Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня
бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Левдик Ірини Андріївни*
(ПІБ)
академічної групи *121-17-1*
(шифр)
спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)
освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)
на тему: *Платформа для генерації та проходження учбових тестів онлайн. Розробка серверної частини*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Приходченко С.Д.*			
розділів:				
спеціальний	*доц. Приходченко С.Д.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-17-1* *Левдик Ірини Андріївни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Платформа для генерації та проходження учбових тестів онлайн. Розробка серверної частини*

затверджена наказом ректора НТУ «ДП» від . №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Левдик І.А*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 70 с., 36 рис., 7 табл., 3 дод., 20 джерел.

Об'єкт розробки: контрольно-діагностична платформа для генерації та проходження тестів.

Мета кваліфікаційної роботи: розробка контрольно-діагностичної платформи для генерації та проходження тестів.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці контрольно-діагностичної платформи для генерації та проходження онлайн-тестувань, що дозволяє оптимізувати навчальний процес.

Актуальність програмного продукту визначається наявністю великого відсотку впливу людського фактору на навчальний процес та процедуру оцінювання і контролю знань студентів.

Список ключових слів: ПРОГРАМА, КОНТРОЛЬНО-ДІАГНОСТИЧНА ПЛАТФОРМА, ПРОТОКОЛ, КІНЦЕВА ТОЧКА, БРАУЗЕР, КЛІЄНТ, СЕРВЕР, ІНФОРМАЦІЙНА СИСТЕМА, ЕОМ, ПРИСТРІЙ, ТОКЕН ДОСТУПУ, ТОКЕН ОНОВЛЕННЯ, JSON ВЕБ ТОКЕН.

ABSTRACT

Explanatory note: 70 p., 36 figs., 7 tabl., 3 appx., 20 sources.

Object of development: control and diagnostic platform for generating and passing tests.

Purpose of the qualification work: development of a control and diagnostic platform for generating and passing tests.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development is carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

The practical significance lies in the development of a control and diagnostic platform for generating and passing online tests, which allows to optimize the learning process.

The relevance of the software product is determined by the presence of the great percentage of the influence of the human factor on the educational process and the procedure of assessment and control of students' knowledge

Keywords: PROGRAM, CHECK-DIAGNOSTIC PLATFORM, PROTOCOL, END POINT, BROWSER, CLIENT, SERVER, INFORMATION SYSTEM, COMPUTER, DEVICE, ACCESS-TOKEN, REFRESH-TOKEN, JSOB WEB TOKEN.

ЗМІСТ

ВСТУП

На сьогоднішній день все більше набирає обертів автоматизація різноманітних робочих процесів. Починаючи із супермаркетів, шкіл та закінчуючи величезними промисловими заводами. Автоматизація — це завжди про поліпшення, прогрес, економію. Економію сил, ресурсів, заощадження коштів тощо. Не оминає увагою процес автоматизації й освітню сферу. Завдяки технічним досягненням та автоматизації багатьох процесів все легше стає отримувати інформацію, обробляти її, зберігати, засвоювати, запам'ятовувати, аналізувати. Одним із прикладів полегшення навчального процесу є автоматизація процедури оцінювання знань. Яскравим прикладом успішного її застосування у цій сфері є реалізація проведення ЗНО. Результати тестів перевіряються комп'ютерами, що значно заощаджує людський ресурс, який перерозподіляється у більш важливі процеси (перевірка відкритих завдань, задач, творів).

Можна навіть не заглиблюватися у настільки глобальну сферу, а взяти дещо менше, більш локальне. Наприклад, онлайн-тести, які можна проходити за власним бажанням, самостійно складати, оцінювати, аналізувати тощо. Вони дуже спрощують життя своїм користувачам (викладачам, студентам, учням, вчителям і так далі).

Досить популярною річчю, на наш час, є самоосвіта, тож у переважної більшості людей, які прагнуть розвитку, виникає потреба перевірити свої знання, оцінити себе. Звичайно, можна оформити самоперевірку звичайним способом: скласти тест, пройти його, а потім самостійно, прискіпливо вдивляючись, перевіряти кожну відповідь, сподіваючись лише на свою уважність. Але людський фактор — досить непередбачувана річ і зараз світила прогресу намагаються зменшити його вплив. До того ж, самостійний аналіз власних результатів займає час та сили, які можна було б приділити важливішим речам.

Отже, автоматизація оцінювання є лише безперечним плюсом в даному випадку.

З огляду на вищевказану інформацію мною була обрана тема дипломного проекту. Темою дипломного проекту є «Платформа для генерації та проходження учбових тестів онлайн. Розробка серверної частини».

Завдання даної кваліфікаційної роботи та об'єкт її діяльності безпосередньо пов'язані з напрямом підготовки «Інженерія програмного забезпечення» та відповідають узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 121 «Інженерія програмного забезпечення» галузі знань 12 «Інформаційні технології». Виконання кваліфікаційної роботи надає можливості отримання автору кваліфікації «фахівець з розробки та тестування програмного забезпечення».

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної області

Об’єктом дослідження та аналізу предметної області є контрольно-діагностична платформа для створення та проведення автоматизованого онлайн-тестування. Метою проведення аналізу предметної області є виявлення, класифікація, формалізація інформації про усі аспекти предметної області, що впливають на характеристики кінцевого програмного продукту.

Були впроваджені наступні роботи:

- Сбір даних про об’єкт автоматизації та здійснюваних ним видів діяльності
- Вивчення та оцінка якості функціонування об’єкта та здійснюваних ним видів діяльності, виявлення проблем, рішення яких можливе завдяки засобам автоматизації
- Оцінка технічно-економічної, соціальної, практичної доцільності створення інформаційної системи
- Виявлення та аналіз вимог користувача до інформаційної системи
- Аналіз ринку та потенційних конкурентоспроможних вже існуючих рішень
- Оцінка ефективності прийнятих проектних рішень для створення інформаційної системи

Для збору та обробки інформації про аналогічні проектні рішення були застосовані наступні методи:

- Опитування
- Обсервація
- Аналіз документів
- Порівняльний аналіз
- Прогнозування

– Експеримент

В ході аналізу мною було виявлено, що переважна більшість існуючих рішень надають необхідний фукнціонал для створення та проведення опитувань, однак вони є мультизадачними та не спрямовані конкретно на навчальний процес, поліпшення та полегшення його проведення, його оптимізацію, адже надають поверхневий результат, не надаючи статистики для аналізу результатів для подальшої зручності їх використання. До того ж переважна більшість платформ не зберігають детальних результатів тестування, що також заважає проведенню аналізу та поліпшенню навчальних процесів.

Перейдемо до огляду вже існуючих рішень та результатів дослідження цих систем.

Online Test Pad

Платформа надає можливість для створення як простих тестів, так і опитувань, кросвордів, діалогів – тестів, оформлених у вигляді спілкування з віртуальним екзаменатором.

Переваги платформи:

– Багато типів опитувань, інтуїтивно зрозумілий інтерфейс

Недоліки платформи:

– Застарілий дизайн тестів
– Відсутній попередній перегляд тесту
– Погана адаптація під різні девайси
– Багато реклами та монетизації

Let’s test

Сервіс спеціалізується виключно на онлайн-тестах. Конструктор містить всього шість типів питань, проте інтерфейс не є user-friendly, важкий для опанування.

Переваги платформи:

– Ізольована система тестування

Недоліки платформи:

- Застарілий дизайн конструктора
- Не інтуїтивний інтерфейс
- Відсутня можливість самостійно налаштувати дизайн тесту
- Відсутній попередній перегляд тесту
- Багато монетизації, невеликий пробний період
- Завищені тарифи

Anketolog

Сервіс спеціалізується, по першу чергу, на онлайн-опитуваннях, проте також допомагає створювати тести.

Переваги платформи:

- Налаштування дизайну анкетування за допомогою CSS, дозволяє широкий вибір у зборі результатів опитування, тести можуть перевірятися найманими фахівцями

Недоліки платформи:

- Невелика кількість типів питань саме у тестах (4), що зменшує зручність використання та універсальність платформи

Конструктор Тестов.ру

Простий конструктор, який надає можливість створення тесту, проходження тестів, що знаходяться у вільному доступі.

Переваги платформи:

- Великий обсяг вже існуючих тестів, абсолютно відсутня монетизація

Недоліки платформи:

- Застарілий дизайн
- Багато реклами
- Відсутня статистика за відповідями
- Незручна реалізація роботи у ролях викладач-учень: учнів повинен зареєструвати вчитель

– Є більш розважальним порталом, ніж професійним інструментом

ClassMarker

Англомовний сервіс, що дозволяє створювати тести, анкети, передивлятися статистику.

Переваги платформи:

– Приємний та зручний дизайн, дозволяє проведення закритих тестувань, перегляд статистики

Недоліки платформи:

– Англомовний інтерфейс
– Дуже високі тарифи
– Відсутня можливість вивантажувати результати тесту

Testix

Англомовний конструктор тестів з можливістю перемикання на російську мову.

Переваги платформи:

– Наявні готові шаблони, приємний та зручний дизайн

Недоліки платформи:

– Відсутня можливість перегляду результатів тестування

Qzzr

Англомовний конструктор тестів, дозволяє перекладати тест на інші мови, забезпечує інтеграцію з CRM-системою, платформами автоматизації маркетингу.

Переваги платформи:

– Можливість адаптувати тест під дизайн сайту

Недоліки платформи:

– Англійська мова
– Відсутня можливість користування пробною версією без прив’язки банківської карти

– Заточений під маркетингові системи для залучення нових клієнтів

Переважна більшість платформ та конструкторів для створення опитувань спрямовані на узагальнений аналіз відповідей, більшість з них не забезпечують користувачів зберіганням їхніх результатів та наданням статистики; також певні системи були заточені під специфічні галузі, які не є близькими до освітньої; тільки в одній платформі були виявлені ролі викладач/учень, проте їхня реалізація не є досконалою та зручною для користувачів; велика кількість систем надавали свій функціонал платно, здебільшого сучасні та зручні платформи, що також є мінусом; були виявлені огріхи в UI- та UX-дизайні, які гальмують зручність використання платформи та підвищують поріг входження для її використання, на це впливає і англомовний інтерфейс, що був виявлений у чверті з наявних програмних продуктів; у великої кількості систем була виявлена відсутність функціоналу попереднього перегляду створених тестів, що лише підвищує вплив людського фактору та знижує ефективність наданої системою автоматизації.

Отже, підсумовуючи все вищезазначене, можна зробити висновок про унікальність та актуальність розроблюваного програмного продукту, а також про потенційну його затребуваність.

1.2. Призначення розробки та галузь застосування

Інформаційна система, що виконана для кваліфікаційної роботи, має назву «Платформа для генерації та проходження учбових тестів онлайн. Серверна частина».

Програма призначена для оптимізації та автоматизації навчального процесу, застосування в профільних підрозділах на об'єктах замовника.

Функціональним призначенням програми є надання користувачеві у ролі вчителя можливості створення тестів, проведення тестування серед групи користувачів у ролі учнів, збору результатів тестування, перегляду результатів

тестування та подальшої їхньої обробки; користувачеві у ролі учня — можливості проходження тестування онлайн, перегляду результатів своїх попередніх тестувань, отримання нотифікацій про тестування, його результати.

Програма повинна експлуатуватися в профільних підрозділах на об'єктах замовника. Користувачами програми повинні бути співробітники профільних підрозділів об'єктів замовника.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 275-с від 26.05 2021 р;

– завдання на кваліфікаційну роботу на тему «Платформа для генерації та проходження учбових тестів онлайн. Розробка серверної частини».

1.4. Постановка завдання

Темою дипломного проекту є «Платформа для генерації та проходження учбових тестів онлайн. Розробка серверної частини».

Розробити серверну частину програмної контрольно-діагностичної платформи для створення онлайн тестів та їхнього проходження в рамках навчального процесу. Програмна серверна частина передбачає:

– створення акаунта по ролях (викладач/учень)

- керування акаунтом (реєстрація, вхід, вихід, верифікація електронної пошти, редагування інформації про акаунт користувачем, перегляд інформації користувачем)
- забезпечення захисту акаунта від зловмисників (реалізація надійного механізму аутентифікації , розробка оптимально безпечного способу збереження credentials: вірчих грамот)
- створення тесту в ролі викладача
- групування тестів завдяки створенню курсу в ролі викладача
- групування учнів завдяки створенню групи в ролі викладача та маніпуляції з групою в ролі викладача (редагування групи, видалення групи, прикріплення групи учнів до курсу)
- перегляд та оцінювання результатів відкритих питань тесту, пройдених учнями, у ролі викладача
- автоматичний підрахунок балів за тестові питання тесту
- забезпечення проходження тесту за певний часовий проміжок
- створення питань чотирьох типів у ролі вчителя: одна відповідь, декілька відповідей, відкрите питання, ручне введення, питання з прикріпленням відповіді файлом
- проходження тесту в ролі учня
- перегляд свого результату проходження тесту в ролі учня
- перегляд пройдених та запланованих тестів в ролі учня
- забезпечення видачі фільтрованих/шуканих даних (фільтрація тестів за датою, темою, пройденими/не пройденими; пошук тесту за темою)
- формування метрик статистики за результатами тестування
- формування деталей тесту та їхній перегляд користувачем у ролі викладача
- нотифікацію користувачів у ролі учня про заплановане тестування, яке їм потрібно пройти

– нотифікацію користувачів у ролі викладача про тести, які потрібно перевірити

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт має дотримуватися наступних функціональних вимог:

– програма повинна забезпечувати користувачів у ролі викладача можливістю створювати тест
– програмний продукт повинен забезпечувати користувачів у ролі учня можливістю проходити тест, на який учня підписує викладач
– програмний продукт повинен забезпечувати користувачів можливістю створювання акаунту, його редагування, перегляду інформації
– система повинна контролювати доступ користувачів до інформації залежно від їхніх ролей та креденшелів
– система має зберігати результати проходження тесту кожним користувачем у ролі учня та надавати змогу передивлятися хронологію пройдених тестувань
– система має зберігати тести, створені користувачем у ролі викладача та надавати змогу передивлятися їхні деталі
– платформа повинна надавати ряд статистичних метрик для будування діаграм на клієнтській частині

1.5.2.Вимоги до інформаційної безпеки

Для забезпечення надійного функціонування системи необхідно реалізувати наступні вимоги:

– валідація введених даних: перевірка на відповідність типів, на введення обов’язкових полів даних, валідація діапазонів;

– відмовостійкість (при виникненні помилок програма не повинна припиняти своєї роботи та повідомлювати про помилки клієнтській частині);

– захист від несанкціонованого доступу до даних, наданих серверною частиною клієнтській частині платформи;

– верифікація пошти користувача під час реєстрації;

– хешування паролів за допомогою надійного алгоритму.

1.5.3. Вимоги до складу та параметрів технічних засобів

Відштовхуючись від того, що серверна частина додатку працюватиме з потенційно великими обсягами інформації та локальною базою даних, виконуватиме певні обчислення, вибірку даних, звернення до сторонніх ресурсів, були визначені наступні вимоги до конфігурації ПК:

– SSD-накопичувач: 512 ГБ

– оперативна пам’ять [RAM]: 16 ГБ

– ЦП [CPU]: Intel Core i7–2600 3.40GHz

– Мережевий інтерфейс: Gigabit Ethernet 1ГБіт

1.5.4. Вимоги до інформаційної та програмної сумісності

Інформаційна система потребує від користувача мати середовище з такими складовими:

– одна з операційних систем: Windows, починаючи з 8-ої, Linux, Mac OS, Android, IOS;

– постійний доступ до мережі Інтернет.

Серверна частина програмної платформи матиме мову програмування C# з використанням кросплатформеного фреймворку ASP.NET Core, фреймворку Entity Framework та інших додаткових NuGet-пакетів.

Для розробки програмного продукта необхідна наявність наступних програм та систем:

- Visual Studio 2019 Community/JetBrains Rider
- MS SQL Server
- GitHubDesktop
- gitbash

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути програма, сумісна з будь-яким клієнтом та працююча на будь-якій платформі, що є серверною частиною для контрольно-тестової платформи генерації тестів, адаптованою під користувачів, що бажають проводити онлайн-тестування та проходити їх, мають намір оптимізувати навчальний процес.

Експлуатаційне призначення – автоматизація та спрощення ручної праці людини під час проходження тестів, проведення тестувань, перевірки результатів тестувань; оптимізація навчального процесу. Розроблена система дозволяє звести використання паперу, що застосовується під час проведення очних тестів, до мінімуму, значно зменшує витрати часу та людських ресурсів для аналізу отриманих даних, їх упорядкування, збереження, а також сортування, редагування, пошуку та додавання.

Функціональне призначення – надати можливість користувачам у ролі вчителя генерувати власні тести, надавати доступ до цих тестів іншим користувачам у ролі учня, перевіряти результати тестувань; надати можливість користувачам у ролі учня проходити тести, на які вони підписані, передивлятися свої навчальні досягнення.

2.2. Опис використаної архітектури та шаблонів проектування

Під час розробки програми була реалізована N-рівнева архітектура(або багатошарова архітектура, вона ж Onion-архітектура).

Коли у програмному додатку є простий логічний поділ, він називається багатошаровим додатком або N-шаровим додатком. У випадках, коли існує як

фізичний, так і логічний поділ завдань, його часто називають n-рівневим додатком, де n - кількість поділів. 3 - найбільш поширене значення .

Таке розбиття допомагає в поділі проблем, розділяючи рішення на більш дрібні одиниці, щоб кожна одиниця відповідала за конкретну задачу, а також допомагає використати переваги абстракції. Для проектів середнього і великого масштабу, де працюють кілька команд, багатошаровість має очевидні переваги. Це дозволяє конкретній команді або окремій робочій одиниці працювати на певному рівні, не порушуючи цілісності інших.

Нижче можна побачити просте схематичне уявлення варіанту N-рівневої архітектури.

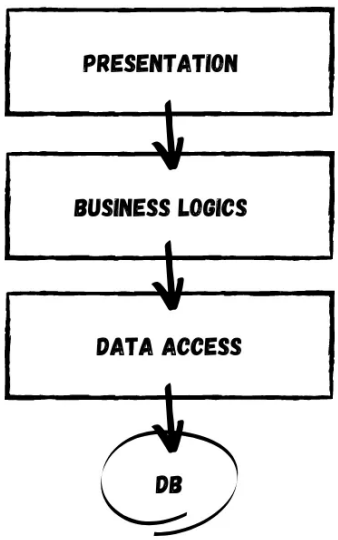

Рис. 2.1. N-рівнева архітектура

Вона містить три рівні:

– рівень представлення - це найнижчий рівень, на якому програмісти прикладних програм розглядають структуру даних та презентацію, а не просто передають дані у вигляді дейтаграм або пакетів між хостами;

– рівень бізнес-логіки - інкапсулює всю бізнес-логіку, всі необхідні обчислення, отримує об'єкти з шару доступу до даних і передає їх на рівень представлення, або, навпаки, отримує дані з рівня представлення і передає їх на рівень даних;
– рівень доступу до даних – рівень, що надає спрощений доступ до даних, що зберігаються в постійному сховищі будь-якого типу, такому як реляційна база даних, наприклад.

Також під час розробки програми були реалізовані наступні шаблони проектування:

– UnitOfWork

Патерн визначає логічну транзакцію тобто атомарну синхронізацію змін в об'єктах, поміщених в об'єкт UoW зі сховищем (базою даних).

Якщо звернутися до вихідного опису цього патерну у Мартіна Фаулера - то видно що об'єкт, який реалізує цей патерн відповідає за накопичення інформації про те, які об'єкти входять до транзакції і чим вони відмінні від початкових значень в сховищі.

Патерн Unit of Work як правило не є повністю самостійним, він зазвичай тісно пов'язаний із патерном Identity Map, завдання якого - збереження карти створених об'єктів, взятих зі сховища для того, щоб гарантувати, що одна одиниця інформації зі сховища представлена рівно одним екземпляром об'єкта даних в додатку (контекстом). Це дозволяє уникнути конфліктів змін тому не допускає ситуації коли два об'єкти, що представляють один і той же елемент даних в сховищі, змінені по-різному.

Оскільки для обчислення різниці (і, відповідно, визначення того що і яким чином має бути змінено в сховище) необхідно знати які дані і як саме зберігаються в об'єктах - як правило необхідна також реалізація патерну Metadata Mapping, що описує зв'язок між вмістом сховища (наприклад таблицями і стовпцями бази даних) і класами / властивостями об'єктів.

Також, якщо дані в сховище не є незалежними (наприклад зв'язку між таблицями в базі даних) - може знадобитися реалізації ряду патернів, що відповідають за збереження інформації про зв'язки між даними.

– Репозиторій

Патерн-посередник між рівнями області визначення (сховище) і розподілу даних. Використовує інтерфейс, схожий на колекції, для доступу до об'єктів області визначення. Репозиторій інкапсулює набір об'єктів, що зберігаються в сховищі даних, і операції, що виконуються над ними, забезпечуючи більш об'єктно-орієнтоване уявлення реальних даних. Репозиторій також має на меті досягнення повного поділу і односторонньої залежності між рівнями області визначення і розподілу даних.

– Інтерфейс

Основний шаблон проектування, що є загальним методом для структурування проекту з метою легшого подальшого його використання та доповнення. Загалом, інтерфейс - це клас, який забезпечує програмісту простий або більш програмно-специфічний спосіб доступу до інших класів.

Інтерфейс може містити набір об'єктів і забезпечувати просту, високорівневу функціональність для програміста; він може забезпечувати більш чистий або більш специфічний спосіб використання складних класів; він може використовуватися в якості «клею» між двома різними API; і для багатьох інших цілей.

– Анотація (інтерфейс-маркер)

Шаблон проектування, застосовуваний в мовах програмування з перевіркою типів під час виконання. Шаблон надає можливість зв'язати метадані (інтерфейс) з класом навіть при відсутності в мові явної підтримки для метаданих.

Щоб використовувати цю модель, клас реалізує інтерфейс, а методи, що взаємодіють із класом, перевіряють наявність інтерфейсу. На відміну від

звичайного інтерфейсу, який визначає функціональність (у вигляді оголошень методів і властивостей), якою повинен володіти реалізований клас об'єктів, важливий сам факт володіння класу маркером. Маркер лише є ознакою наявності певної поведінки у об'єктів класу, позначеного маркером.

Також система була спроектована, дотримуючись 5 принципів SOLID:

- Принцип єдиної відповідальності (single responsibility principle) - для кожного класу має бути визначено єдине призначення. Всі ресурси, необхідні для його здійснення, повинні бути вміщені в цей клас і підпорядковані тільки цьому завданню;
- Принцип відкритості / закритості (open-closed principle) - програмні сутності повинні бути відкриті для розширення, але закриті для модифікації;
- Принцип підстановки Лісков (Liskov substitution principle) - об'єкти в програмі повинні бути замінними на екземпляри їх підтипів без зміни правильності виконання програми;
- Принцип поділу інтерфейсу (interface segregation principle) - багато інтерфейсів, спеціально призначених для клієнтів, краще, ніж один інтерфейс загального призначення;
- Принцип інверсії залежностей (dependency inversion principle) - залежність на абстракцію. Немає залежності на щось конкретне.

Система реалізована згідно методології REST, що вимагає наявності наступних пунктів:

1. Модель клієнт-сервер

Зазначена вимога виконується, адже платформа розбита на дві програмні складові: серверну та клієнтську частини. Програмний додаток є серверною частиною цієї платформи.

2. Відсутність стану

Протокол взаємодії між клієнтом і сервером вимагає дотримання наступного умови: в період між запитами клієнта ніяка інформація про стан

клієнта на сервері не зберігається (Stateless protocol або «протокол без збереження стану»).

У системі спілкування між клієнтом та сервером відбувається за допомогою stateless HTTP-протоколу.

3. Єдиний інтерфейс

Наявність уніфікованого інтерфейсу є фундаментальною вимогою для використання методології REST. Уніфіковані інтерфейси дозволяють кожному із сервісів розвиватися незалежно. До уніфікованих інтерфейсів висувається 4 основні вимоги, яких дотримується даний проект:

– Ідентифікація ресурсів

Всі ресурси ідентифікуються в запитах, наприклад, з використанням URI в інтернет-системах. Ресурси концептуально відокремлені від уявлень, які повертаються клієнтам. Наприклад, сервер може надсилати дані з бази даних у вигляді HTML, XML або JSON, жоден з яких не є типом зберігання всередині сервера.

Вимога дотримується, усі ресурси є проідентифікованими згідно конвенціям з неймінгу, кожен ендпоінт є унікальним та вертає лише дані, інформацію.

– Маніпуляція ресурсами через представлення

Якщо клієнт зберігає уявлення ресурсу, включаючи метадані - він володіє достатньою інформацією для модифікації або видалення ресурсу.

Представлення в REST використовується для виконання дій над ресурсами. Представлення ресурсу являє собою поточний або бажаний стан ресурсу. Наприклад, якщо ресурсом є користувач, то представлень може бути XML або HTML опис цього користувача.

Вимога виконується, адже за допомогою представлень у вигляді JSON-об’єктів є можливість змінювати ресурси.

– Самоописувані повідомлення

Запит і відповідь повинні зберігати в собі всю необхідну інформацію для їх обробки.

Вимога виконується, система спроектована максимально змістовно.

– Гіпермедіа як засіб зміни стану програми

Статус ресурсу передається через вміст body, параметри рядка запиту, заголовки запитів і запитуваний URI (ім'я ресурсу). Це називається гіпермедіа (або гіперпосилання з гіпертекстом).

Система підтримує зміну стану ресурсів за допомогою гіпермедіа.

4. Шари

Клієнт зазвичай не здатний точно визначити, взаємодіє він безпосередньо з сервером або ж з проміжним вузлом, в зв'язку з ієрархічною структурою мереж (маючи на увазі, що така структура утворює шари).

Система багатошарова (побудована згідно з N-layer архітектурою).

5. Код на вимогу (опційно)

Дозволяється завантаження і виконання коду на стороні клієнта.

Сервери можуть тимчасово розширювати або кастомізувати функціонал клієнта, передаючи йому логіку, яку він може виконувати. Наприклад, це можуть бути скомпільовані Java-аплети або клієнтські скрипти на Javascript.

Поточна версія проекту не потребує завантаження та виконання коду на стороні клієнта.

6. Кешування (опційне)

Як і у Всесвітній павутині, клієнти, а також проміжні вузли, можуть виконувати кешування відповідей сервера. Відповіді сервера, в свою чергу, повинні мати явне або неявне позначення про те, можна чи ні її кешувати, з метою запобігання отримання клієнтами застарілих або невірних даних у відповідь на наступні запити. Правильне використання кешування здатне частково або повністю усунути деякі клієнт-серверні взаємодії, ще більше підвищуючи продуктивність і масштабованість системи.

Поточна версія проекту не підтримує функціонал, пов’язаний із кешуванням відповідей.

2.3. Опис використаних технологій та мов програмування

Серверна частина розроблюваної платформи написана мовою C#. з використанням фреймворку ASP.NET Core, вона є WebAPI-службою. В якості бази даних використовується MS SQL (Local DB), також використаний Entity Framework для спілкування між серверною програмною частиною та базою даних. Також присутні різноманітні NuGet-пакети.

В якості IDE обрана Visual Studio 2019 Community, веб-серверу – IIS. Система управління версій – git, робота з яким виконувалася завдяки додатку GitHubDesktop.

C # (вимовляється як "сі Шарп") - сучасна об'єктно-орієнтована і типово безпечна мова програмування. C # дозволяє розробникам створювати безліч типів безпечних і надійних програм, які працюють в екосистемі .NET. C # відноситься до широко відомої родини мов C.

C # - це об'єктно-і компонентно-орієнтована мова програмування. C # надає мовні конструкції для безпосередньої підтримки такої концепції роботи. Завдяки цьому C # підходить для створення і застосування програмних компонентів. З моменту створення мова C # збагатилася функціями для підтримки нових робочих навантажень і сучасними рекомендаціями по розробці ПЗ.

Ось лише кілька функцій мови C #, які дозволяють створювати надійні та стійкі додатки.

- Прибирання сміття - автоматично звільняє пам'ять, зайняту недоступними невикористовуваними об'єктами.
- Типи, що допускають значення null, забезпечують захист від змінних, які не посилаються на виділені об'єкти.

- Обробка винятків надає структурований і розширюваний підхід до виявлення помилок і відновлення після них.
- Лямбда-вирази підтримують прийоми функціонального програмування. Синтаксис LINQ створює загальний шаблон для роботи з даними з будь-якого джерела.
- Підтримка мов для асинхронних операцій надає синтаксис для створення розподілених систем.
- У C # діє _ * єдина система типів **. Всі типи C #, включаючи типи-примітиви, такі як int і double, успадковують від одного кореневого типу object. Всі типи використовують загальний набір операцій, а значення будь-якого типу можна зберігати, передавати і обробляти схожим чином. Більш того, C # підтримує як визначаються користувачами посилальні типи, так і типи значень. C # дозволяє динамічно виділяти об'єкти та зберігати спрощені структури в стеці.
- C # підтримує універсальні методи і типи, що забезпечують підвищену безпеку типів і продуктивність.
- C # надає ітератори, які дозволяють розробникам класів колекцій визначати призначені для користувача варіанти поведінки для клієнтського коду.

У C # особлива увага приділяється управлінню версіями для забезпечення сумісності програм і бібліотек при їх зміні. Питання управління версіями істотно вплинули на такі аспекти розробки C #, як роздільні модифікатори virtual і override, правила вирішення перевантаження методів і підтримка явного оголошення членів інтерфейсу.

Програми C # виконуються в .NET, віртуальній системі виконання, що викликає загальномовне середовище виконання (CLR) і набір бібліотек класів. Середа CLR - це реалізація загальномовної інфраструктури мови (CLI), що є міжнародним стандартом, від корпорації Майкрософт. CLI є основою для

створення середовищ виконання і розробки, в яких мови і бібліотеки прозоро працюють один з одним.

Вихідний код, написаний на мові C # компілюється в проміжну мову (IL), яка відповідає специфікаціям CLI. Код на мові IL і ресурси, в тому числі растрові зображення і рядки, зберігаються в збірці, зазвичай з розширенням .dll. Збірка містить маніфест з інформацією про типи, версії, мовою і регіональних параметрах для цієї збірки.

При виконанні програми C # збірка завантажується в середу CLR. Середа CLR виконує JIT-компіляцію з коду на мову IL в інструкції машинної мови. Середа CLR також виконує інші операції, наприклад, автоматичну збірку сміття, обробку винятків і управління ресурсами. Код, що виконується середовищем CLR, іноді називають "керованим кодом", щоб підкреслити відмінності цього підходу від "некерованого коду", який відразу компілюється в машинний мову для певної платформи.

Забезпечення взаємодії між мовами є ключовою особливістю .NET.

Код IL, створений компілятором C #, відповідає специфікації загальних типів (CTS).

Код IL, створений з коду на C #, може взаємодіяти з кодом, створеним з версій .NET для мов F #, Visual Basic, C ++ і будь-яких інших з більш ніж 20 мов, сумісних з CTS.

Одна збірка може містити кілька модулів, написаних на різних мовах .NET, і всі типи можуть посилатися один на одного, як якщо б вони були написані на одній мові.

На додаток до runtime-служб .NET також включає розширені бібліотеки. Ці бібліотеки підтримують безліч різних робочих навантажень. Вони впорядковані по просторах імен, які надають різні корисні можливості: від операцій файлового введення і виведення до управління рядками і синтаксичного аналізу XML, від платформ веб-додатків до елементів управління Windows Forms. Зазвичай

додаток C # активно використовують бібліотеку класів .NET для вирішення типових задач.

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. А в листопаді 2020 року побачила версія ASP.NET Core 5.0, яка власне і буде охоплена в поточному керівництві.

ASP.NET Core тепер повністю є opensource-фреймворком. Всі вихідні файли фреймворку доступні на GitHub.

ASP.NET Core може працювати поверх крос-платформного середовища .NET Core, яке може бути розгорнуте на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core можна створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі превалює, але тепер вже немає обмеження тільки цією операційною системою. Тобто можна веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатків можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої

функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле.

Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій.

Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис html з кодом C #.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

В якості інструментарію розробки можна використовувати останні випуски Visual Studio, починаючи з версії Visual Studio 2015. Крім того, можна створювати додатки в середовищі Visual Studio Code, яка є крос-платформної і може працювати як на Windows, так і на Mac OS X і Linux.

Для обробки запитів тепер використовується новий конвеєр HTTP, який заснований на компонентах Katana і специфікації OWIN. А його модульність дозволяє легко додати свої власні компоненти.

Якщо підсумувати, то можна виділити наступні ключові відмінності та переваги ASP.NET Core:

- Новий легкий і модульний конвеєр HTTP-запитів
- Можливість розгортати додаток як на IIS, так і в рамках свого власного процесу
- Використання платформи .NET Core і її функціональності
- Поширення пакетів платформи через NuGet

- Інтегрована підтримка для створення та використання пакетів NuGet
- Єдиний стек веб-розробки, що поєднує Web UI і Web API
- Конфігурація для спрощеного використання в хмарі
- Вбудована підтримка для впровадження залежностей
- Можливість розширення
- Кросплатформеність: можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux
- Розвиток як open source, відкритість до змін

Ці та інші особливості і можливості стали основою для нової моделі програмування.

Visual Studio це лінійка програмних компонентів, призначена для полегшення написання коду різними мовами. Продукт дозволяє розробляти консольні і графічні додатки.

На сьогоднішній день ринок програмного забезпечення в основному представлений продуктами корпорації Майкрософт, яка свої розробки адаптує під операційну систему Віндоус.

У Visual Studio ці параметри універсальні, оскільки кожна версія суттєво відрізняється, як в функціоналі так і в ціні.

Git (вимовляється «гіт») - розподілена система керування версіями. Проект був створений Лінус Торвальдсом для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року. На сьогоднішній день його підтримує Джун Хама.

Система спроектована як набір програм, спеціально розроблених з урахуванням їх використання в сценаріях. Це дозволяє зручно створювати спеціалізовані системи контролю версій на базі Git або призначені для користувача інтерфейси. Наприклад, Cogito є саме таким прикладом оболонки до репозиторіїв Git, а StGit використовує Git для управління колекцією виправлень (патчів).

Git підтримує швидкий поділ і злиття версій, включає інструменти для візуалізації та навігації по нелінійный історії розробки. Як і Darcs, BitKeeper, Mercurial, Bazaar і Monotone, Git надає кожному розробнику локальну копію всієї історії розробки, зміни копіюються з одного сховища в інший.

Віддалений доступ до репозиторіїв Git забезпечується git-демоном, SSH- або HTTP-сервером. TCP-сервіс git-daemon входить в дистрибутив Git і є поряд з SSH найбільш поширеним і надійним методом доступу. Метод доступу по HTTP, незважаючи на ряд обмежень, дуже популярний в контрольованих мережах, тому що дозволяє використовувати існуючі конфігурації мережевих фільтрів.

IIS (Internet Information Services, до версії 5.1 - Internet Information Server) - пропріетарний набір серверів для декількох служб Інтернету від компанії Microsoft. IIS поширюється з Windows NT.

Основним компонентом IIS є веб-сервер, який дозволяє розміщувати в Інтернеті сайти. IIS підтримує протоколи HTTP, HTTPS, FTP, POP3, SMTP, NNTP.

Основним компонентом IIS є веб-сервер - служба WWW (звана також W3SVC), яка надає клієнтам доступ до сайтів за протоколами HTTP і, якщо проведена настройка, HTTPS.

Один сервер IIS може обслуговувати кілька сайтів (IIS 6.0 і вище). Кожен сайт має наступні атрибути:

- IP-адреса сайту;
- TCP-порт, на якому служба WWW чекає підключень до даного сайту;
- Тема вузла (Host header name) - значення заголовка Host запиту HTTP, яке вказує зазвичай DNS-ім'я сайту.

Таким чином, наприклад, один сервер з одним IP-адресою може обслуговувати на одному TCP-порту кілька сайтів. Для цього необхідно створити кілька DNS-записів, що вказують на IP-адресу сервера, і розрізняти сайти по заголовкам вузла.

Для кожного сайту вказується домашній каталог - каталог файлової системи сервера, відповідний «корені» сайту. Наприклад, якщо сайту www.example.com зіставлений домашній каталог D: \ example, то на запит ресурсу з адресою

Веб-сервер IIS підтримує кілька різних технологій створення веб-додатків:

- ASP.NET - розроблена Microsoft технологія; для IIS - це основний на сьогоднішній день засіб створення веб-додатків і веб-служб. IIS 6.0 поставляється разом з операційними системами, в які також спочатку входить .NET Framework, так що підтримка ASP.NET начебто вже вбудована в IIS 6.0; для більш ранніх версій необхідно окремо завантажити і встановити .NET Framework.
- ASP - передувала ASP.NET технології створення динамічних веб-сторінок на основі сценаріїв. Входить в поставку IIS починаючи з версії 3.0.
- CGI - стандартна межплатформенна низькорівнева технологія створення динамічних веб-сторінок.
- FastCGI - клієнт-серверний протокол взаємодії веб-сервера і додатки.
- ISAPI - низькорівнева технологія, аналогічна інтерфейсу модулів Apache, що надає повний доступ до всіх можливостей IIS, можливість розробки веб-додатків в машинному коді і можливість перевизначення частини функцій IIS і додавання до нього функцій, як пов'язаних з генерацією контента, так і не пов'язаних з цим. Підсистема виконання скриптів ASP і підсистема ASP.NET виконані як модулі ISAPI.
- SSI - включення в одні сторінки тексту з інших сторінок. Строго кажучи, веб-додатком не є, оскільки IIS підтримує лише обмежений набір можливостей і без того малофункціонального SSI. Зокрема, IIS5 підтримує тільки статичну включення і ігнорує команди умовного розгалуження.

IIS підтримує роботу SMTP / POP3-сервісів. У сучасних версіях Microsoft Exchange Server реалізація протоколів SMTP, POP3 та IMAP виконана у вигляді підсистем до IIS, які вигідно відрізняються поставляються з IIS поштові підсистеми.

2.4. Опис структури системи та алгоритмів її функціонування

В основі структури системи лежить архітектура клієнт-сервер, у якій обмін даними між клієнтською та серверною частинами здійснюється завдяки HTTP(s) протоколу.

Нижче можна побачити приблизну схему взаємодії користувача з системою.

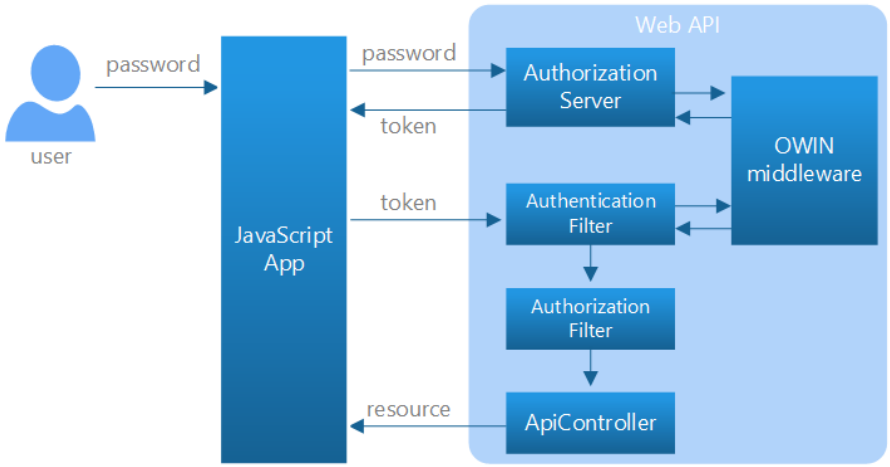

Рис. 2.2. «Схема взаємодії користувача з системою»

За допомогою клієнтського додатку, користувач вводить свої логін та пароль для авторизації в системі.

Клієнтський додаток надсилає їх до серверної частини, яка валідує їх та, після успішної аутентифікації та авторизації надсилає на клієнтську частину токен.

Було прийняте рішення реалізувати механізм аутентифікації за допомогою JWT (JSON Web токенів). Отже, сервер надсилає пару токен доступу та токен оновлення.

Після авторизації користувач має доступ для авторизованих ендпоїнтів та ендпоїнтів, досяжних його ролі.

Для валідації користувача за роллю використовуються фільтри, що складають проміжний рівень. Кожний запит також надсилається разом із заголовком Authentication, який містить токен доступу, що був виданий у момент входу до системи.

Запит, пройшовши фільтри, надсилається до API-контролеру за вказаним маршрутом, який згодом повертає відгук, що містить код статусу відгуку та дані (опційно) у форматі JSON.

Спроектована база даних містить 10 пов’язаних між собою таблиць у першій її версії. Нижче можна побачити фізичну модель створюваної бази даних.

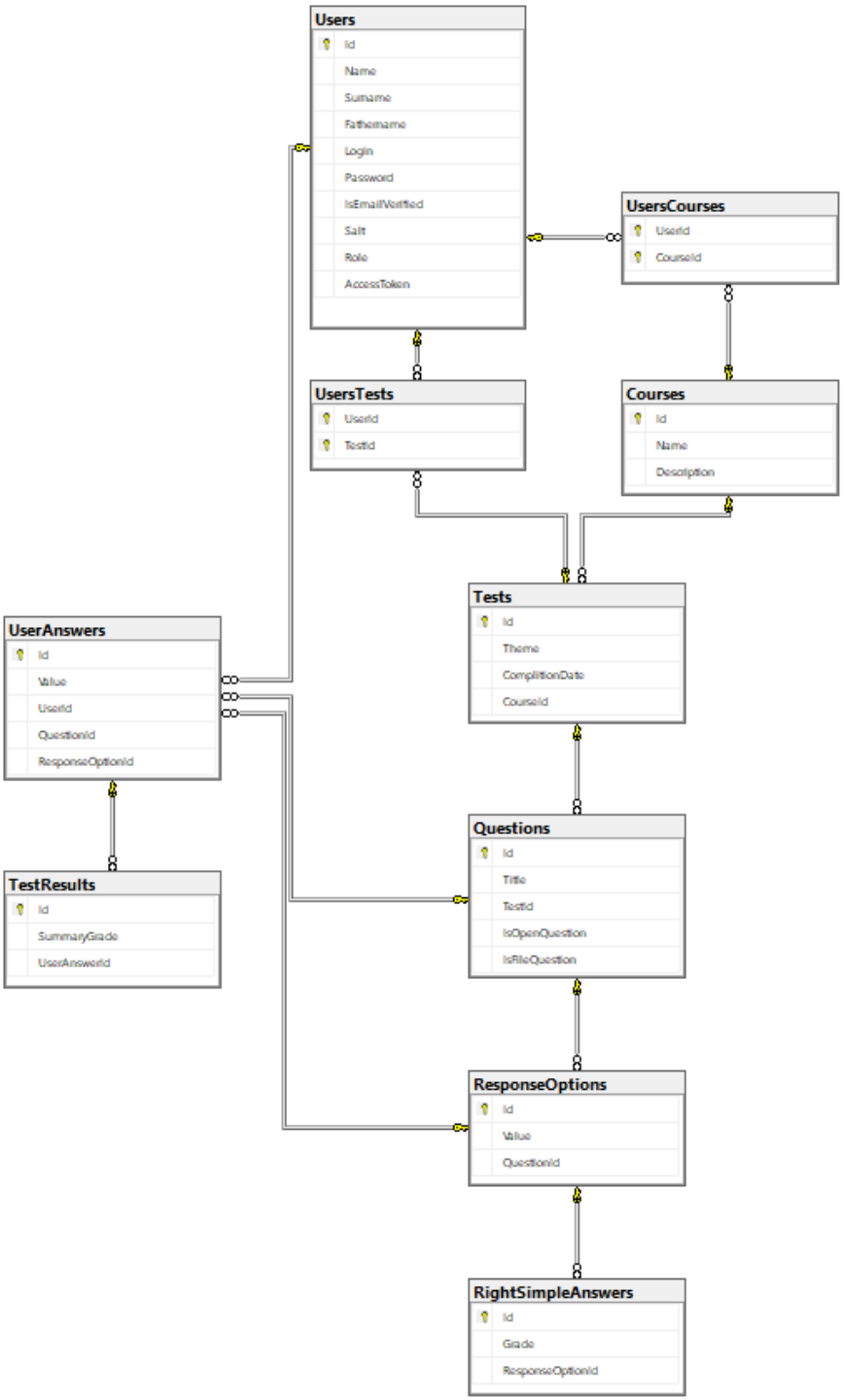

Рис.2.3. «Фізична модель бази даних»

Розглянемо таблиці у вищезазначеній базі даних.

Таблиця 2.1

Користувач

Назва поля	Призначення	Тип даних	Ключ
Id	ID користувача	int	PK
Name	Ім'я	Varchar(50)	
Surname	Прізвище	Varchar(50)	
Fathername	По-батькові	Varchar(50)	
Login	Логін (електронна пошта)	Varchar(50)	
Password	Захешований пароль	Varchar(50)	
IsEmailVerified	Флаг верифікації електронної пошти	Bit	
Salt	Сіль для хешування паролю	Varchar(50)	
Role	Роль	int	
AccessToken	Токен доступу	Varchar(50)	

Таблиця 2.2

Курс

Назва поля	Призначення	Тип даних	Ключ
Id	ID курсу	int	PK
Name	Назва	Varchar(50)	
Description	Детальний опис курсу	Varchar(50)	

Таблиця 2.3

Тест

Назва поля	Призначення	Тип даних	Ключ
Id	ID теста	int	PK
Theme	Тема теста	Varchar(50)	
ComplitionDate	Дата проходження теста	Date (UTC)	
PassTime	Час проходження теста (у хвилинах)	int	
CourseId	ID курсу, до якого прикріплений тест	int	FK

Таблиця 2.4

Питання

Назва поля	Призначення	Тип даних	Ключ
Id	ID питання	int	PK
Title	Заголовок	Varchar(50)	
TestId	ID теста, якому належить питання	int	FK
IsOpenQuestion	Флаг відкритого питання	Bit	
IsFileQuestion	Флаг файлового питання	Bit	

Таблиця 2.5

Варіант відповіді

Назва поля	Призначення	Тип даних	Ключ
Id	ID варіанту відповіді	int	PK
Title	Заголовок	Varchar(50)	

Value	Зміст варіанту відповіді	Varchar(50)	
QuestionId	ID питання, за яким закріплений варіант відповіді	int	FK

Таблиця 2.6

Правильні відповіді

Назва поля	Призначення	Тип даних	Ключ
Id	ID правильної відповіді	int	PK
Grade	Оцінка за обраний варіант відповіді	Decimal	
ResponseOptionId	ID варіанта відповіді, яка є вірною	int	FK

Таблиця 2.7

Відповідь користувача

Назва поля	Призначення	Тип даних	Ключ
Id	ID відповіді користувача	int	PK
UserId	ID користувача	int	FK
QuestionId	ID питання, якому надав відповідь користувач	int	FK
ResponseOptionId	ID варіанта відповіді, обраної користувачем	int	FK
Value	Шлях до файлу, якщо користувач надав	Varchar(500)	

	файлову відповідь на питання, або відкрита відповідь на питання		

Також в базі наявні сполучні таблиці для забезпечення зв’язку багато до багатьох між сутностями.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані програми надсилаються до ендпоїнтів. Ендпоїнти повертають статус обробки запиту та дані (опційно).

Серверна частина побудована у форматі WebAPI та має ряд наступних ендпоїнтів.

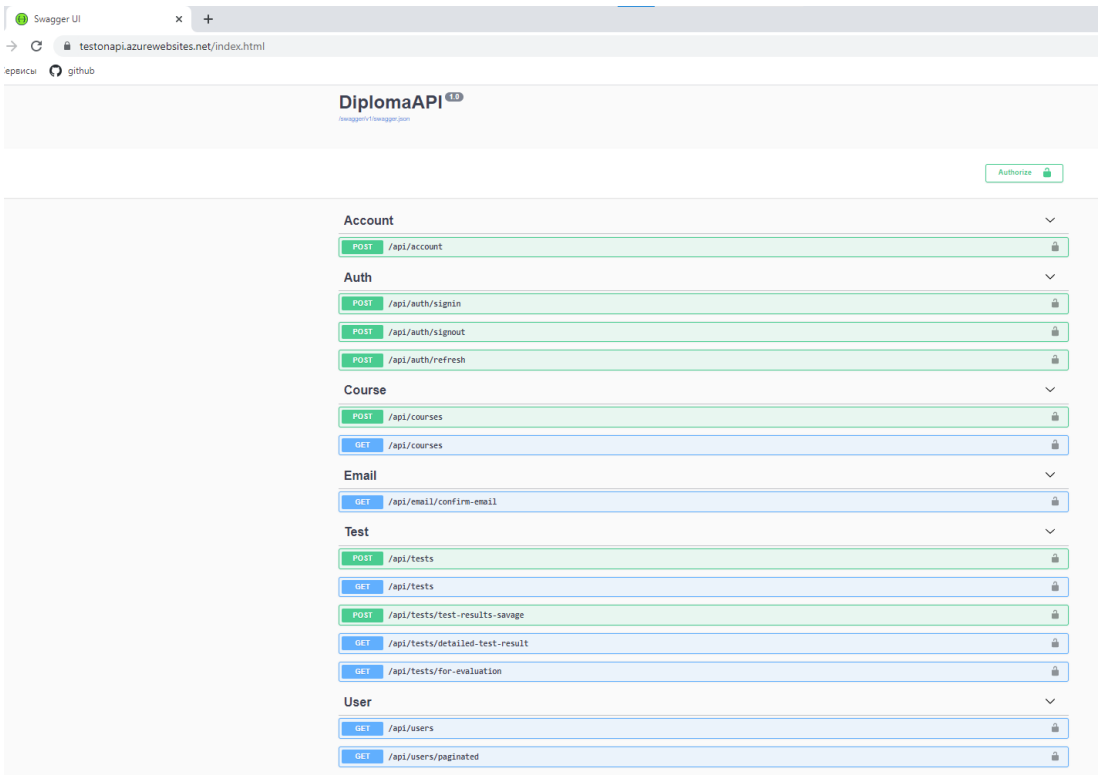

Рис. 2.4. «Самодокументовані за допомогою Swagger ендпоїнти серверної частини системи»

Ендпоїнти в системі:

1. Account

– /api/account (POST) – створення акаунту

Вимагає наступну вхідну модель:

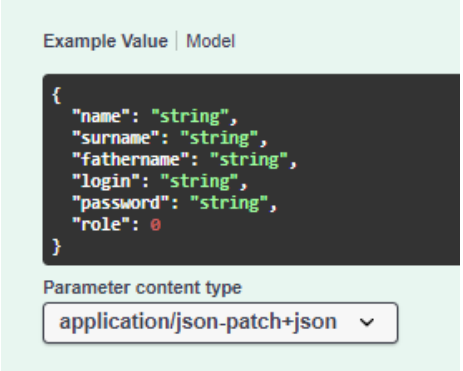

Рис. 2.5. «Вхідна модель на створення акаунту»

2. Auth

– /api/auth/signin (POST) – вхід до системи

Вимагає наступну вхідну модель:

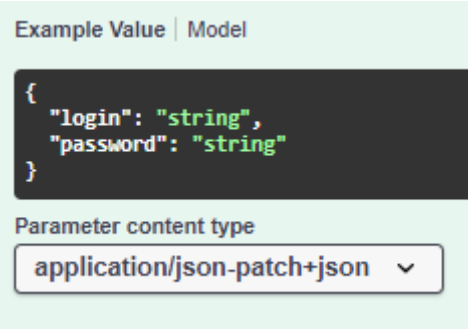

Рис. 2.6. «Вхідна модель на вхід до системи»

– /api/auth/signout (POST) – вихід із системи

Вимагає наступну вхідну модель:

Рис. 2.7. «Вхідна модель на вихід із системи»

– /api/auth/refresh (POST) – оновлення токену доступу

Вимагає наступну вхідну модель:

Рис. 2.8. «Вхідна модель на оновлення токену доступу»

3. Course

– /api/courses (POST) – створення курсу

Вимагає наступну вхідну модель:

Рис. 2.9. «Вхідна модель на створення курсу»

- /api/courses (GET) – отримання курсів

4. Email
   - /api/email/confirm-email (GET) –підтвердження email-у
5. Test
   - /api/tests (POST) – створення тесту

Вимагає наступну вхідну модель:

Рис. 2.10. «Вхідна модель на створення тесту»

- /api/tests (GET) – отримання тестів
- /api/tests/test-results-savage (POST) – збереження результатів проходження тесту

Вимагає наступну вхідну модель:

Рис. 2.11. «Вхідна модель на збереження результатів проходження тесту»

- /api/tests/detailed-test-result (GET) – отримання детальної інформації про результат тесту
- /api/tests/for-evaluation (GET) – отримання тесту викладачем для оцінювання

6. User
   - /api/users (GET) – отримання користувачів
   - /api/users/paginated (GET) – отримання пагінованих користувачів

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для користувача є важливим мати систему, спроможну запускати та працювати з сучасними браузерами, тож необхідно мати такі мінімальні параметри ЕОМ :

- ЦП [CPU]: Pentium 4.
- Відеоадаптер [GPU]: 3D адаптер nVidia, Intel, AMD/ATI.
- Відеопам’ять [VRAM]: 64 МБ.
- Накопічувач [HDD]: 350 МБ.
- Оперативна пам’ять [RAM]: 512 МБ.

2.6.2.Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

– VisualStudio 2019 Community;

– SQL Server;

– Git, GitHub, GithubDesktop.

– IIS-server

– .NET Core Environment

2.6.3.Виклик та завантаження програми

Для роботи з розробленим веб-додатком потрібно лише звернутися за адресою https://testonapi.azurewebsites.net/index.html (підтримується хмарним сервісом Azure та не потребує додаткових дій або ПЗ, за виключенням засобу звернення до інтернету – браузера, вбудованих сервісів Windows тощо).

2.6.4.Опис інтерфейсу користувача

1. Головна сторінка.

Головна сторінка сайту містить туторіал з користування платформою а також є візитною карткою платформи, описуючи функціонал, що надаватиметься користувачеві.

Вона складається з заголовка, у якому можна перейти до авторизації (Рис. 2.12), з інформаційної складової, а також з футера (Рис.2.13), в якому вказані контактні дані розробників для подальшої комунікації.

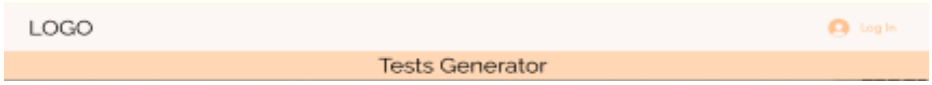

Рис. 2.12. «Заголовок головної сторінки»

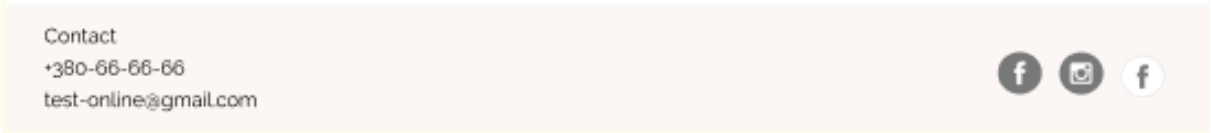

Рис.2.13. «Футер головної сторінки»

В цілому головна сторінка виглядає так:

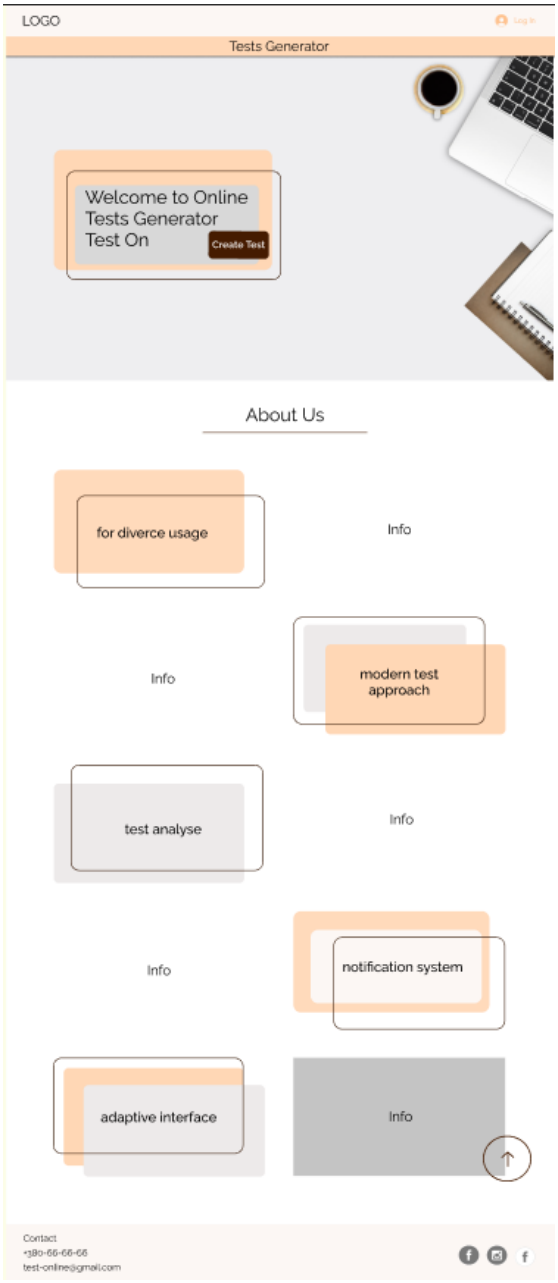

Рис. 2.14. «Головна сторінка»

2. Форма входу до системи

Форма входу до системи (Рис.2.15.) містить поля для вводу логіну (електронної пошти) та паролю. Також з неї можна перейти на форму створення акаунту.

Рис.2.15. «Форма входу до системи»

3. Форма створення акаунту

Форма створення акаунту (Рис.2.16.) дозволяє створити акаунт а також перейти до форми входу до системи, якщо акаунт вже є. Містить ряд наступних полів для вводу необхідної інформації:

- Електронна пошта
- Пароль
- Повторне підтвердження паролю
- Персональні дані (ПІБ)

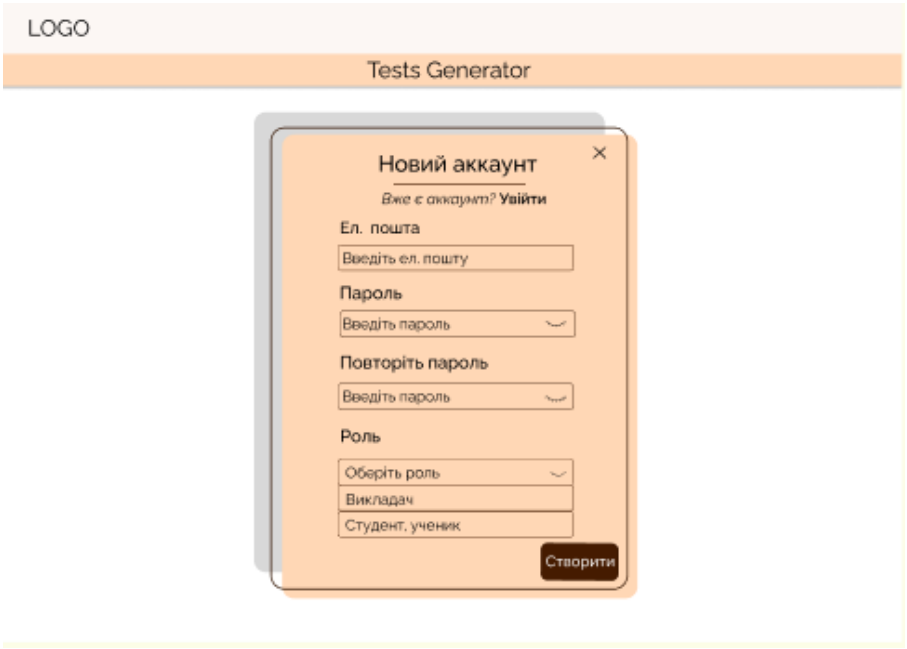

Рис.2.16. «Форма створення акаунту»

Після створення акаунту користувачеві надійде лист електронною поштою, в якому буде посилання для підтвердження електронної пошти (Рис.2.17.).

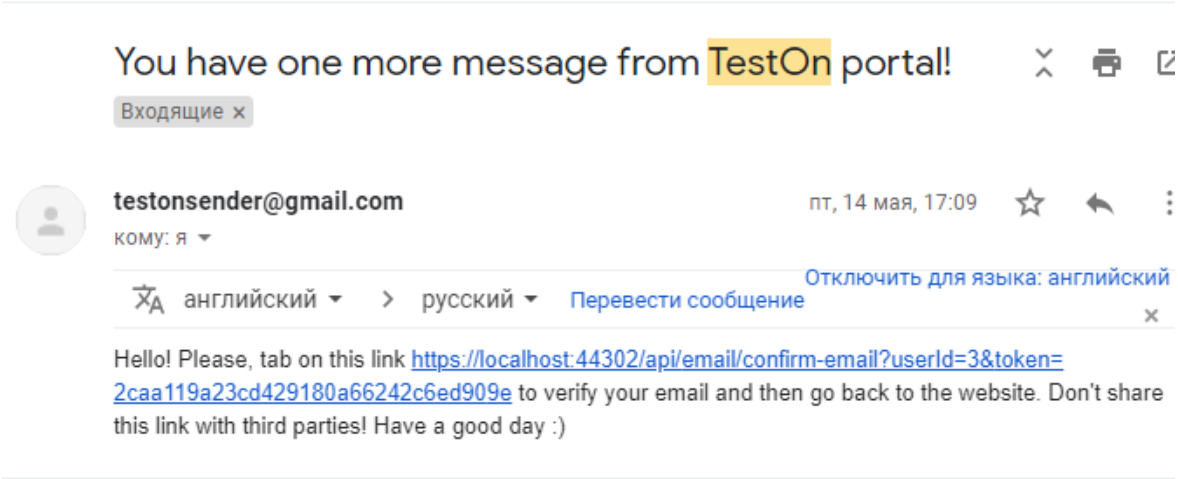

Рис.2.17. «Лист із посиланням для підтвердження електронної пошти користувача»

Без верифікації електронної пошти користувач не зможе увійти до свого акаунту, оскільки він вважається не перевіреним.

Після входу до акаунту користувач має доступ до інших UI-компонентів. Змінюється заголовок: в ньому з'являється ім'я користувача, а також для користувача в ролі вчителя є меню (Рис..2.18.), що складається з кількох пунктів:

- Профіль (форма перегляду інформації про акаунт)
- Оцінювання робіт (дозволяє переходити до пройдені тести, які готові для оцінювання, містить нотифікатор)
- Вихід

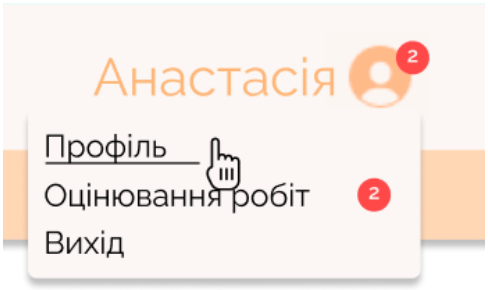

Рис.2.18. «Меню користувача в ролі вчителя»

Також для користувача в ролі вчителя наявна меню-стрічка, що містить наступні поїнти:

- Тести (Рис.2.19.) –містить два підпункти:
  - Тести (форма перегляду тестів)
  - Додати тест (форма створення тесту)

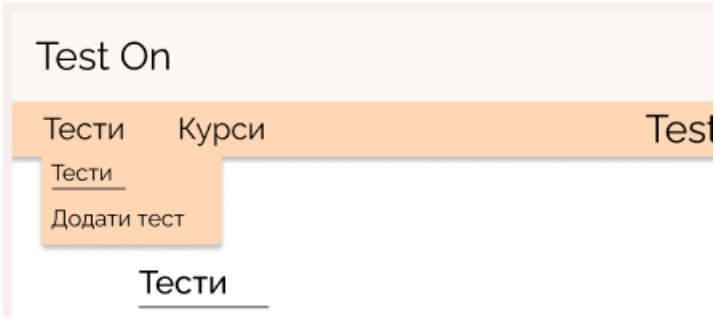

Рис.2.19. «Меню для маніпуляцій із тестами»

– Курси (Рис.2.20.) – містить два підпункти:
  – Курси (форма перегляду курсів)
  – Додати курс (форма створення курсу)

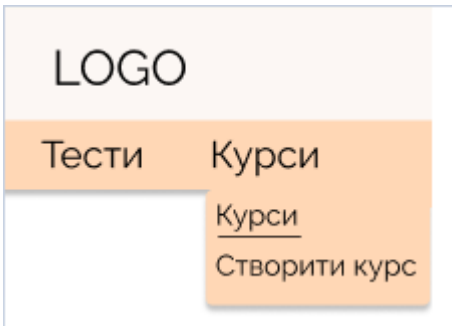

Рис.2.20. «Меню для маніпуляцій із курсами»

4. Форма перегляду інформації про акаунт

Надає користувачу змогу передивлятися інформацію про власні дані, закріплені за його акаунтом.

Рис.2.21. «Форма перегляду інформації про акаунт»

5. Форма перегляду тестів

Форма перегляду тестів (Рис.2.22.) дозволяє користувачу в ролі вчителя передивлятися інформацію про створені ним тести. Має набір фільтрів таких, як:

– За курсом
– За датою проведення тесту
– За статусом тесту

Також містить пошукове поле для навігації за записами. Містить кнопку переходу до форми створення тесту.

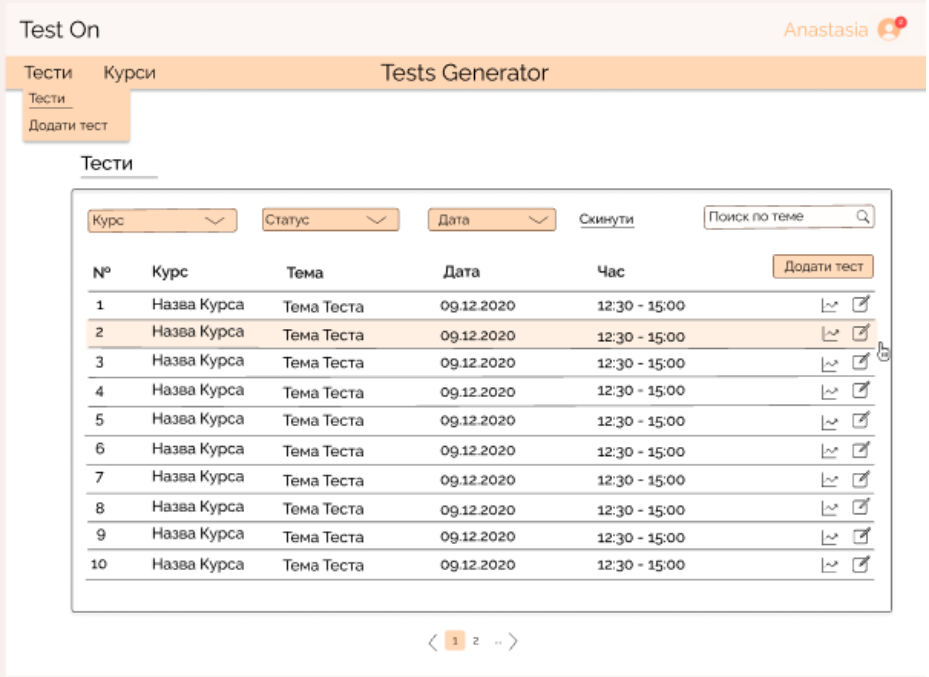

Рис.2.22. «Форма створення тесту»

6. Форма створення тесту

Багатоярусний компонент, що дозволяє створювати тест. Містить sidebar із покроковими етапами створення тесту для кращої навігації за сторінкою (Рис.2.23.).

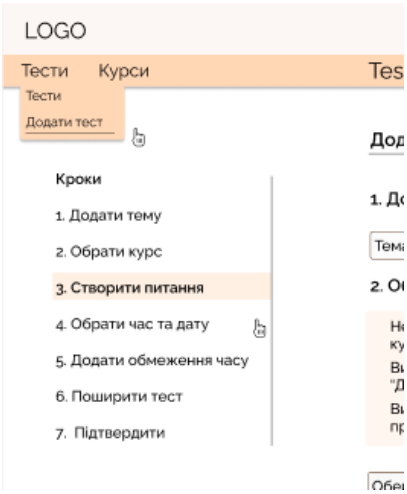

Рис.2.23. «Покроковий туторіал створення тесту»

Містить компоненти для додання інформації про тест (Рис.2.24., Рис.2.25), підказки для ліпшого опанування системою (Рис.2.26.).

Рис.2.24-2.25. «Додання інформації про тест»

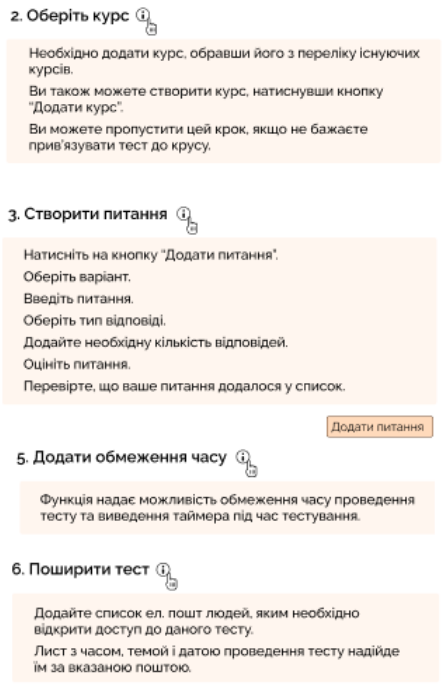

Рис.2.26. «Підказки для ліпшого опанування системою»

Розглянемо більш детально список компонентів для створення тесту. Всього є 6 пунктів:

1. Тема тесту – текстове поле, що дозволяє ввести тему тесту
2. Обрання курсу – опційне поле – тести можна закріплювати за певним курсом для ліпшої структуризації інформації, форма для обрання курсу являє собою список усіх курсів даного користувача.
3. Створення питання

Цей компонент містить кнопку переходу до форми створення питання, а також дозволяє попередній перегляд вже створених для даного тесту питань.

4. Обрання дати та часу проведення тестування
5. Додання обмеження за часом

Опційне поле. Якщо користувач бажає, він може обмежити час проходження тестування для своїх учнів.

6. Поширення тесту

Після створення тесту учні, яких викладач зробив підписниками на даний тест, будуть попереджені про те, що вони підписані на тест. Їм надійде лист наступного змісту на електронну пошту (Рис. 2.27).

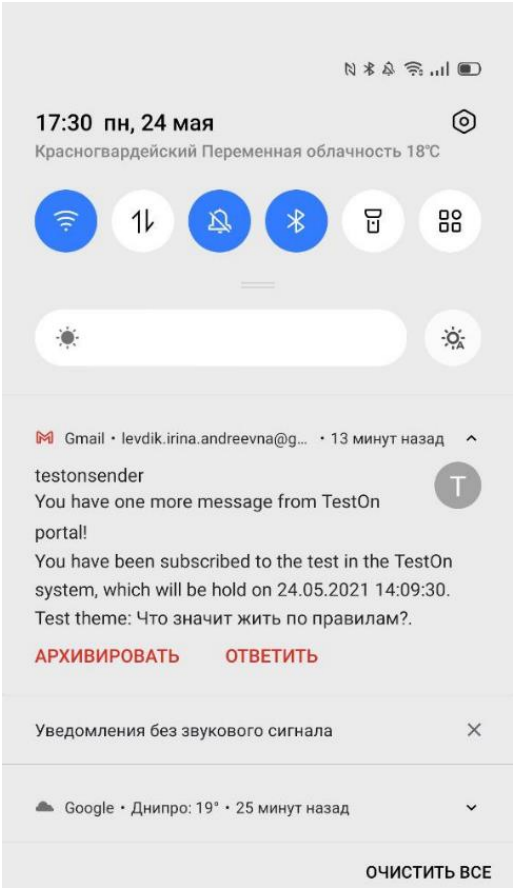

Рис.2.27. «Лист нотифікації про створену підписку на тест»

7. Форма створення питання

Форма створення питання (Рис.2.28.) дозволяє вводити інформацію, необхідну для формування питання для тесту і містить наступні поля для вводу даних:

- Варіант
- Питання
- Тип відповіді – дозволяє формувати тип відповіді для питання.

Всього в системі наявні 4 типів питань:

1. Питання з однією правильною відповіддю

2. Питання з кількома правильними відповідями
3. Питання з відкритою відповіддю (правильної відповіді немає, відповідь перевіряється викладачем)
4. Питання з файловою відповіддю (правильної відповіді немає, студент прикріпляє файл, який потім переглядається та перевіряється викладачем)
    - Варіанти відповідей – додаються викладачем, якщо питання не відкрите і не файлове
    - Список відповідей – вказуються правильні відповіді
    - Оцінка за питання – максимальна оцінка, що може бути виставлена за питання.

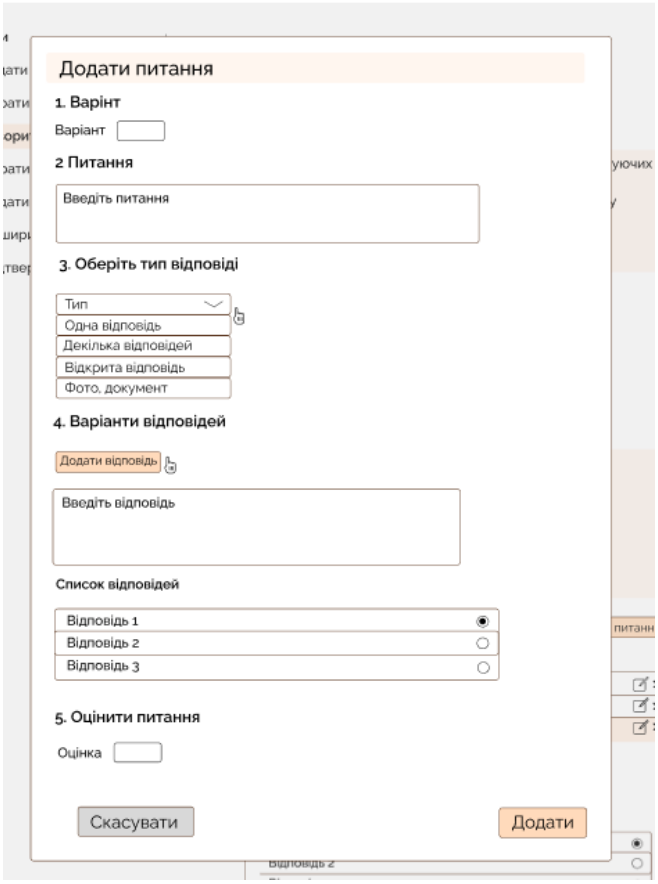

Рис.2.28. «Форма створення питання»

8. Форма попереднього перегляду тесту

Перед збереженням тесту можна передивитися створений тест на наявність невідповідностей або помилок.

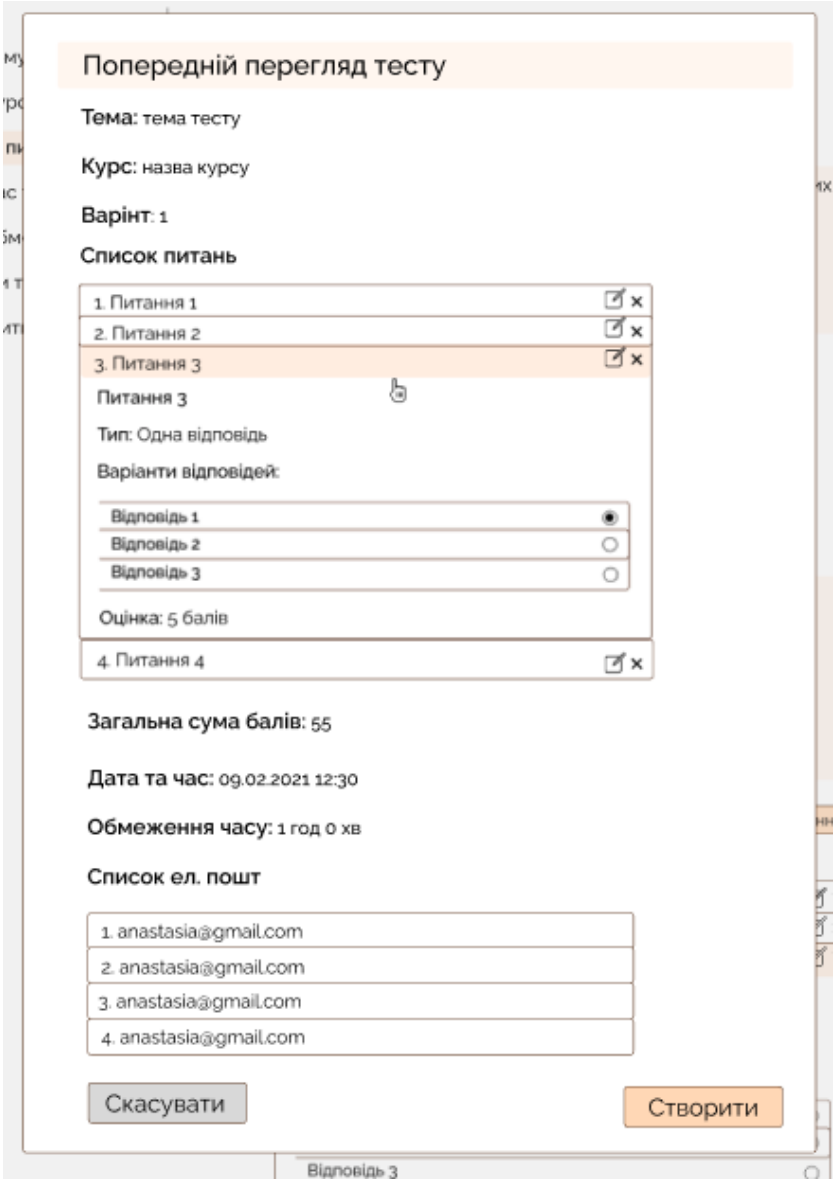

Рис.2.29. «Форма попереднього перегляду тесту».

9. Перегляд детальної інформації за тестом

Для кожного створеного тесту можна переглянути детальну інформацію, що за ним закріплена.

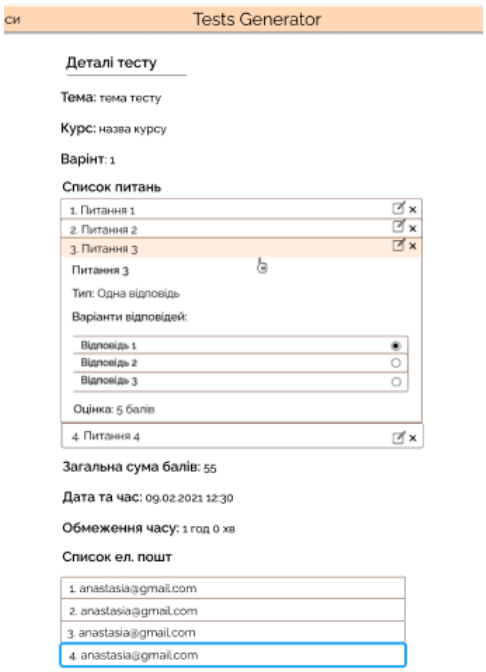

Рис.2.30. «Деталі тесту»

10. Перегляд вхідних робіт на оцінювання

Зі свого меню у заголовку викладач має змогу передивитися список вхідних робіт студентів, які чекають на оцінювання (Рис.2.31.). Із цього списку можна перейти на сторінку оцінювання роботи.

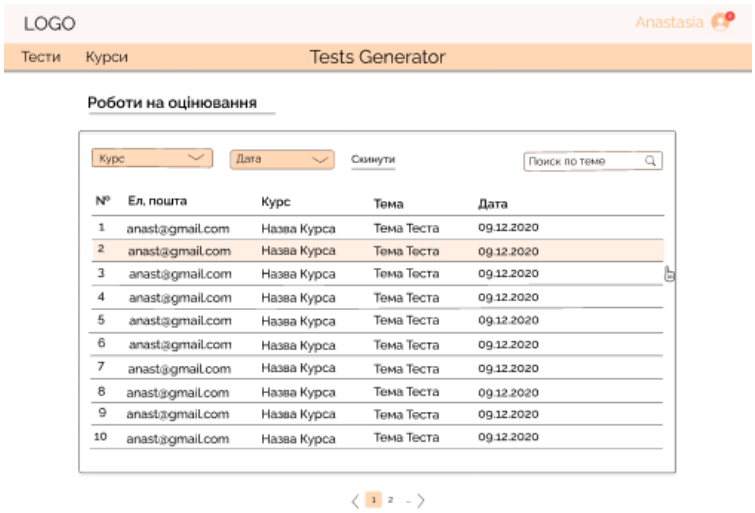

Рис.2.31. «Перегляд вхідних робіт на оцінювання»

11. Оцінювання результатів тестування

Компонент містить інформацію про відповіді, надані студентом. Дозволяє переглядати відповіді на тестові завдання, оцінювати відкриті або файлові завдання.

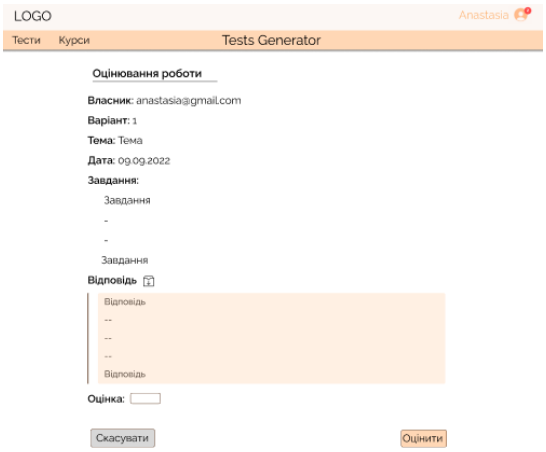

Рис.2.32. «Оцінювання результатів тестування»

12. Аналіз результатів тестування

Дозволяє переглядати середній бал всіх результатів проходження певного тестування.

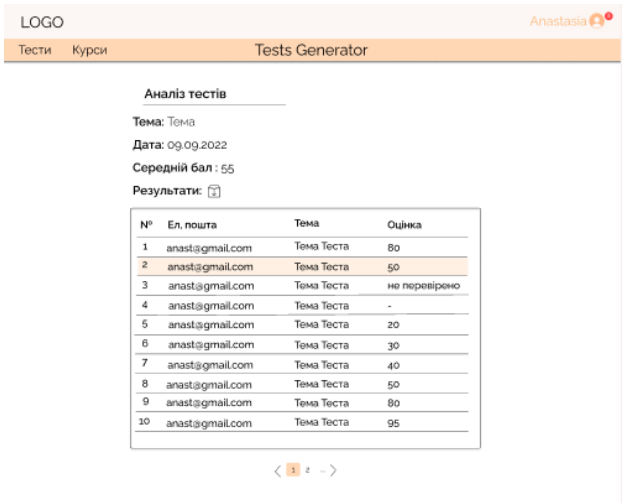

Рис.2.33. «Аналіз результатів тестування»

13. Форма перегляду списку курсів

Дозволяє передивлятися список створених користувачем курсів.

З форми перегляду списку курсів користувач має можливість додавання нових підписників на курс, можливість перегляду деталей курсу, додання нового курсу.

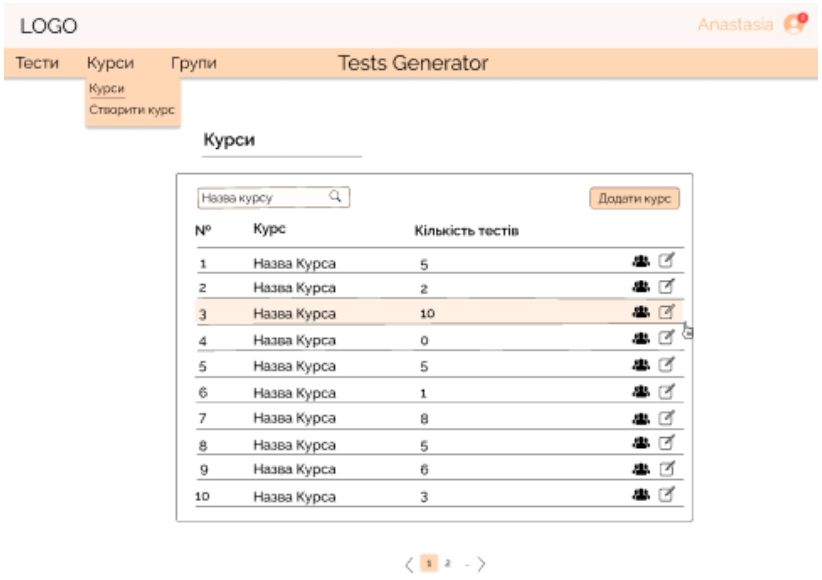

Рис.2.34. «Перегляд списку курсів користувача»

14. Створення курсу

Форма дозволяє створювати курс, потребує вводу наступних даних:

- Назва курсу
- Підписники курсу (опційно)

Рис.2.35. «Створення курсу»

15. Додання підписників на курс

Форма редагування вже існуючого курсу, дозволяє додавати нових підписників.

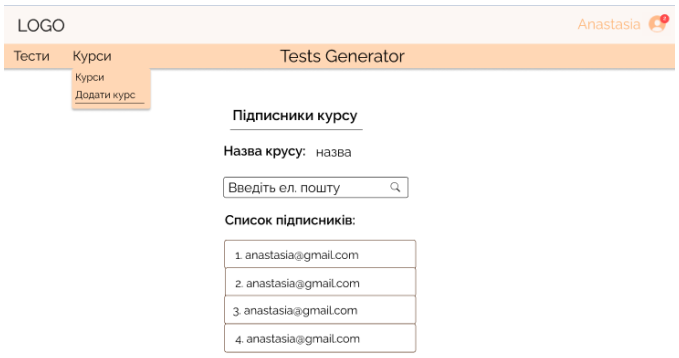

Рис.2.36. «Додання підписників на курс»

Сайт оптимізований для перегляду на всіх девайсах без горизонтальної смуги прокрутки (за винятком звітів) і без порожніх (білих) полів для основних типів дозволу. На кожній сторінці відображається контактна інформація.

Інтерфейс модулів виконаний в єдиному стилі з інтерфейсом ядра системи і забезпечує можливість прозорого переміщення між модулями системи і використання однакових процедур управління і навігаційних елементів для виконання однотипних операцій.

РОЗДІЛ 3

ЕКОНОМІЧНИЙ РОЗДІЛ

Автоматизуючи суспільні процеси за допомогою ПЗ важливо визначити трудомісткість розробки програмного забезпечення і розрахувати витрати на створення інформаційної системи.

3.1. Розрахунок трудомісткості та вартості розробки інформаційної системи

Початкові дані:

1. передбачуване число операторів – 1000;

2. коефіцієнт складності програми – 1.5;

3. коефіцієнт корекції програми в ході її розробки – 0.09;

4. годинна заробітна плата програміста, грн/год – 78;

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1.2;

7. вартість машино-години, грн/год – 10.

Нормування праці в процесі створення програмного забезпечення істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки програмного забезпечення може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки програмного забезпечення можна розрахувати за формулою:

$$t = t_0 + t_u + t_a + t_n + t_{отл} + t_д, людино - годин, \quad (3.1)$$

де t_0 - витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ - витрати праці на дослідження алгоритму рішення задачі;

$t_а$ - витрати праці на розробку блок-схеми алгоритму;

$t_п$ - витрати праці на програмування за готовою блок-схемою;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється. Умовне число операторів (тегів):

$$Q = q * C * (1 + p), \qquad (3.2)$$

де q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт корекції програми в ході її розробки.

Звідси умовне число операторів в програмі:

$$Q = 1000 * 1{,}5 * (1 + 0{,}09) = 1635. \qquad (3.3)$$

Витрати праці на вивчення опису задачі $t_и$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75...85)*k}, людино - годин, \qquad (3.4)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{1635*1,3}{85*1,2} = 21, людино - годин. \quad (3.5)$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)*k}, людино - годин \quad (3.6)$$

$$t_a = \frac{1635}{25*1,2} = 55, людино - годин. \quad (3.7)$$

Витрати на складання програми за готовою блок-схемою:

$$t_n = \frac{Q}{(20...25)*k}, людино - годин, \quad (3.8)$$

$$t_n = \frac{1635}{25*1,2} = 55, людино - годин. \quad (3.9)$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)*k}, людино - годин. \quad (3.10)$$

$$t_{отл} = \frac{1635}{5*1,2} = 273, людино - годин. \quad (3.11)$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 * t_{отл}, людино - годин. \quad (3.12)$$

$$t_{отл}^{k} = 1{,}5 * 273 = 410, людино - годин. \quad (3.13)$$

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, людино - годин, \quad (3.14)$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{15...20*k}, людино - годин. \quad (3.15)$$

$$t_{др} = \frac{1635}{12*1{,}2} = 14, людино - годин, \quad (3.16)$$

де $t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 * t_{др}, людино - годин. \quad (3.17)$$

$$t_{до} = 0{,}75 * 14 = 11, людино - годин. \quad (3.18)$$

$$t_{д} = 14 + 11 = 25, людино - годин. \quad (3.19)$$

Тепер розрахуємо трудомісткість ПЗ:

$$t = 50 + 21 + 55 + 55 + 273 + 25 = 479, людино - годин. \quad (3.20)$$

3.2. Розрахунок витрат на створення інформаційної системи

Витрати на створення програмного забезпечення Витрати на створення даного продукту $К_{по}$ включають витрати на заробітну плату виконавця програми $З_{зп}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{по} = З_{зп} + З_{мв}, грн. \quad (3.21)$$

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t * С_{пр}, грн, \quad (3.22)$$

де t - загальна трудомісткість, людино-годин;

$С_{пр}$ - середня годинна заробітна плата програміста, грн/година.

$$З_{зп} = 479 * 78 = 37{,}362грн. \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} * С_{мч}, грн, \quad (3.24)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 273 * 10 = 2730 грн. \quad (3.25)$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП:

$$К_{по} = 37{,}362 + 2730 = 40{,}092 грн. \quad (3.26)$$

Очікуваний період створення ПЗ:

$$Т = \frac{t}{B_к * F_р}, міс, \quad (3.27)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$Т = \frac{479}{1*176} = 3 міс. \quad (3.28)$$

Висновок:

У кваліфікаційній роботі була створена контрольно-діагностична платформа для створення та проходження онлайн-тестувань. В третьому (економічному) розділі були визначені витрати на створення ПЗ, зазначеного у технічному завданні КР – 40,092 грн. і час створення програмного забезпечення – 3 місяці.

ВИСНОВКИ

Об’єктом розробки та аналізу предметної області є контрольно-діагностична платформа для створення та проведення автоматизованого онлайн-тестування.

Актуальність платформи зумовлена необхідністю автоматизації процесу оцінювання, контролю, діагностики, обробки результатів навчання.

Створена платформа реалізує наступні функціональні характеристики:

- забезпечує користувачів у ролі викладача можливістю створювати тест
- програмний продукт забезпечує користувачів у ролі учня можливістю проходити тест, на який учня підписує викладач
- програмний продукт забезпечує користувачів можливістю створювання акаунту, його редагування, перегляду інформації
- система контролює доступ користувачів до інформації залежно від їхніх ролей та креденшелів
- система зберігає результати проходження тесту кожним користувачем у ролі учня та надає змогу передивлятися хронологію пройдених тестувань
- система зберігає тести, створені користувачем у ролі викладача та надає змогу передивлятися їхні деталі
- платформа надає ряд статистичних метрик для будування діаграм на клієнтській частині.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Terry Felke-Morris. Web Development and Design Foundations with HTML5, 9th Edition. Pearson Higher Education, 2018. 720 с.
2. Keith J. Grant. CSS in Depth. Manning, 2018. 472 с.
3. Jogn Dean. Web programming with HTML5, CSS, and JavaScript. O'Reilly Media, 2019. 678 с.
4. Marijn Haverbeke. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming. O'Reilly Media, 2019. 474 с.
5. Robin Wieruch. The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js. CreateSpace Independent Publishing Platform, 2018. 208 с.
6. М. Пацианский. Основы Redux. Питер, 215 с.
7. Jaroslaw Krochmalski. IntelliJ IDEA Essentials. Packt Publishing, 2014. 276с.
8. Brent Laster. Professional Git. 1st Edition. Wrox, 2016. 433 с.
9. Воронов М.В. Профессиональное обучение студентов на основе интегрированных курсов // Инновации в образовании. – 2011. - № 9. – С. 4-15.
10. Гришнова Е.Е. Модернизация учебного процесса: проблемы и тенденции // Высшее образование в России. – 2011. - № 8-9. – С. 41-46.
11. Игошев Б.М. Современное образование: проблемы и решения // Альма Матер. – 2011. - № 10. – С. 6-11.
12. Симан М. Внедрение зависимостей в.NET. / Марк Симан., 2014. – 464 с. – (2). – (Для профессионалов).
13. Price M. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition / Mark Price., 2021. – 816 с. – (4).
14. Эффективность применения технологии Asp. Net Core для разработки веб-приложений [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/effektivnost-primeneniya-tehnologii-asp-net-core-dlya-razrabotki-veb-prilozheniy/viewer. (останній доступ 09.06.2021)

15. С. А. Л. ВОЗМОЖНОСТИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ C# (C SHARP) [Електронний ресурс] / А. Л. С., Э. А. Бекирова (Менумерова). – 2018. – Режим доступу до ресурсу: https://www.elibrary.ru/item.asp?id=36605587. (останній доступ 09.06.2021)
16. Чеглаков А. Л. Композиция web-сервисов на основе архитектуры rest [Електронний ресурс] / А. Л. Чеглаков. – 2016. – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/kompozitsiya-web-servisov-na-osnove-arhitektury-rest/viewer. (останній доступ 09.06.2021)
17. Щенникова Е. В. СРАВНИТЕЛЬНЫЙ АНАЛИЗ ТЕХНОЛОГИИ ASP.NET CORE [Електронний ресурс] / Е. В. Щенникова, О. С. Макаров. – 2020. – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-tehnologii-asp-net-core/viewer. (останній доступ 09.06.2021)
18. Makosiy R. DevOps concepts [Електронний ресурс] / Roman Makosiy. – 2017. – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/devops-concepts/viewer. (останній доступ 09.06.2021)
19. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. – Питер, 2013. – 928 с.
20. Хейлсберг А. Язык программирования C# / А. Хейлсберг. – Питер, 2016. – 784 с.

ДОДАТОК А

КОД ПРОГРАМИ

AccountController.cs

```
using Microsoft.AspNetCore.Mvc;
using System;
using DiplomaServices.Models;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;

namespace DiplomaAPI.Controllers
{
    [Route("api/account")]
    [ApiController]
    public class AccountController : ControllerBase
    {
        #region

        private readonly IAccountService accountService;

        private readonly IEmailService emailService;

        #endregion

        public AccountController(IAccountService accountService, IEmailService emailService)
        {
            this.accountService = accountService;
            this.emailService = emailService;

            mapper = new MapperService();
        }
        // POST: api/account
        [HttpPost]
        public IActionResult CreateAccount(CreateAccountModel added)
        {
            try
            {
                var response = accountService.CreateAccount(added);

                //verifying email
                var confirmationLink = Url.Action("ConfirmEmail", "Email",
                    new
                    {
                        userId = response,
                        token = emailService.GenerateConfirmationToken()
                    },
                    Request.Scheme);

                //emailService.SetConfirmationLink(confirmationLink);
                emailService.SendMessage(added.Login, null, confirmationLink);

                return Ok();
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }
    }
```

```
}
```

AuthController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Text;
using DiplomaServices.Models;
using DiplomaServices.Interfaces;

namespace DiplomaAPI.Controllers
{
    [Route("api/auth")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        #region

        private readonly IAuthService authService;

        #endregion

        public AuthController(IAuthService authService)
        {
            this.authService = authService;
        }


        [HttpPost]
        [Route("signin")]
        public IActionResult SignIn(SignInModel signIn)
        {
            try
            {
                var response = authService.SignIn(signIn);

                HttpContext.Session.Set("refreshToken",
Encoding.ASCII.GetBytes(response.RefreshToken));
                HttpContext.Session.Set("accessToken",
Encoding.ASCII.GetBytes(response.AccessToken));
                HttpContext.Session.Set("userLogin",
Encoding.ASCII.GetBytes(response.UserLogin));
                HttpContext.Session.Set("userName",
Encoding.ASCII.GetBytes(response.UserName));
                HttpContext.Session.Set("userId", Encoding.ASCII.GetBytes(response.UserId));

                return Ok(response);
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        [HttpPost]
        [Route("signout")]
        public IActionResult SignOut(SignOutModel signOut)
        {
            try
            {
```

```
                var userId = Convert.ToInt32(new
string(Encoding.ASCII.GetChars(HttpContext.Session.Get("userId"))));
                signOut.UserId = userId;
                authService.SignOut(signOut);

                HttpContext.Session.Clear();

                return Ok();
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        [HttpPost]
        [Route("refresh")]
        public IActionResult Refresh(RefreshModel model)
        {
            try
            {
                if (HttpContext.Session.Get("refreshToken") != null)
                {
                    var incomingRefresh = model.RefreshToken;
                    var refreshFromSession = new
string(Encoding.ASCII.GetChars(HttpContext.Session.Get("refreshToken")));

                    if (refreshFromSession == incomingRefresh)
                    {
                        var accessToken = authService.RefreshAccessToken();
                        HttpContext.Session.Set("accessToken",
Encoding.ASCII.GetBytes(accessToken));

                        return Ok(new { AccessToken = accessToken });
                    }
                    else
                    {
                        //We can't give new access if refresh token wasn't found

                        return BadRequest("Your refresh token isn't right, access denied");
                    }
                }

                return BadRequest("Your session has expired. Please, re-login to the
system.");

            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

    }
}
```

TestControler.cs

```
using DiplomaAPI.Filters;
using DiplomaServices.Interfaces;
using DiplomaServices.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
```

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Text;

namespace DiplomaAPI.Controllers
{
    [Route("api/tests")]
    [TypeFilter(typeof(AuthFilter))]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [ApiController]
    public class TestController : ControllerBase
    {
        #region

        private readonly ITestService testService;

        #endregion

        public TestController(ITestService testService)
        {
            this.testService = testService;
        }

        [HttpPost]
        public IActionResult CreateTest(CreateTestModel model)
        {
            try
            {
                return Ok(testService.CreateTest(model));
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        [HttpGet]
        public IActionResult GetTests()
        {
            try
            {
                return Ok(testService.GetAll());
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        [HttpPost]
        [Route("test-results-savage")]
        public IActionResult SaveStudentTestResults(SavePassedTestResultsModel testResult)
        {
            try
            {
                var userId = Convert.ToInt32(new
string(Encoding.ASCII.GetChars(HttpContext.Session.Get("userId"))));
                testService.ProcessTestResultSaving(testResult, userId);

                return Ok();
            }
```

```
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        [HttpGet]
        [Route("detailed-test-result")]
        public IActionResult GetDetailedTestResultByStudentId(int studentId)
        {
            return Ok();
        }

        //[HttpGet]
        //[Route("test-result")]
        //public IActionResult GetTestResultsByTestId(int testId)
        //{
        //    return Ok();
        //}

        [HttpGet]
        [Route("for-evaluation")]
        public IActionResult GetTestsPassedForEvaluation(int studentId)
        {
            return Ok();
        }
    }
}
```

CourseService.cs

```
using DataAccess;
using DataAccess.Entities;
using DataAccess.Entities.ManyToManyEntities;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;
using DiplomaServices.Models;
using System.Collections.Generic;

namespace DiplomaServices.Services
{
    public class CourseService : ICourseService
    {
        #region private members

        private readonly IUnitOfWork uow;

        private readonly IUserService userService;

        private readonly MapperService mapper;

        #endregion

        public CourseService(IUnitOfWork uow, IUserService userService)
        {
            this.uow = uow;
            this.userService = userService;

            mapper = new MapperService();
        }

        public bool CheckIfCourseExists(int id)
        {
            var course = uow.Courses.Get(c => c.Id == id);
```

```
            return course != null;
        }

        public AddCourseApplicantsResponseModel CreateCourse(CreateCourseModel model)
        {
            var course = mapper.Map<CreateCourseModel, Course>(model);
            uow.Courses.Create(course);
            uow.Save();

            var response = AddApplicants(model.Applicants, course.Id);

            return response;
        }

        public IEnumerable<Course> GetAll()
        {
            return uow.Courses.GetAll();
        }
        private AddCourseApplicantsResponseModel AddApplicants(List<CreateApplicantModel>
applicants, int courseId)
        {
            var applicantsToAdd = new List<CreateApplicantModel>();
            var failedApplicantsToAdd = new List<FailedApplicantResponseModel>();

            ValidateApplicants(applicantsToAdd, failedApplicantsToAdd, applicants);

            foreach (var applicant in applicantsToAdd)
            {
                var user = uow.Users.Get(u => u.Login == applicant.Login);
                var usersCourses = new UsersCourses
                {
                    UserId = user.Id,
                    CourseId = courseId
                };

                uow.UsersCourses.Create(usersCourses);
                uow.Save();
            }

            var response = new AddCourseApplicantsResponseModel();

            response.SuccessfullyAddedApplicants = applicantsToAdd;
            response.FailedAddedApplicants = failedApplicantsToAdd;

            return response;
        }

        private void ValidateApplicants(
            List<CreateApplicantModel> applicantsToAdd,
            List<FailedApplicantResponseModel> failedApplicantsToAdd,
            List<CreateApplicantModel> applicants)
        {

            foreach (var applicant in applicants)
            {
                var isExistResponse = userService.CheckIfUserExists(applicant.Login);

                if (isExistResponse.IsFound)
                {
                    applicantsToAdd.Add(applicant);
                }
                else
```

```
                {
                    failedApplicantsToAdd.Add(new FailedApplicantResponseModel
                    {
                        ErrorMessage = isExistResponse.ErrorMessage,
                        Login = isExistResponse.Login
                    });

                }
            }
        }
    }
}
```

TestService.cs

```
using DataAccess;
using DataAccess.Entities.ManyToManyEntities;
using DataAccess.Entities.TestEntities;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;
using DiplomaServices.Models;
using System;
using System.Collections.Generic;

namespace DiplomaServices.Services.TestServices
{
    public class TestService : ITestService
    {
        #region private members

        private readonly IUnitOfWork uow;

        private readonly ICourseService courseService;

        private readonly IQuestionService questionsService;

        private readonly IEmailService emailService;

        private readonly IUserService userService;

        private readonly IAnswersService answersService;

        private readonly MapperService mapper;

        #endregion

        public TestService(IUnitOfWork uow,
            ICourseService courseService,
            IQuestionService questionsService,
            IEmailService emailService,
            IUserService userService,
            IAnswersService answersService)
        {
            this.uow = uow;
            this.courseService = courseService;
            this.questionsService = questionsService;
            this.emailService = emailService;
            this.userService = userService;
            this.answersService = answersService;

            mapper = new MapperService();
        }
        public CreateTestModel CreateTest(CreateTestModel model)
        {
```

```
            var test = new Test();

            //Bind Course to Test
            if (courseService.CheckIfCourseExists(model.CourseId.Value))
            {
                test.CourseId = model.CourseId.Value;
            }
            else
            {
                throw new Exception(string.Format("Course with id {0} doesn't exist!",
model.CourseId));
            }

            test.ComplitionDate = model.DateTime;
            test.Theme = model.Theme;
            uow.Tests.Create(test);
            uow.Save();

            //Create Questions
            if (model.Questions.Count > 0)
            {
                foreach (var question in model.Questions)
                {
                    question.TestId = test.Id;

                    questionsService.CreateQuestion(question);
                }
            }
            else
            {
                throw new Exception("Test doesn't contain questions!");
            }

            //Add Applicants
            if (model.Applicants.Count > 0)
            {
                AddApplicants(model.Applicants, test.Id);
            }

            //Notify Applicants about Test
            if (model.Applicants.Count > 0)
            {
                foreach (var applicant in model.Applicants)
                {
                    var isExistResponse = userService.CheckIfUserExists(applicant.Login);

                    if (isExistResponse.IsFound)
                    {
                        string messageText = string.Format("You have been subscribed to the
test in the TestOn system, which will be hold on {0}. " +
                            "Test theme: {1}.", model.DateTime, model.Theme);

                        emailService.SendMessage(applicant.Login, messageText, null);
                    }
                }
            }

            return null;
        }
        public IEnumerable<Test> GetAll()
        {
            return uow.Tests.GetAll();
        }
```

```
        public IEnumerable<GetTestDetailsModel> GetTestDetailsByStudentId(int userId)
        {
            return null;
        }
        public IEnumerable<decimal> ProcessTestResultSaving(SavePassedTestResultsModel
testResult, int userId)
        {
            var gradesForTest = new List<decimal>();

            foreach (var question in testResult.Questions)
            {
                answersService.SaveAnswers(question, userId, testResult.TestId);
                var gradeForQuestion =
answersService.EvaluateAnswersForQuestion(question.QuestionId, userId,
question.ChosenResponseOptionIds);

                gradesForTest.Add(gradeForQuestion);
            }

            return gradesForTest;
        }
        private void AddApplicants(List<CreateApplicantModel> applicants, int testId)
        {
            foreach (var applicant in applicants)
            {
                var isExistResponse = userService.CheckIfUserExists(applicant.Login);

                if (isExistResponse.IsFound)
                {
                    var user = uow.Users.Get(u => u.Login == applicant.Login);

                    uow.UsersTests.Create(new UsersTests { UserId = user.Id, TestId = testId
});


                    uow.Save();
                }

            }
        }
    }
}
```

QuestionService.cs

```
using DataAccess;
using DataAccess.Entities.Answers;
using DataAccess.Entities.TestEntities;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;
using DiplomaServices.Models;
using System.Collections.Generic;

namespace DiplomaServices.Services.TestServices
{
    public class QuestionService : IQuestionService
    {
        #region private members

        private readonly IUnitOfWork uow;

        private readonly MapperService mapper;
```

```
        #endregion

        public QuestionService(IUnitOfWork uow)
        {
            this.uow = uow;
            mapper = new MapperService();
        }

        public void CreateQuestion(CreateQuestionModel model)
        {
            //Create Question
            var question = mapper.Map<CreateQuestionModel, Question>(model);
            uow.Questions.Create(question);
            uow.Save();

            //Create Response Options
            var fullResponseOptions = CreateResponseOptionsForTheQuestion(model,
question.Id);

            //Create RightAnswers
            if(!model.IsFileQuestion && !model.IsOpenQuestion && fullResponseOptions !=
null)
            {
                CreateRightAnswersForTheQuestion(fullResponseOptions);
            }
    }

        public List<GetQuestionModel> GetQuestionsForTheTest(int testId)
        {
            var questions = uow.Questions.GetSeveral(q => q.TestId == testId);
            var responseQuestionModels = new List<GetQuestionModel>();

            foreach (var question in questions)
            {
                var responseOptionsByQuestion = uow.ResponseOptions.GetSeveral(rp =>
rp.QuestionId == question.Id);
                var getResponseOptions = new List<GetResponseOptionModel>();

                foreach (var item in responseOptionsByQuestion)
                {
                    getResponseOptions.Add(new GetResponseOptionModel { Value = item.Value
});
                }

                var responseQuestionModel = new GetQuestionModel()
                {
                    Title = question.Title,
                    ResponseOptions = getResponseOptions,
                    IsFileQuestion = question.IsFileQuestion,
                    IsOpenQuestion = question.IsOpenQuestion
                };

                responseQuestionModels.Add(responseQuestionModel);
            }

            return responseQuestionModels;
        }

        private IEnumerable<CreateResponseOptionModelWithId>
CreateResponseOptionsForTheQuestion(CreateQuestionModel model, int questionId)
        {
            List<CreateResponseOptionModelWithId> fullResponseOptions = new
List<CreateResponseOptionModelWithId>();
```

```
            if (!model.IsFileQuestion && !model.IsOpenQuestion &&
model.ResponseOptions.Count > 0)
            {
                foreach (var responseOption in model.ResponseOptions)
                {
                    decimal grade = 0;

                    var numberOfValidAnswers = CountNumberOfValidAnswers(model);

                    if (numberOfValidAnswers > 1)
                    {
                        grade = model.Grade/ numberOfValidAnswers;
                    }
                    else
                    {
                        grade = model.Grade;
                    }

                    var responseOptionEntity = mapper.Map<CreateResponseOptionModel,
ResponseOption>(responseOption);
                    responseOptionEntity.QuestionId = questionId;

                    uow.ResponseOptions.Create(responseOptionEntity);
                    uow.Save();

                    fullResponseOptions.Add(new CreateResponseOptionModelWithId
                    {
                        Id = responseOptionEntity.Id,
                        Value = responseOptionEntity.Value,
                        IsValid = responseOption.IsValid,
                        Grade = grade
                    });
                }

                return fullResponseOptions;
            }
            else
            {
                return null;
            }
        }

        private void
CreateRightAnswersForTheQuestion(IEnumerable<CreateResponseOptionModelWithId>
fullResponseOptions)
        {
            List<CreateRightAnswerModel> rightAnswersToCreate = new
List<CreateRightAnswerModel>();

            foreach (var fullResponseOption in fullResponseOptions)
            {
                if (fullResponseOption.IsValid)
                {
                    var rightAnswer = new CreateRightAnswerModel() { ResponseOptionId =
fullResponseOption.Id, Grade = fullResponseOption.Grade };
                    rightAnswersToCreate.Add(rightAnswer);
                }
            }

            foreach(var rightAnswer in rightAnswersToCreate)
            {
```

```
                uow.RightSimpleAnswers.Create(mapper.Map<CreateRightAnswerModel,
RightSimpleAnswer>(rightAnswer));
                uow.Save();
            }
        }

        private decimal CountNumberOfValidAnswers(CreateQuestionModel model)
        {
            var numberOfValidAnswers = 0;

            foreach (var responseOption in model.ResponseOptions)
            {
                if (responseOption.IsValid)
                {
                    numberOfValidAnswers++;
                }
            }

            return numberOfValidAnswers;
        }
    }
}
```

AnswersService.cs

```
using DataAccess;
using DataAccess.Entities.Answers;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;
using DiplomaServices.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.IO;

namespace DiplomaServices.Services.TestServices
{
    public class AnswersService : IAnswersService
    {

        #region private members

        private readonly IUnitOfWork uow;

        #endregion

        public AnswersService(IUnitOfWork uow)
        {
            this.uow = uow;
        }
        public void SaveAnswers(SavePassedTestResultQuestionModel questionWithAnswers, int
userId, int testId)
        {
            var question = uow.Questions.Get(q => q.Id == questionWithAnswers.QuestionId);

            if (question.IsFileQuestion)
            {
                SaveFiledAnswer(userId, question.Id, testId, questionWithAnswers.FileName,
questionWithAnswers.OpenOrFileAnswerValue);
            }
            else if (question.IsOpenQuestion)
            {
                SaveOpenAnswer(userId, question.Id,
questionWithAnswers.OpenOrFileAnswerValue);
```

```
            }
            else
            {
                SaveUsualAnswer(userId, question.Id,
questionWithAnswers.ChosenResponseOptionIds);
            }
        }


        public decimal EvaluