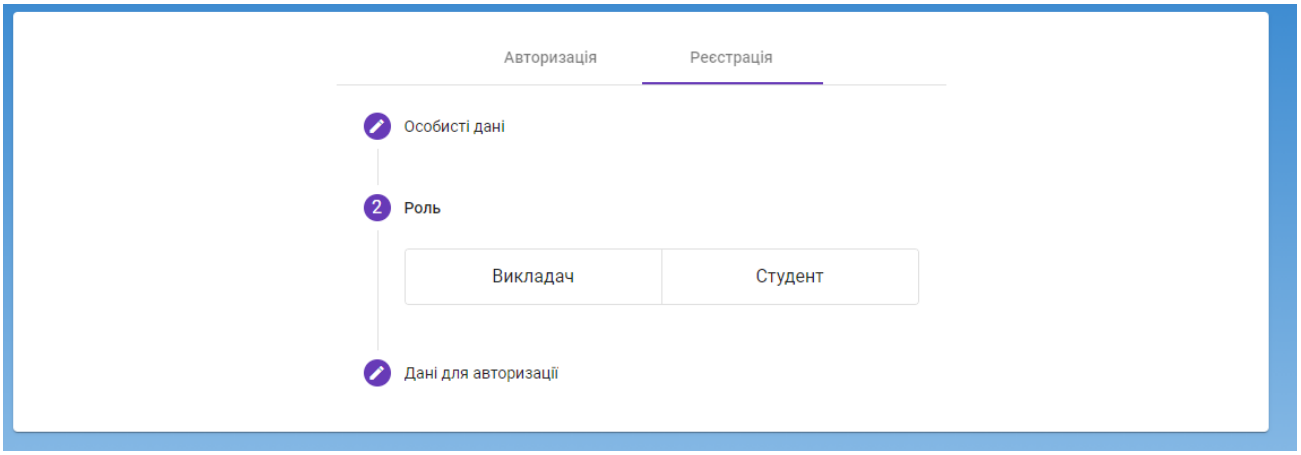


Рисунок 3.4 – Пункт вибору ролі користувача

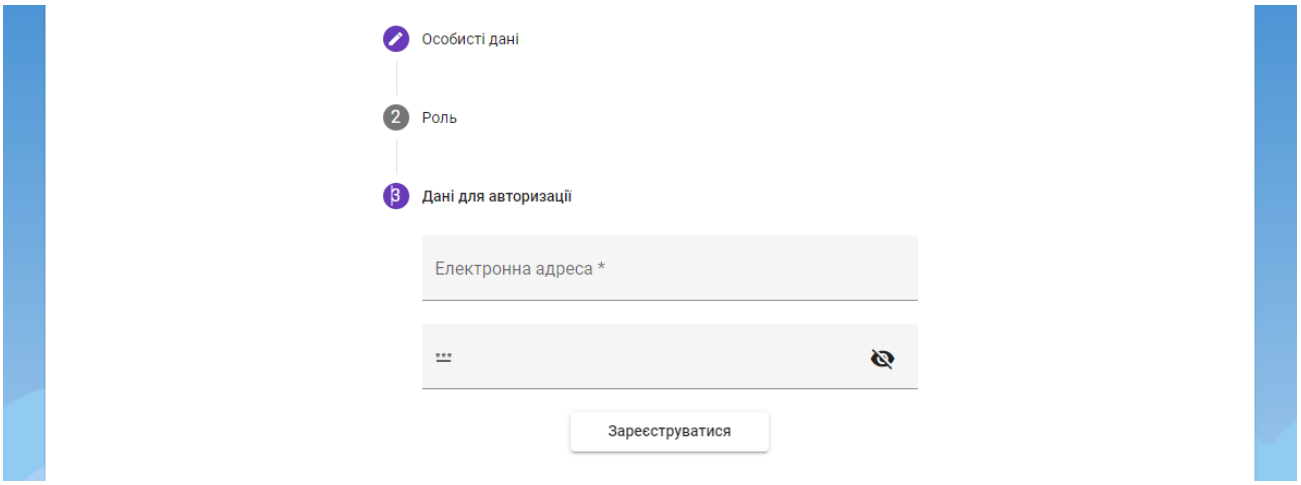


Рисунок 3.5 – Пункт введення електронної пошти і паролю

Після авторизації користувачу з роллю викладач доступна вкладкака перегляду створених ним тестів (рисунок 3.6). Вона організована по типу списку форм Кожна форма це окремий тест. Форма складається з двох вкладок: «Інформація» і «Запитання». У вкладці інформація наявна основна інформація про тест – його назва, посилання для поширення, унікальний ідентифікатор та кнопки імпорту тесту в інші формати.

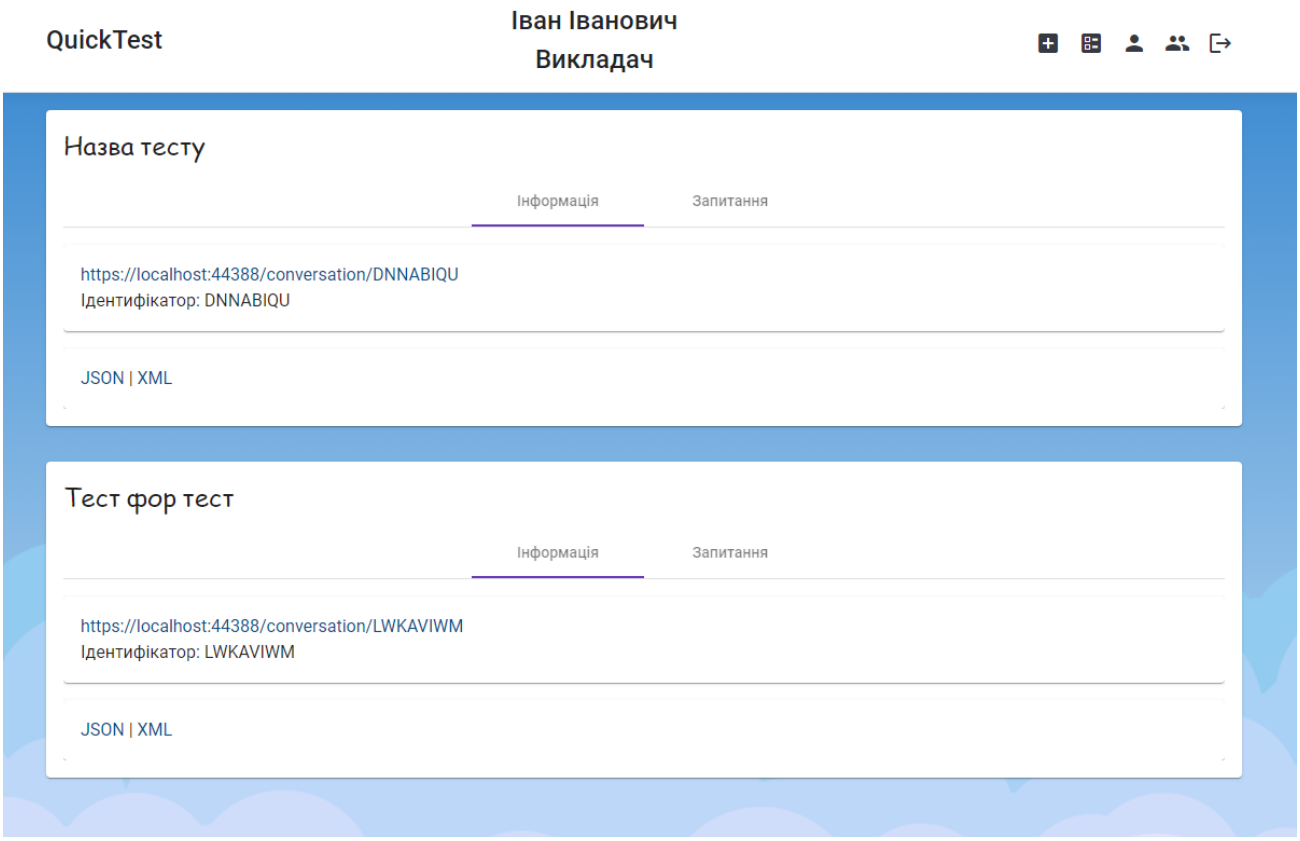


Рисунок 3.6 – Вкладка перегляду створених тестів

Наступна вкладка форми – «Запитання» (рисунок 3.7). У ній міститься конструктор тесту, з яким можна створювати питання та додавати варіанти відповіді. Спочатку у відповідне поле вводимо запитання та натискаємо на кнопку з знаком «+». Далі натискаємо на це запитання, і будуть виїжджати варіанти відповідей. Можна ввести вірант нової відповіді та натиснути кнопку «+1), після чого варіант відповіді буде додано. Для вибору правильної відповіді варто лише натиснути на відповідний варіант, і його кружок стане жовтим.

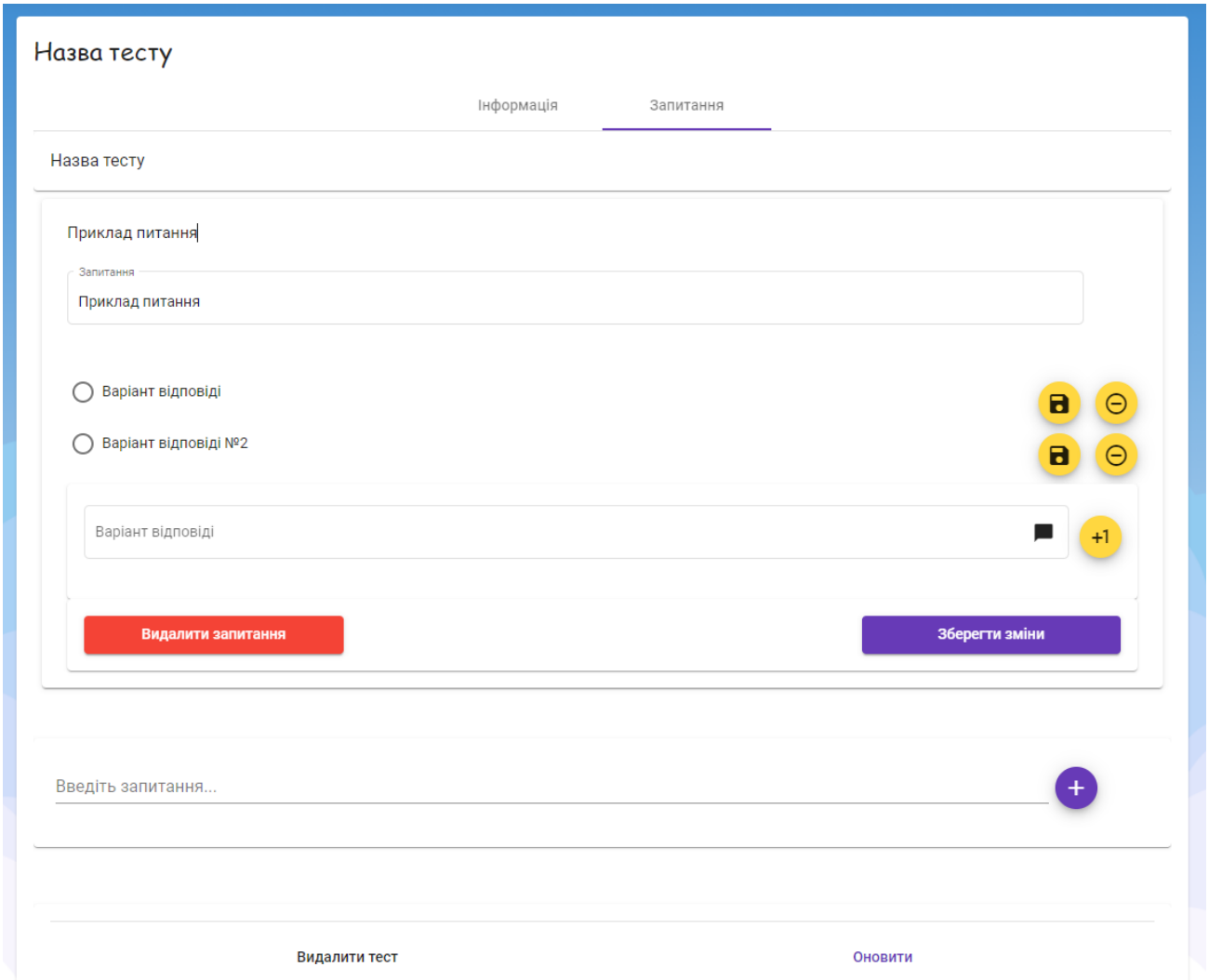


Рисунок 3.7 – Вкладка форми для створення та редагування тесту

Також у цій вкладці доступні кнопки для видалення тесту та оновлення інформації з бази даних. Варіанти відповіді також можна видаляти натискаючи на кнопку «-».

Для викладача доступна сторінка перегляду результатів проходження тестів (рисунок 3.8). Доступ до неї наявний з верхньої панелі сайту. Вона організована у вигляді списку студентів. Натискаючи на кожного студнта, буде випадати виїжджаючий список з всіма результатами цього студента по пройденим ним тестам. Інформація організована у форматі назва тесту – оцінка – спроба – дата проходження. Дана форма має дві вкладки. Перша – для перегляду нрезультатів по зараз активних тестах. Інша призначена для перегляду результатів проходження вже закритих тестів.

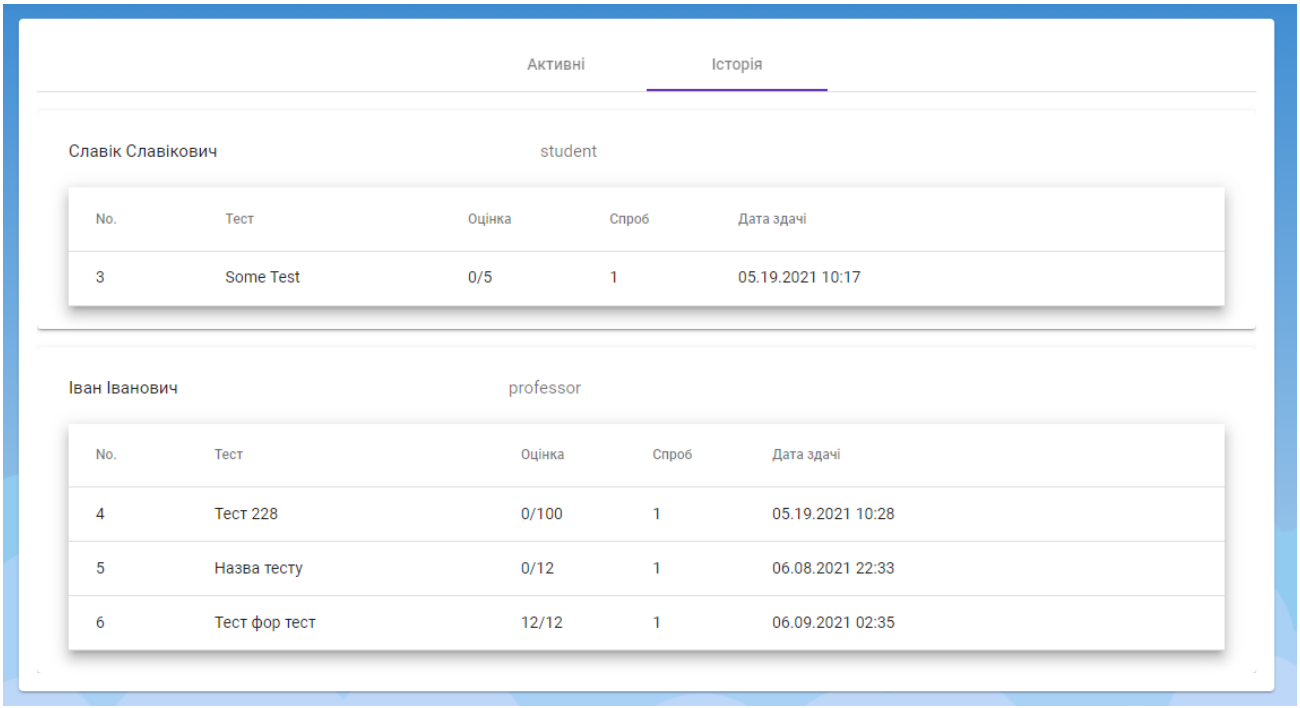


Рисунок 3.8 – Форма результатів проходження тестів

Тепер розглянемо можливості користувача із роллю студент. Після авторизації такого користувача зустрічає форма завантаження тесту за ієнтифікатором (рисунок 3.9).

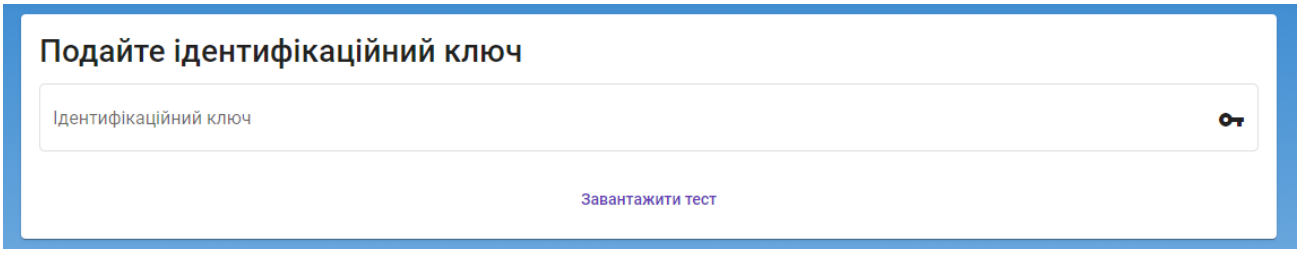


Рисунок 3.9 – Форма завантаження тесту за ідентифікатором

Після введення коректного ідентифікатора студенту відкриється форма для проходження тесту (рисунок 3.10). Структура цієї форми така: назва тесту, далі іде прізвище та ім’я викладача, потім система оцінювання та поточна спроба студента. Після вибору всіх правильних відповідей, потрібно натиснути кнопку «Завершити тест». Система перевірить відповіді та визначить оцінку студента за цей тест.

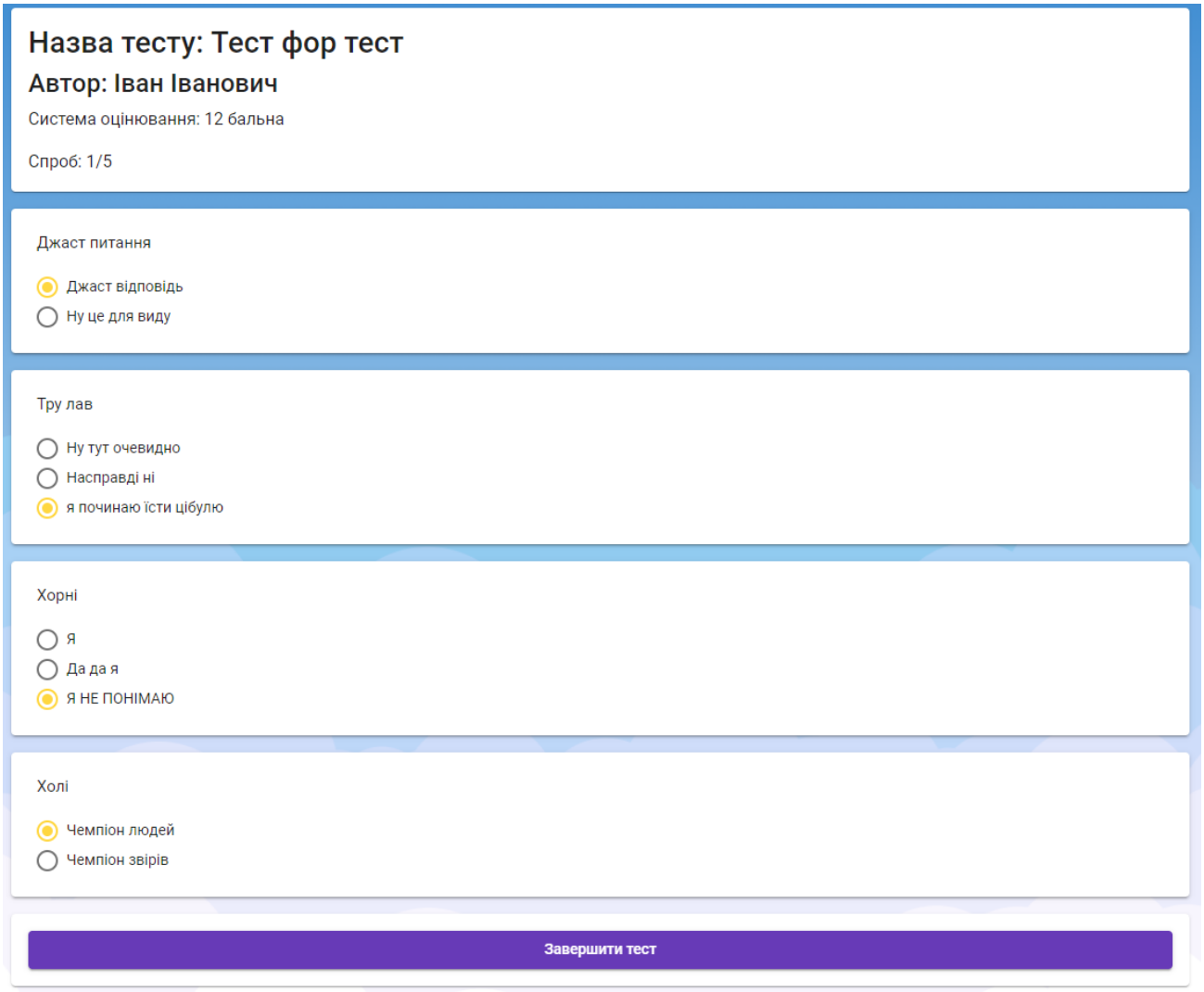
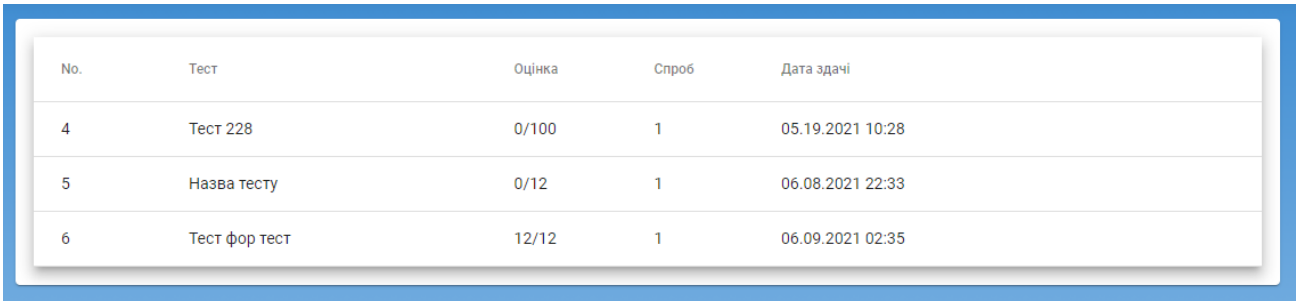


Рисунок 3.10 – Форма для проходження тесту

Студенту також доступна вкладка перегляду своїх результатів пройдених тестів (рисунок 3.11). Доступна вона також із вехньої навігаційної стрічки. Організована у вигляді форми із списком пройдених тестів.



No.	Тест	Оцінка	Спроб	Дата здачі
4	Тест 228	0/100	1	05.19.2021 10:28
5	Назва тесту	0/12	1	06.08.2021 22:33
6	Тест фор тест	12/12	1	06.09.2021 02:35

Рисунок 3.11 – Форма результатів пройдених студентом тестів

3.4 Технічні характеристики програмної системи

Для забезпечення стабільної роботи системи на сервері, останній має відповідати мінімальним технічним вимогам, а саме:

– Мінімум 4 ГБ оперативної пам’яті;

– Мінімум 6 ГБ вільного місця на фізичному носії;

– Мінімальний об’єм трафіку на місяць – 4 ГБ;

– Процесор на 4 ядра з мінімальною частотою не нижче 2.5 Ггц.

Користувачеві для комфортного використання програмної системи варто лише мати веб-браузер та доступ до мережі інтернет. Клієнтська частина є досить простою, тому великої потужності для її відображення не потрібно, а тому користуватися системою можна майже зі всіх пристроїв

Отже, у процесі написання даного розділу було розібрано та описано реалізацію основних модулів системи. Переглянуто способи і варіанти взаємодії клієнтської та серверної частин. Розібрано спосіб та метод для авторизації користувачів та його реалізацію. Проведено та описано аналіз технологій, які використовувалися при написанні програми. Також написано короткий посібник користувача, який описує основні аспекти при використанні програми. Та насамкінець було описано мінімальні технічні вимоги до сервера та клієнтської машини.

4 ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

4.1 Вибір та обґрунтування методів тестування веб-додатка

Метою використання численних методологій тестування у процесі розробки є переконання у тому, що розроблюване програмне забезпечення може успішно працювати в різних середовищах та на різних платформах. Зазвичай їх можна розділити на функціональне та нефункціональне тестування. Функціональне тестування передбачає перевірку програми на відповідність бізнес-вимогам. Він включає всі типи тестів, розроблені для того, щоб гарантувати, що кожна частина програмного забезпечення поводиться належним чином, використовуючи варіанти використання, надані командою розробників або бізнес-аналітиком. Ці методи тестування зазвичай проводяться в порядку і включають:

- unit- тестування;
- інтеграційне тестування;
- системне тестування;
- приймальне тестування.

Нефункціональні методи тестування включають усі типи тестів, орієнтовані на експлуатаційні аспекти програмного забезпечення. До них належить таке тестування:

- тестування продуктивності;
- тестування безпеки;
- тестування зручності використання;
- тестування на сумісність.

Ключем до випуску високоякісного програмного забезпечення, яке легко може бути прийнято кінцевими користувачами, – це створення надійної структури тестування, яка реалізує як функціональні, так і нефункціональні методології тестування програмного забезпечення.

Модульне тестування – це перший рівень тестування, і його часто проводять самі розробники. Це процес забезпечення того, щоб окремі

компоненти програмного забезпечення на рівні коду функціонують та працюють так, як вони мали б працювати.

Інтеграційне тестування. Після ретельного випробування кожного блоку він інтегрується з іншими блоками для створення модулів або компонентів, призначених для виконання конкретних завдань або видів діяльності. Потім вони перевіряються як групові за допомогою інтеграційного тестування, щоб забезпечити поводження належним чином цілих сегментів програми (тобто взаємодія між блоками безперебійна).

Системне тестування – це метод тестування по принципу «чорного ящика», що використовується для оцінки завершеної та інтегрованої системи в цілому для забезпечення її відповідності визначеним вимогам.

Приймальне тестування є останнім етапом функціонального тестування і використовується для оцінки того, чи готовий остаточний компонент програмного забезпечення чи ні. Це передбачає те, що створений продукт відповідає оригінальним бізнес-критеріям та відповідності потребам кінцевого користувача. Тестування продуктивності – це нефункціональна методика тестування, яка використовується для визначення того, як програма поводитиметься за різних умов. Мета такого тестування це перевірити його чутливість та стабільність у реальних ситуаціях користувача. Тестування продуктивності можна розділити на чотири типи.

Тестування навантаження – це процес надання більшої кількості змодельованого навантаження на ваше програмне забезпечення, додаток чи веб-сайт, щоб перевірити, чи зможе воно впоратися з ним.

Стрес-тестування робить крок далі і використовується для оцінки реакції вашого програмного забезпечення на пікове навантаження або поза ним. Метою стрес-тестування є спеціальне перевантаження програми, поки вона не зламається, застосовуючи як реалістичні, так і нереальні сценарії завантаження. За допомогою стрес-тестування можна знайти точку відмови створеного програмного забезпечення.

Тестування на витривалість використовується для аналізу поведінки програми при певній кількості імітованого навантаження протягом тривалого часу. Найважливішим елементом тестування на витривалість є те, що воно допомагає виявити витоки пам’яті.

Перевірка шипами – це тип перевірки навантаження, який використовується для визначення того, як ваше програмне забезпечення реагуватиме на значно більші сплески одночасної діяльності користувача чи системи протягом різного періоду часу. В ідеалі це допоможе вам зрозуміти, що станеться, коли навантаження раптово і різко збільшиться.

Тестування безпеки – це нефункціональний метод тестування програмного забезпечення, який використовується для визначення того, чи захищені інформація та дані в системі. Тестування зручності використання – це метод тестування, який вимірює простоту використання програми з точки зору кінцевого користувача і часто виконується на етапах перевірки системи або приймання.

Оскільки розроблене програмне забезпечення складається з різни компонентів, найкраще для перевірки його працездатності буде підходити модульне тестування програмного забезпечення.

4.2 Перевірка на помилки за допомогою unit-тестів

Тестування програмної системи розпочнеться із модульного тестування і перевірки: правильність роботи веб-додатку, коректне виконання його функцій, коректна обробка вхідних та вихідних даних. Ідея полягає в тому, щоб писати тести для кожної нетривіальною функції або методу. Це дозволяє досить швидко перевірити, чи не призвела чергова зміну коду до регресії, тобто до появи помилок в уже протестованих місцях програми, а також полегшує виявлення і усунення таких помилок.

Спочатку сформуємо таблиці, в яких будуть описані сценарії тестування. Таблиці складатимуться із таких стовпців: назва тестового сценарію; тестований модуль; тестований метод; вхідні дані; очікуваний результат;

У них будуть описані тестові сценарії, вхідні дані для них та очікуваний результат від системи. Розроблені сценарії тестування будуть міститися у таблиці 4.1 поданій нижче.

Таблиця 4.1 – Тестові сценарії основного функціоналу

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання даних користувача	AccountController	GetUserInfromation	Унікальний номер користувача	Екземпляр класу User із заповненими всіма полями
Реєстрація користувача	AccountController	UserRegistry	Прізвище ім’я користувача, його роль та логін і пароль	Вся інформація про користувача записана у базу даних
Видача JWT-токену	AccountController	Token	Логін і пароль користувача	Токен коректно дешифрується і надає доступ користувачеві до системи
Імпорт тесту в JSON	TestController	ImportJson	Екземпляр класу Test	Записаний файл JSON
Імпорт тесту в XML	TestController	ImportXML	Екземпляр класу Test	Записаний файл XML
Експорт тесту з JSON	TestController	ExportJson	Файл JSON	Екземпляр класу Test
Експорт тесту з XML	TestController	ExportXML	Файл XML	Екземпляр класу Test
Отримання тесту за ідентифікатором	TestController	GetTestByIdentyficator	Ідентифікатор тесту	Екземпляр класу Test

Кінець таблиці 4.1

Перевірка тесту	TestControll er	PassTest	Екземпляр класу PassTest	Екземпляр класу PassedTest
Генерація ідентифікатору	RandomGen erator	GenerateIdent	Кількість символів в ідентифікаторі	Ідентифікатор потрібної довжини
Оновлення тесту	ProfessorCon troller	UpdateTest	Список змін та ідентифікатор тесту	Збережені зміни в базі даних
Підключення бази даних	DBContext	OnConfigerat ion	Адрес для підключення бази даних	Успішно пройдене підключення
Отримання результатів проходження тестів	StudentContr oller	GetResults	Унікальний номер студента	Список пройдених студентом тестів

4.3 Аналіз результатів тестування веб-додатка

У ході виконання тестування було проведено модульні тести. Результати тестування та їх опис наведено у таблиці 4.2.

Таблиця 4.2 – Результати проведення модульного тестування

Назва сценарію	Метод	Вхідні дані	Очікуваний результат	Результат
1	2	3	4	5
Отримання даних користувача	GetUserInfro mation	Унікальний номер користувача	Екземпляр класу User із заповненими всіма полями	Правильно
Реєстрація користувача	UserRegistry	Прізвище ім’я користувача, його роль та логін і пароль	Вся інформація про користувача записана у базу даних	Правильно
Видача JWT-токену	Token	Логін і пароль користувача	Токен коректно дешифрується і надає доступ користувачеві до системи	Правильно

Кінець таблиці 4.2

Імпорт тесту в JSON	ImportJson	Екземпляр класу Test	Записаний файл JSON	Правильно
Імпорт тесту в XML	ImportXML	Екземпляр класу Test	Записаний файл XML	Правильно
Експорт тесту з JSON	ExportJson	Файл JSON	Екземпляр класу Test	Правильно
Експорт тесту з XML	ExportXML	Файл XML	Екземпляр класу Test	Правильно
Отримання тесту за ідентифікатором	GetTestByIdentyficator	Ідентифікатор тесту	Екземпляр класу Test	Правильно
Перевірка тесту	PassTest	Екземпляр класу PassTest	Екземпляр класу PassedTest	Правильно
Генерація ідентифікатору	GenerateIdent	Кількість символів в ідентифікаторі	Ідентифікатор потрібної довжини	Правильно
Оновлення тесту	UpdateTest	Список змін та ідентифікатор тесту	Збережені зміни в базі даних	Правильно
Підключення бази даних	OnConfigeration	Адрес для підключення бази даних	Успішно пройдене підключення	Правильно
Отримання результатів проходження тестів	GetResults	Унікальний номер студента	Список пройдених ним тестів	Правильно

Отже, у ході написання даного розділу було описано та проведено модульне тестування веб-додатку і в результаті з’ясовано, що програмне забезпечення працює правильно і задовольняє усім вимогам та потребам, описаним раніше.

ВИСНОВКИ

По звершенню роботи по дипломному проектуванню можна винести такі висновки стосовно зробленого.

У першому розділі було проаналізовано основні проблеми оцінювання якості знань студентів, які виникають під час дистанційного навчання, а також запропоновано вирішення її – створення програмної системи для швидкого та зручного тестування.

Попри те, було проаналізовано інше наявне на ринку програмне забезпечення, виділено їх основні плюси та мінуси.

Вимоги до програмної системи були висвітлені у вигляді діаграми варіантів використання та витікали також з аналізу наявного на ринку програмного забезпечення.

У другому розділі було проаналізовано та порівняно основні архітектурні рішення для побудови веб-додатку, та вибрано клієнт-серверну архітектуру з моделлю взаємодії тонкого клієнта.

Також було проаналізовано найпопулярніші архітектури для побудови додатків, і визначено, що найкращим рішенням буде монолітна архітектура додатку із використанням патерну проектування MVC. У кінці розділу було проведено проектування серверної та клієнтських частин додатку та описано засоби та методи для їх реалізації.

У процесі написання третього розділу у було розібрано та описано реалізацію основних модулів системи. Переглянуто способи і варіанти взаємодії клієнтської та серверної частин. Розібрано спосіб та метод для авторизації прав користувачів та його реалізацію. Проведено та описано аналіз технології та програмній, які використовувалися при написанні програми. Також написано короткий посібник користувача, який описує основні аспекти при використанні програми. Та насамкінець було і визначено мінімальні технічні вимоги для запуску і безпроблемної роботи створюваного програмного забезпечення

В четвертому розділі дипломного проекту було здійснено аналіз основних способів і варіантів тестування програмної системи та обрано модульне тестування для створеного додатку. Було складено та описано сценарії тестування додатку та реалізовано їх на практиці.

За результатами проведеного тестування було сформовано таблицю та визначено, що програмне забезпечення працює правильно та відповідає усім поставленим до нього вимогам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Л. П. Бедратюк. Дипломний проект : методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун, О. М. Яшина. – Хмельницький: ХНУ, 2020. – 77 с.

2. Client/Server Architecture [Online] / techopedia. – Available: https://www.techopedia.com/definition/438/clientserver-architecture

3. THE CONCEPT OF THIN AND THICK CLIENTS [Online] / TestMatick. – Available: https://testmatick.com/the-concept-of-thin-and-thick-clients/

4. Monolithic vs Microservices architecture [Online] / GeeksForGeeks. – Availale: https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/

5. Chapter 1. Layered Architecture [Online] / O`Reilly. – Available: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

6. 4 типа архитектуры программного обеспечения [Электронный ресурс] / Nuancesprog. – Режим доступа: https://nuancesprog.ru/p/12019/

7. Что такое Microsoft .Net Framework [Электронный ресурс] / Microsoft.NET Framework. – Режим доступа: http://net-framework.ru/article/chto-takoe

8. .NET Framework и ASP.NET — платформы для веб-разработки [Электронный ресурс] / Web Creator. – Режим доступа: https://web-creator.ru/articles/dot_net_and_asp

9. Что такое Entity Framework? [Электронный ресурс] / интернет-технологии.ру. – Режим доступа: https://www.internet-technologies.ru/articles/chto-takoe-entity-framework.html

10. Авторизация с помощью JWT-токенов [Электронный ресурс] / Metanit. – Режим доступа: https://metanit.com/sharp/aspnet5/23.7.php

11. SignalR Core [Электронный ресурс] / Metanit. – Режим доступа: https://metanit.com/sharp/aspnet5/30.1.php

12. MVC Framework – Introduction [Online] / Tutorialspont. – Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

13. How the Model View Controller Architecture Works – MVC Explained [Online] / freeCodeCamp. – Available: https://www.freecodecamp.org/news/model-view-architecture/

14. What is Angular?: Architecture, Features, and Advantages [Online] / SimpliLearn. – Availale: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular

15. Angular Material Tutorial [Online] / tutorialspoint. – Available: https://www.tutorialspoint.com/angular_material/index.htm

16. Чистий код / Роберт С. Мартін – Фабула, 2019 – 419 с.

17. CLR via C# (Developer Reference) 4th Edition / Jeffrey Richter – Microsoft Press – 896 p

.

ДОДАТОК А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки програмної системи тестування студентів з можливістю імпорту тестових завдань з інших форматів

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів.

2 Призначення розробки

Метою проекту – розробка комп’ютеризованої системи тестування для контролю одержаних знань, умінь і навиків студентами.

Вкажемо функціональне і експлуатаційне призначення комплексної системи, що розробляється:

а) функціональне призначення: аналіз одержаних знань, умінь і навиків студентами, швидке створення чи імпорт готових тестів.

б) експлуатаційне призначення: система доступна з будь якого пристрою, який підключений до мережі Інтернет та має встановлений браузер.

3 Вимоги до програми

При реалізації і використовуванні статистики по тестуванню знань студентів повинні бути враховані вимоги до функціональних характеристик, надійності проекту, умов експлуатації, складу і параметрів технічних засобів, апаратної і програмної сумісності.

3.1 Вимоги до функціональних характеристик

Програмна система повинна надавати наступні можливості:

– об'єктивність оцінки, оскільки в тестовому контролі вплив суб'єктивних чинників (таких, як обізнаність екзаменатора про поточну успішність

іспитується, облік його поведінки на учбових заняттях і т.п.) виключений;

– достовірність інформації про об'єм засвоєного матеріалу і про рівень його засвоєння;

– ефективність – можна одночасно тестувати велике число учнів, причому перевірка результатів при цьому вироблятися набагато легше і швидше, ніж при традиційному контролі;

– надійність – тестова оцінка однозначна;

– порівнянність результатів тестування для різних груп студентів, які навчаються за різними програмами, з використанням різних методів і організаційних форм навчання.

3.2 Вимоги до надійності

Система повинна виконувати наступні вимоги до надійності:

– підтримувати функції захисту від несанкціонованого доступу (розділення прав доступу до інформації баз даних для користувачів системи);

– обробляти помилкові дії користувача і повідомляти його про це;

– виключати аварійні ситуації, які прямо або побічно можуть привести до псування апаратної, програмної або інформаційної складової оточення користувача.

3.3 Умови експлуатації

Для користуванням програмною системою користувачу потрібен лише браузер, тому неважливо яким пристроєм де і як ви користуєтеся. Єдина умова – підключення до мережі Інтернет.

4 Вимоги до інформаційної і програмної сумісності

Для створення серверної частини програмної системи будуть використовуватися наступні технології:

- .NET C# – об’єктно-орієнтована мова програмування високого рівня;
- ASP.NET Core – фреймворк для створення веб-додатків;

- SQLite – полегшена реляційна система керування базами даних;
- Entity Framework Core – ORM, технологія доступу та керування реляційними даним, та перетворення їх у об’єктно-орієнтовану структуру;
- Angular – фреймворк для написання клієнтської частини веб-додатків в якості односторінкових;

Для створення клієнтської частини будуть використовуватися технології HTML та CSS (для верстки та стилізації додатку).

5 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– текст програми з відповідними коментарями та поясненнями;

– опис програми – відомості про функціонування програми;

– технічне завдання;

– короткий посібник (довідкова інформація) користувачу;

– керівництво програмісту.

6 Стадії і етапи розробки

Розробка програмного продукту проходить декілька стадій і етапів, які представлені в таблиці 2.3.

Таблиця 2.3 – Стадії і етапи розробки ПП

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.21 – 31.01.21	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.21 – 14.02.21	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; розробка загальної алгоритмічної структури системи, що буде розроблюватися

Кінець таблиці 2.3

1	2	3
Технічний проект 15.02.21 – 28.02.21	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми; остаточне визначення конфігурації технічних засобів
Робочий проект 01.03.21 – 10.04.21	Розробка програмного забезпечення	Реалізація програмного забезпечення; відладка; проведення попереднього тестування
Розробка програмної документації 11.04.21 – 20.04.21	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 21.04.21 – 30.04.21	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і передача програмного забезпечення; навчання персоналу використовуванню програмного забезпечення; внесення коректувань в програмне забезпечення і документацію

7 Порядок контролю і приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування системи. Після закінчення розробки системи повинні бути проведене тестування на захист від некоректного введення.

ДОДАТОК Б
(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

Клас Startup

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to
add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
        services.AddHttpsRedirection(options =>
        {
            options.RedirectStatusCode =
(int)HttpStatusCode.PermanentRedirect;
            options.HttpsPort = 443;
        });
        services.Configure<ForwardedHeadersOptions>(options =>
        {
            options.ForwardedHeaders =
                ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto;
        });
        // In production, the Angular files will be served from
this directory
        services.AddSpaStaticFiles(configuration =>
        {
            configuration.RootPath = "ClientApp/dist";
        });
        services.AddDbContext<DBContext>();
        services.AddScoped<DBContext>();
        services.AddHostedService<MyService>();
        services.AddSingleton<SignalRClient, SignalRClient>();
        services.AddScoped<ProfessorController,
ProfessorController>();

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(options =>
                {
                    options.RequireHttpsMetadata = false;
                    options.TokenValidationParameters = new
TokenValidationParameters
                    {
                        // укзывает, будет ли валидироваться
издатель при валидации токена
```

```
                        ValidateIssuer = true,
                        // строка, представляющая издателя
                        ValidIssuer = AuthOptions.ISSUER,
                        // будет ли валидироваться потребитель
токена
                        ValidateAudience = true,
                        // установка потребителя токена
                        ValidAudience = AuthOptions.AUDIENCE,
                        // будет ли валидироваться время
существования
                        ValidateLifetime = true,
                        // установка ключа безопасности
                        IssuerSigningKey =
AuthOptions.GetSymmetricSecurityKey(),
                        // валидация ключа безопасности
                        ValidateIssuerSigningKey = true,
                    };
                });
        services.AddSignalR();
        services.AddControllers().AddXmlSerializerFormatters();

    }

    // This method gets called by the runtime. Use this method to
configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseForwardedHeaders();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseForwardedHeaders();
            // The default HSTS value is 30 days. You may want to
change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        if (!env.IsDevelopment())
        {
            app.UseSpaStaticFiles();
        }
        app.UseRouting();
        app.UseDefaultFiles();
        app.UseStaticFiles();
        app.UseAuthentication();
```

```
        app.UseAuthorization();
        app.UseForwardedHeaders(new ForwardedHeadersOptions
        {
            ForwardedHeaders = ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto
        });
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller}/{action=Index}/{id?}");
            endpoints.MapHub<WssHub>("/hub");
        });
        app.UseSpa(spa =>
        {
            // To learn more about options for serving an Angular
SPA from ASP.NET Core,
            // see https://go.microsoft.com/fwlink/?linkid=864501

            spa.Options.SourcePath = "ClientApp";

            if (env.IsDevelopment())
            {
                spa.UseAngularCliServer(npmScript: "start");
            }
        });
    }
}
```

Клас SignalRClient

```
public class SignalRClient
{
    public readonly HubConnection connection = new
HubConnectionBuilder()
                    .WithUrl("https://localhost:44388/Hub")
                    .WithAutomaticReconnect()
                    .Build();
    public async Task Start()
    {
        await connection.StartAsync();
    }
    public void Resive()
    {
        connection.On("passedConversationsReceived", (string
action, Conversations PassedConversations) =>
        {
            Console.WriteLine(action);
        });
```

```
        connection.On("questionReceived", (string action, Question
PassedConversations) =>
        {
            Console.WriteLine(action);
        });

        connection.On("answerOptionsReceived", (string action,
AnswerOptions PassedConversations) =>
        {
            Console.WriteLine(action);
        });

        connection.On("conversationsReceived", (string action,
PassedConversations PassedConversations) =>
        {
            Console.WriteLine(action);
        });
    }
}
```

Клас RandomGenerator

```
public class RandomGen
{
    public string RandomString(int length)
    {
        StringBuilder str_build = new StringBuilder();
        Random rnd = new Random();
        char letter;
        for (int i = 0; i < length; i++)
        {
            double flt = rnd.NextDouble();
            int shift = Convert.ToInt32(Math.Floor(25 * flt));
            letter = Convert.ToChar(shift + 65);
            str_build.Append(letter);
        }
        return str_build.ToString();
    }
}
```

Клас AuthOptions

```
public class AuthOptions
{
    public const string ISSUER = "QuickTest"; // издатель токена
```

```
    public const string AUDIENCE = "QuickTest"; // потребитель
токена
    const string KEY = "QuickTestSecretKey";   // ключ для шифрации
    public const int LIFETIME = 525948; // время жизни токена - 1
минута (525948мин = год)
    public static SymmetricSecurityKey GetSymmetricSecurityKey()
    {
        return new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));
    }
}
```

Клас AccountController

```
[ApiController]
[Route("[controller]")]
public class AccountController : Controller
{
    private readonly SignalRClient _signalRClient;

    private readonly DBContext _db;
    public AccountController(SignalRClient signalRClient, DBContext
db)
    {
        _signalRClient = signalRClient;
        _db = db;
    }
    [HttpPost("token")]
    public IActionResult Token(string username, string password)
    {
        People person = _db.People.FirstOrDefault(x => x.Login ==
username && x.Password == password);
        var identity = GetIdentity(username, password, person);
        if (identity == null)
        {
            ModelState.AddModelError("errorMessage", "Invalid
username or password.");
            return BadRequest(ModelState);
        }
        var now = DateTime.UtcNow;
        // создаем JWT-токен
        var jwt = new JwtSecurityToken(
                issuer: AuthOptions.ISSUER,
                audience: AuthOptions.AUDIENCE,
                notBefore: now,
                claims: identity.Claims,
                expires:
now.Add(TimeSpan.FromMinutes(AuthOptions.LIFETIME)),
```

```
                signingCredentials: new
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256));
        var encodedJwt = new
JwtSecurityTokenHandler().WriteToken(jwt);
        var response = new
        {
            access_token = encodedJwt,
            email = identity.Name,
            username = $"{person.FirstName} {person.LastName}",
            role = person.Role
        };
        return Json(response);
    }

    private ClaimsIdentity GetIdentity(string username, string
password, People people){
        if (people != null)
        {
            var claims = new List<Claim>
            {
                new Claim(ClaimsIdentity.DefaultNameClaimType,
people.Login),
                new Claim(ClaimsIdentity.DefaultRoleClaimType,
people.Role),
            };
            ClaimsIdentity claimsIdentity =
            new ClaimsIdentity(claims, "Token",
ClaimsIdentity.DefaultNameClaimType,
                ClaimsIdentity.DefaultRoleClaimType);
            return claimsIdentity;
        }
        // если пользователя не найдено
        return null;}
    [HttpGet("Register")]
    public IActionResult Register(string login, string password,
string firstName, string lastName, string role)
    {
        var persons = _db.People.FirstOrDefault(x => x.Login ==
login);
        if (persons != null) return BadRequest(new { errorText =
"Email already registered" });
        var user = new People
        {
            Login = login,
            Password = password,
            FirstName = firstName,
            LastName = lastName,
            Role = role
        };
        _db.People.Add(user);
        _db.SaveChanges();
```

```
        return Ok();
    }
}
```

Клас TestController

```
[ApiController]
[Route("[controller]")]
public class ConversationsController : Controller
{
    private readonly SignalRClient _signalRClient;

    private readonly DBContext _db;

    private ProfessorController _professorController;

    public ConversationsController(SignalRClient signalRClient,
DBContext db, ProfessorController professorController)
    {
        _signalRClient = signalRClient;
        _db = db;
        _professorController = professorController;
    }
    [Authorize]
    [HttpPost("PassConversation")]
    public int? PassConversation(int conversationId, int
professorId, float mark)
    {
        var student = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var conversation = _db.Conversations.FirstOrDefault(x =>
x.Id == conversationId);
        var passedConversationss = _db.PassedConversations.Where(x
=> x.ConversationId == conversationId && x.StudentId ==
student.Id).ToList();
        int attempt = (int)(1);
        if (passedConversationss != null)
        {

            foreach (var passedConversation in
passedConversationss)
            {
                if (passedConversation.Attempt >=
conversation.HowManyAttempt)
                {
                    return null;
                }
                attempt = (int)(passedConversation.Attempt + 1);
            }
        }
        var passedConversations = new PassedConversations
```

```
        {
            StudentId = student.Id,
            StudentName = $"{student.FirstName}
{student.LastName}",
            ProfessorId = professorId,
            ConversationId = conversationId,
            Attempt = attempt,
            Mark = mark,
            MarkRange = conversation.MarkRange,
            ConversationName = conversation.Name,
            PassDate = DateTime.Now.ToString("MM/dd/yyyy HH:mm"),
        };
        _db.PassedConversations.Add(passedConversations);
        _db.SaveChanges();
        return attempt;
    }

    [HttpGet("GetConversationJsonByIdentyficator")]
    public JsonResult GetConversationJsonByIdentyficator(string
identyficator)
    {
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
        var questions = _db.Question.Where(x => x.ConversationId ==
conversations.Id).ToList();
        foreach (var question in questions)
        {
            var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
            question.AnswerOptions = answerOptions;
        }
        conversations.Question = questions;
        return Json(conversations);
    }

    [Authorize]
    [HttpPost("ImportJson")]
    public async Task<IActionResult> ImportJson([FromBody]
Conversations request)
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        if (_db.Conversations.FirstOrDefault(x => x.Name ==
request.Name) == null)
            if (request != null)
            {
                await _professorController.CreateConversation(
professor.Login, request.Name, request.HowManyAttempt,
request.MarkRange);
                if (request.Question != null)
                {
```

```
                    var convercetion =
_db.Conversations.FirstOrDefault(x => x.ProfessorId == professor.Id
&& x.Name == request.Name);
                    foreach (var question in request.Question)
                    {
                        await
_professorController.CreateQuestion(professor.Login,
convercetion.Id, question.Text, question.TrueAnswerId);
                        if (question.AnswerOptions != null)
                        {
                            var questionFromDb =
_db.Question.FirstOrDefault(x => x.ConversationId ==
convercetion.Id && x.Text == question.Text);
                            foreach (var answerOptions in
question.AnswerOptions)
                            {
                                await
_professorController.CreateAnswerOptions(professor.Login,
questionFromDb.Id, answerOptions.Text);
                            }
                        }
                    }
                }
            }
            else return BadRequest();
        return Ok();
    }

    [HttpGet("GetConversationXmlByIdentyficator")]
    public string GetConversationXmlByIdentyficator(string
identyficator)
    {
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
        var questions = _db.Question.Where(x => x.ConversationId ==
conversations.Id).ToList();
        foreach (var question in questions)
        {
            var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
            question.AnswerOptions = answerOptions;
        }
        conversations.Question = questions;
        using(var stringwriter = new System.IO.StringWriter())
    {
        var serializer = new
XmlSerializer(conversations.GetType());
        serializer.Serialize(stringwriter, conversations);
            string xmlUtf8 = stringwriter.ToString().Replace("utf-
16", "utf-8");
            return xmlUtf8;
        }
```

```
    }

    [Authorize(Roles = "professor,student")]
    [HttpPost("GetConversationByIdentyficator")]
    public Conversations? GetConversationByIdentyficator(string
identyficator)
    {
        var student = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
        if (conversations == null)
        {
            return null;
        }
        var passedConversationss = _db.PassedConversations.Where(x
=> x.ConversationId == conversations.Id && x.StudentId ==
student.Id).ToList();
        if (passedConversationss != null)
        {
            foreach (var passedConversation in
passedConversationss)
            {
                if (passedConversation.Attempt >=
conversations.HowManyAttempt)
                {
                    return null;
                }
            }
        }
        var questions = _db.Question.Where(x => x.ConversationId ==
conversations.Id).ToList();
        foreach (var question in questions)
        {
            var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
            question.AnswerOptions = answerOptions;
        }
        conversations.Question = questions;
        conversations.PassedConversations =
_db.PassedConversations.Where(x => x.ConversationId ==
conversations.Id && x.StudentId == student.Id).ToList();
        return conversations;
    }

    [Authorize(Roles = "professor,student")]
    [HttpPost("GetPassedConversationsByUser")]
    public List<PassedConversations> GetPassedConversationsByUser()
    {
        var people = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
```

```
        var passedConversations = _db.PassedConversations.Where(x
=> x.StudentId == people.Id).ToList();
        return passedConversations;
    }

    [Authorize(Roles = "professor")]
    [HttpPost("GetPassedConversationsForProfessor")]
    public List<People> GetPassedConversationsForProfessor()
    {
        List<PassedConversations> passedConversations = new
List<PassedConversations>();

        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var conversations = _db.Conversations.Where(x =>
x.ProfessorId == professor.Id).ToList();
        List<People> students = new List<People>();
        foreach (var conversation in conversations)
        {

passedConversations.AddRange(_db.PassedConversations.Where(x =>
x.ConversationId == conversation.Id));
        }
        foreach (var passed in passedConversations)
        {
            if (students.FirstOrDefault(x => x.Id ==
passed.StudentId) == null)
            {
                students.Add(_db.People.FirstOrDefault(x => x.Id ==
passed.StudentId));
            }
        }
        foreach (var passed in passedConversations)
        {
            foreach (var student in students)
            {
                if (passed.StudentId == student.Id)
                {
                    student.PassedConversations.Add(passed);
                }
            }
        }
        return students;
    }

    [Authorize(Roles = "professor")]
    [HttpPost("GetHistoryPassedConversationsForProfessor")]
    public List<People> GetHistoryPassedConversationsForProfessor()
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
```

```
        var conversations = _db.Conversations.Where(x =>
x.ProfessorId == professor.Id).ToList();
        List<People> students = new List<People>();
        var passedConversations = _db.PassedConversations.Where(x
=> x.ProfessorId == professor.Id).ToList();
        foreach (var passed in passedConversations)
        {
            if (students.FirstOrDefault(x => x.Id ==
passed.StudentId) == null)
            {
                students.Add(_db.People.FirstOrDefault(x => x.Id ==
passed.StudentId));
            }
        }
        foreach (var passed in passedConversations)
        {
            foreach (var student in students)
            {
                if (passed.StudentId == student.Id)
                {
                    student.PassedConversations.Add(passed);
                }
            }
        }
        return students;
    }
}
```

Клас ProfessorController

```
[ApiController]
[Route("[controller]")]
public class ProfessorController : Controller
{
    private readonly SignalRClient _signalRClient;

    private readonly DBContext _db;
    public ProfessorController(SignalRClient signalRClient,
DBContext db)
    {
        _signalRClient = signalRClient;
        _db = db;
    }

    [Authorize(Roles = "professor")]
    [HttpPost("CreateConversation")]
    public async Task<IActionResult> CreateConversation(string
name, int? howManyAttempt, int markRange)
    {
```

```
        var rndGen = new RandomGen();
        Conversations conversations;
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        string identyficator;
        do
        {
            identyficator = rndGen.RandomString(8);
            conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
        } while (conversations != null);

        var conversation = new Conversations
        {
            Name = name,
            ProfessorId = professor.Id,
            ProfessorName = $"{professor.FirstName}
{professor.LastName}",
            HowManyAttempt = howManyAttempt,
            MarkRange = markRange,
            DateOfCreation = DateTime.Now.ToString("MM/dd/yyyy
HH:mm"),
            Identyficator = identyficator
        };
        _db.Conversations.Add(conversation);
        _db.SaveChanges();
        var conversationFromDb = _db.Conversations.FirstOrDefault(x
=> x.Name == name && x.HowManyAttempt == howManyAttempt &&
x.MarkRange == markRange);
        await _signalRClient.connection.SendAsync("Conversations",
"add", conversationFromDb);
        return Ok();
    }
    public async Task<IActionResult> CreateConversation(string user
, string name, int? howManyAttempt, int markRange)
    {
        var rndGen = new RandomGen();
        Conversations conversations;
        var professor = _db.People.FirstOrDefault(x => x.Login ==
user);
        string identyficator;
        do
        {
            identyficator = rndGen.RandomString(8);
            conversations = _db.Conversations.FirstOrDefault(x =>
x.Identyficator == identyficator);
        } while (conversations != null);

        var conversation = new Conversations
        {
            Name = name,
            ProfessorId = professor.Id,
```

```
            ProfessorName = $"{professor.FirstName}
{professor.LastName}",
            HowManyAttempt = howManyAttempt,
            MarkRange = markRange,
            DateOfCreation = DateTime.Now.ToString("MM/dd/yyyy
HH:mm"),
            Identyficator = identyficator
        };
        _db.Conversations.Add(conversation);
        _db.SaveChanges();
        var conversationFromDb = _db.Conversations.FirstOrDefault(x
=> x.Name == name && x.HowManyAttempt == howManyAttempt &&
x.MarkRange == markRange);
        await _signalRClient.connection.SendAsync("Conversations",
"add", conversationFromDb);
        return Ok();
    }

    [Authorize(Roles = "professor")]
    [HttpPost("DeleteConversation")]
    public async Task<IActionResult> DeleteConversation(int id)
    {
        var user = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.Id == id && x.ProfessorId == user.Id);
        if (conversations == null) return BadRequest();
        _db.Conversations.Remove(conversations);
        await _signalRClient.connection.SendAsync("Conversations",
"delete", conversations);
        _db.SaveChanges();
        return Ok();
    }

    [Authorize(Roles = "professor")]
    [HttpPost("CreateQuestion")]
    public async Task<IActionResult> CreateQuestion(int
conversationId, string text)
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.ProfessorId == professor.Id && x.Id == conversationId);
        var questions = _db.Question.FirstOrDefault(x =>
x.ConversationId == conversations.Id && x.Text == text);
        if (conversations == null || questions != null) return
BadRequest("Access is denied");
        var question = new Question
        {
            ConversationId = conversationId,
            Text = text
        };
```

```
        _db.Question.Add(question);
        _db.SaveChanges();
        var optionFromDb = _db.Question.FirstOrDefault(x => x.Text
== text && x.ConversationId == conversationId);
        await _signalRClient.connection.SendAsync("Question",
"add", optionFromDb);
        return Ok();
    }
    public async Task<IActionResult> CreateQuestion(string user,
int conversationId, string text, int trueAnswerId)
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
user);
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.ProfessorId == professor.Id && x.Id == conversationId);
        var questions = _db.Question.FirstOrDefault(x =>
x.ConversationId == conversations.Id && x.Text == text);
        if (conversations == null || questions != null) return
BadRequest("Access is denied");
        var question = new Question
        {
            ConversationId = conversationId,
            Text = text,
            TrueAnswerId = trueAnswerId
        };
        _db.Question.Add(question);
        _db.SaveChanges();
        var optionFromDb = _db.Question.FirstOrDefault(x => x.Text
== text && x.ConversationId == conversationId);
        await _signalRClient.connection.SendAsync("Question",
"add", optionFromDb);
        return Ok();
    }

    [Authorize(Roles = "professor")]
    [HttpPost("DeleteQuestion")]
    public async Task<IActionResult> DeleteQuestion(int id)
    {

        var user = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var questions = _db.Question.FirstOrDefault(x => x.Id ==
id);
        if (questions == null) return BadRequest();
        _db.Question.Remove(questions);
        await _signalRClient.connection.SendAsync("Question",
"delete", questions);
        _db.SaveChanges();
        return Ok();
    }

    [Authorize(Roles = "professor")]
```

```
    [HttpPost("CreateAnswerOptions")]
    public async Task<IActionResult> CreateAnswerOptions(int
questionId, string text)
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        //var conversations = db.Conversations.FirstOrDefault(x =>
x.ProfessorId == professor.Id);
        var answersOptions = _db.Question.FirstOrDefault(x => x.Id
== questionId);
        var answersOption = _db.AnswerOptions.FirstOrDefault(x =>
x.Id == questionId && x.Text == text);

        if (answersOptions == null || answersOption != null) return
BadRequest();
        var answerOptions = new AnswerOptions
        {
            QuestionId = questionId,
            Text = text
        };
        _db.AnswerOptions.Add(answerOptions);
        _db.SaveChanges();
        var answerOptionsFromDb =
_db.AnswerOptions.FirstOrDefault(x => x.Text == text &&
x.QuestionId == questionId);
        await _signalRClient.connection.SendAsync("AnswerOptions",
"add", answerOptionsFromDb);
        return Ok("ok");
    }
    public async Task<IActionResult> CreateAnswerOptions( string
user, int questionId, string text)
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
user);
        //var conversations = db.Conversations.FirstOrDefault(x =>
x.ProfessorId == professor.Id);
        var answersOptions = _db.Question.FirstOrDefault(x => x.Id
== questionId);
        var answersOption = _db.AnswerOptions.FirstOrDefault(x =>
x.Id == questionId && x.Text == text);

        if (answersOptions == null || answersOption != null) return
BadRequest();
        var answerOptions = new AnswerOptions
        {
            QuestionId = questionId,
            Text = text
        };
        _db.AnswerOptions.Add(answerOptions);
        _db.SaveChanges();
```

```
        var answerOptionsFromDb =
_db.AnswerOptions.FirstOrDefault(x => x.Text == text &&
x.QuestionId == questionId);
        await _signalRClient.connection.SendAsync("AnswerOptions",
"add", answerOptionsFromDb);
        return Ok("ok");
    }

    [Authorize(Roles = "professor")]
    [HttpPost("DeleteAnswerOptions")]
    public async Task<IActionResult> DeleteAnswerOptions(int id)
    {
        var user = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var answerOptions = _db.AnswerOptions.FirstOrDefault(x =>
x.Id == id);
        if (answerOptions == null) return BadRequest();
        _db.AnswerOptions.Remove(answerOptions);

        await _signalRClient.connection.SendAsync("AnswerOptions",
"delete", answerOptions);
        _db.SaveChanges();
        return Ok();
    }

    [Authorize(Roles = "professor")]
    [HttpPost("GetConversationByProfessorId")]
    public List<Conversations> GetConversationsByProfessorId()
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        if (professor == null) return null;
        var conversations = _db.Conversations.Where(x =>
x.ProfessorId == professor.Id).ToList();
        if (conversations == null)
        {
            return null;
        }
        List<Question> questions;
        foreach (var conversation in conversations)
        {
            questions = _db.Question.Where(x => x.ConversationId ==
conversation.Id).ToList();
            foreach (var question in questions)
            {
                var answerOptions = _db.AnswerOptions.Where(x =>
x.QuestionId == question.Id).ToList();
                question.AnswerOptions = answerOptions;
            }
            conversation.Question = questions;
        }
        return conversations;
```

```
    }

    [Authorize(Roles = "professor")]
    [HttpPost("UpdateConversation")]
    public async Task<IActionResult> UpdateConversation(int id,
string name)
    {
        var rndGen = new RandomGen();
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var conversation = _db.Conversations.FirstOrDefault(x =>
x.Id == id && x.ProfessorId == professor.Id);
        conversation.Name = name;
        await _signalRClient.connection.SendAsync("Conversations",
"update", conversation);
        _db.Conversations.Add(conversation);
        return Ok();
    }

    [Authorize(Roles = "professor")]
    [HttpPost("UpdateQuestion")]
    public async Task<IActionResult> UpdateQuestion(int id, int
trueAnswerId, string text)
    {
        var professor = _db.People.FirstOrDefault(x => x.Login ==
User.Identity.Name);
        var question = _db.Question.FirstOrDefault(x => x.Id ==
id);
        var conversations = _db.Conversations.FirstOrDefault(x =>
x.Id == question.ConversationId);
        if (conversations == null) return BadRequest("Access is
denied");
        question.Text = text;
        question.TrueAnswerId = trueAnswerId;
        _db.SaveChanges();
        var optionFromDb = _db.Question.FirstOrDefault(x => x.Id ==
id);
        await _signalRClient.connection.SendAsync("Question",
"update", optionFromDb);
        return Ok();
    }

    [Authorize(Roles = "professor")]
    [HttpPost("UpdateAnswerOptions")]
    public async Task<IActionResult> UpdateAnswerOptions(int id,
string text)
    {
        var answerOptions = _db.AnswerOptions.FirstOrDefault(x =>
x.Id == id);
        answerOptions.Text = text;
        _db.SaveChanges();
```

```
        var answerOptionsFromDb =
_db.AnswerOptions.FirstOrDefault(x => x.Id == id);
        await _signalRClient.connection.SendAsync("AnswerOptions",
"update", answerOptionsFromDb);
        return Ok();
    }
}
```

ДОДАТОК Г
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний Університет
Факультет програмування та комп'ютерних
і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

Дипломний проект, на тему:

«Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів»

Студент: Дзюрбан Едуард Станіславович

Керівник: Радельчук Г. І. кандидат технічних наук, доцент

ВСТУП

Світові тенденції у оцінюванні знань учнів та студентів все більше схиляються до об'єктивнішого методу – тестування. Оскільки при такому виді контролю знань виключений вплив суб'єктивних чинників, таких як обізнаність екзаменатора про поточну успішність студента чи облік його поведінки на учбових заняттях. Також важливим фактором є те, що результат достовірний і надійний – тестова оцінка є однозначною, та добре відображає об'єм та рівень засвоєння матеріалу.

Враховуючи епідеміологічну ситуації в Україні та світі, навчальні заклади масово переходять на дистанційну форму навчання, а отже і контроль знань відбувається дистанційно. Викладачі змушені шукати різні способи комунікації із учнями, та, що найголовніше – способи контролю та оцінки їх знань. Адже тепер немає можливості зібрати їх усіх у одній аудиторії, та написати тест на листі паперу.

АКТУАЛЬНІСТЬ ТЕМИ

Підсумовуючи сказане вище, можна зробити висновок, що подана тема на даний момент є як ніколи актуальною, а її вирішення - затребуваним. З відкритих засобів, які є доволі популярними у викладачів, та дозволяють так чи інакше створити щось на кшталт тестів для перевірки знань, немає таких, які були б максимально зручними та ефективними.

МЕТА ТА ЗАВДАННЯ ДИПЛОМНОГО ПРОЕКТУВАННЯ

Метою проекту є створення програмної системи, яка б дозволяла швидко та зручно створювати нові тести, або ж імпортувати їх з інших форматів; доступ до якої був би швидким та легким, а інтерфейс інтуїтивно зрозумілим та простим.

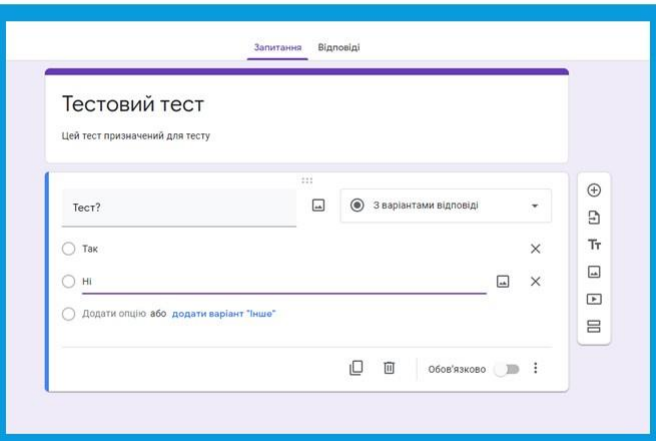
Основними завданнями при виконанні дипломного проекту є:

- визначення та аналіз особливостей предметної області, її суті та основних проблем;
- аналіз наявних на ринку програмних засобів та платформ відповідно предметної області;
- проектування та реалізація програмної системи, яка б відповідала основним вимогам та вирішувала основні проблеми;
- тестування реалізованої системи.

НАЯВНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Найпопулярнішими та найчастіше використовуваними для тестування учнів чи студентів є Google Форми. Це веб-застосунок (тому доступ до нього легкий і швидкий, варто лише авторизуватися через аккаунт Google), частина пакету інструментарію Google Drive.

Має досить приємний дизайн, зручний інтерфейс. Має великі можливості для додавання медіафайлів (таких як фото, відео, посилань, тощо) до запитань. Також має зручний інструмент систематизації результатів та виведення статистики. Основним недоліком є те, що ця система погано підходить для тестування студентів, а більше для опитування, оскільки тут відсутній функціонал перевірки відповідей та визначення результату (оцінки).

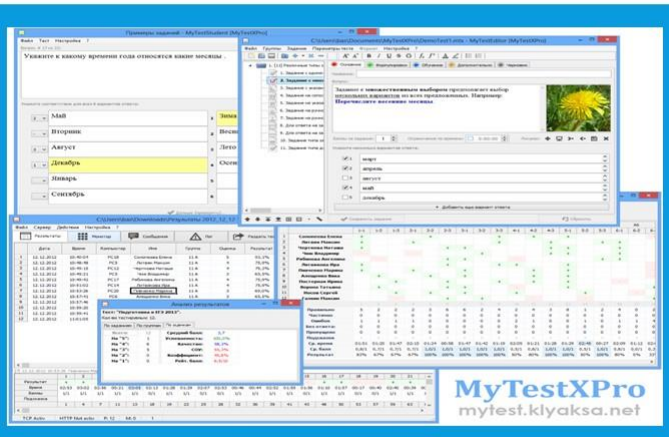


Інтерфейс Google Forms

НАЯВНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Наступна по популярності програма – MyTestX. Являє собою десктопний застосунок. Має досить широкий функціонал, глибокі можливості по виведенню статистичних даних та їх аналізу. Є досить потужним інструментом, що є і плюсом і мінусом водночас. Для викладачів не завжди потрібен такий великий набір інформації, глибокого статистичного аналізу, тощо. Це все дуже нагромаджує інтерфейс програми і робить його важким для сприйняття новими користувачами.

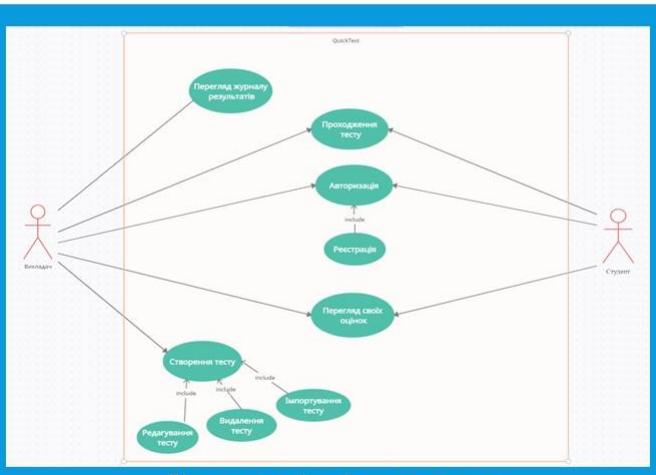
Та, мабуть, найбільшим мінусом цього комплексу є те, що це все ж десктопний додаток, для користування яким потрібна його інсталяція, та ще й на кожному комп'ютері, з якого буде проводитися тестування.



Інтерфейс MyTestX

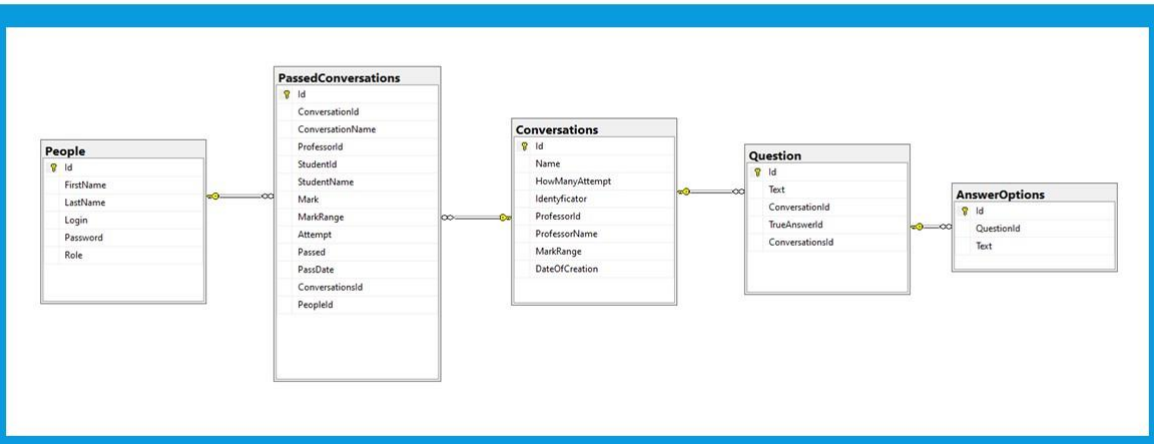
ПРОЕКТУВАННЯ

Щоб представити функціональне призначення системи було побудовано діаграму варіантів використання. Вона складається з користувачів системи (акторів) та варіантами використання (дій) які вони можуть робити.



Діаграма варіантів використання

ПРОЕКТУВАННЯ. БАЗА ДАНИХ

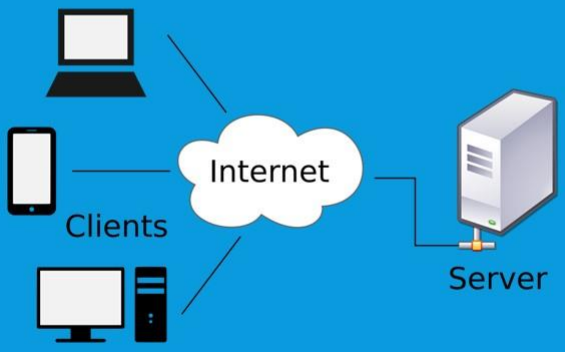


Логічна модель бази даних

ПРОЕКТУВАННЯ. ВИБІР АРХІТЕКТУРИ

Концепція десктопних застосунків при створенні систем збору та обробки даних (а система для тестування такою і являється) на даний момент є частково застарілою. Натомість, в тренді зараз програмні системи побудовані на архітектурі клієнт-сервер.

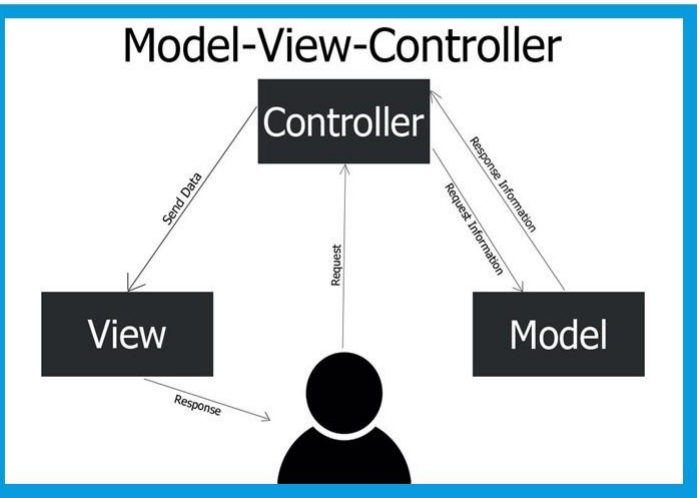
Найкращим вибором при такому підході проектування буде модель «тонкого» клієнта, коли всі обчислення та алгоритми обробки даних відбуваються на сервері (що значно підвищує безпеку та надійність системи), а користувачу для доступу до системи буде достатньо мати лиш доступ до мережі та веб-браузер



ПРОЕКТУВАННЯ. ВИБІР ПАТЕРНУ

Після аналізу найпопулярніших патернів проектування ПЗ, було вирішено, що найкраще для реалізації заявленої програмної системи підійде шаблон MVC. Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування.

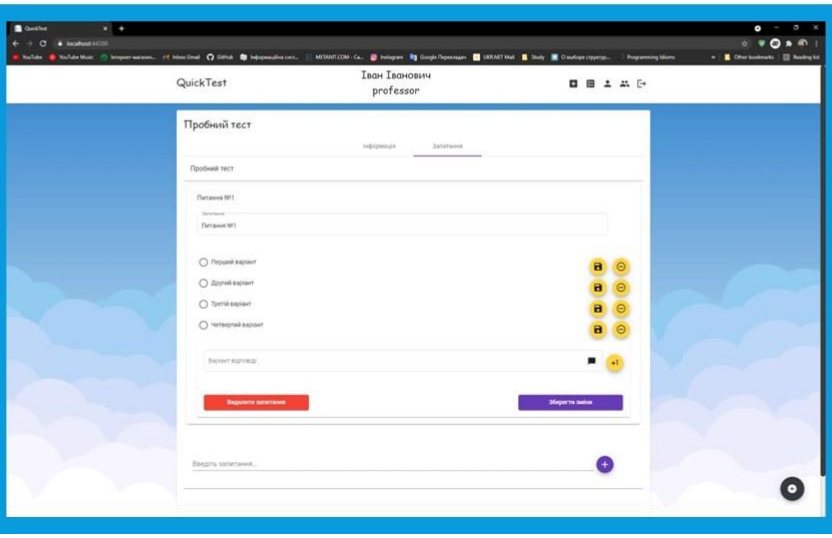
Його основна перевага - це відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.



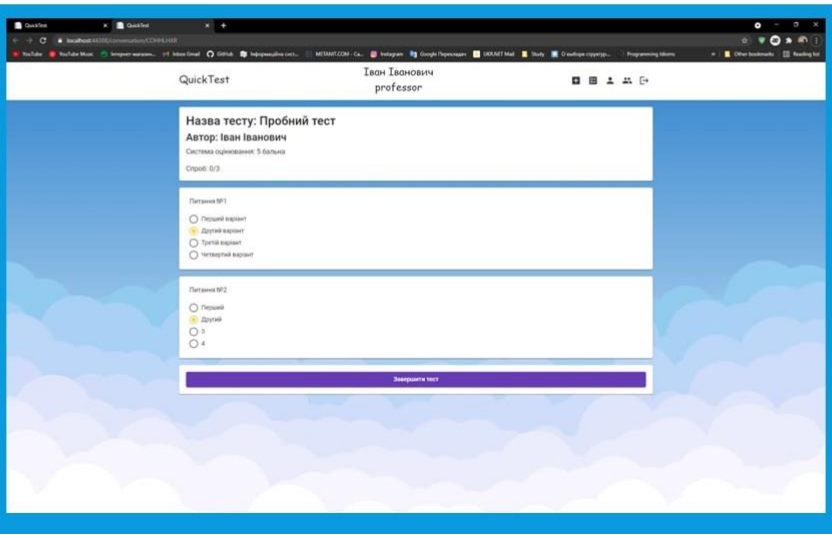
ЗАСОБИ РЕАЛІЗАЦІЇ

- Реалізація програмної системи на стороні сервера написана мовою C# на основі платформи ASP.NET Core. Ця платформа ідеально підходить для вибраного патерну створення застосунку MVC.
- Інтерфейс користувача реалізовано за допомогою фреймворку Angular та бібліотеки компонентів Material Angular.
- Розроблена програмна система з використанням сучасних рішень, таких як: Entity Framework, SignalR, ihostedservice тощо.

ФОРМА РЕДАГУВАННЯ ТЕСТУ



ФОРМА ПРОХОДЖЕННЯ ТЕСТУ



ВИСНОВКИ

У процесі виконання дипломного проекту було проведено аналіз предметної області за заданою темою, та встановлено, що в нинішніх реаліях такий вид контролю знань, як тестування, є найбільш об'єктивним, відносно швидким та досить зручним. А з використанням програмної системи для автоматизації цього процесу він стає ще легшим і швидшим.

Проведено аналіз існуючого інформаційного та програмного забезпечення предметної області, у результаті якого визначено функціональні вимоги до створюваної програмної системи.

Було вибрано архітектури і патерни проектування програмного продукту. Також визначені засоби та сучасні рішення для реалізації спроектованої програмної системи.

В результаті виконання дипломного проекту було створено програмне забезпечення, яке вирішує основні проблеми предметної області та відповідає поставленим функціональним вимогам.

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Дзюрбана Е. С.
Прізвище, ініціали

факультет ПКТС, 4 курс, група ІПЗ-17-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

15.06.2021р
дата

Дзюрбан
підпис

Tue Jun 15 11:38:22 EEST 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 10.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

ID: 93956 Назва: Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів Додано в БД: 2021-06-15 Автора: Е. С. Дзюрбан Керівники: Г. І. Радельчук Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	67852	656	10283 (15%)	114 (17%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1008301743

Дата перевірки:
15.06.2021 12:30:10 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
15.06.2021 12:31:35 EEST

ID користувача:
100005589

Назва документа: **Дипломний проект Дзюрбан Е. С. ІПЗ-17-1**

Кількість сторінок: **71** Кількість слів: **11573** Кількість символів: **93137** Розмір файлу: **3.11 MB** ID файлу: **1008369940**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

12.3%
Схожість

Найбільша схожість: **6.07%** з джерелом з Бібліотеки (ID файлу: **1008272511**)

5.65% Джерела з Інтернету	378	Сторінка 73
7.76% Джерела з Бібліотеки	87	Сторінка 75

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	3
Підозріле форматування	16 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Дзюрбан Едуард Станіславович

Тема Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки_____

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті проведено аналіз предметної області, наявного схожого програмного забезпечення, виділено їх плюси та мінуси. Виконано порівняння різних архітектурних підходів до створення програмних систем та вибір найоптимальнішого для вирішення поставлених задач. Спроектовано серверну та клієнтську частину веб додатку та, відповідно до спроектованої моделі виконана реалізації програмного продукту. Також проведено тестування програмної системи.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект освітнього ступеня «бакалавр» в основному відповідає поставленому завданню як в теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі описано актуальність дипломної роботи. У першому розділі проводиться аналіз предметної області, досліджуються її проблеми та варіанти їх вирішення, описуються вимоги до програмного продукту. У другому розділі аналізуються архітектурні підходи до розробки програмного забезпечення, обґрунтовується вибір архітектури додатку та шаблонів проектування, проектується користувацька та серверна частини додатку, проводиться аналіз способів реалізації та вибір технологій для розробки. Третій розділ описує реалізацію програмної системи. Останній розділ присвячений процесу тестування програмного забезпечення і опис результатів роботи модулів.

4. Позитивні сторони проекту Тема дипломного проекту є актуальною, оскільки в даний час питання дистанційного оцінювання якості знань як ніколи потребує вирішення. До позитивних сторін реалізації програмної системи можна віднести те, що при розробці системи було використано низку сучасних технологій.

5. Негативні сторони проекту До негативних сторін проекту можна віднести відсутність варіативності тестів та відсутність можливості обмеження часу на одну спробу проходження. Також відсутня функція підтвердження електронної пошти та відновлення паролю.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до теми дипломного проекту. Графічне представлення виконано на задовільному рівні. Пояснювальна записка відповідає стандартам оформлення.

7. Відгук про дипломний проект в цілому В цілому дипломний проект заслуговує задовільної оцінки. Матеріал пояснювальної записки структурований та розписаний на достатньому рівні для розуміння викладеного матеріалу по заданій темі дипломного проекту. Усі графічні матеріали підкріплені інформацією, що дозволяє наочно побачити та зрозуміти результати виконання дипломного проекту.

8. Інші зауваження

9. Оцінка дипломного проекту Розглянувши позитивні та негативні сторони проекту можна зробити висновок що даний дипломний проект заслуговує оцінки «задовільно»

РЕЦЕНЗЕНТ Мартинюк Валерій Володимирович, доктор технічних наук, професор, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій і телекомунікацій (АКІТ І ТК)

"09" червня 2021 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Програмна система тестування студентів з можливістю імпорту тестових завдань з інших форматів»
Автор: Дзюрбан Едуард Станіславович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Радельчук Галина Іванівна, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломного проекту системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо та в назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 12,3% і адресується до 378 джерел з Інтернет та 87 джерел з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ Г. І. Радельчук

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк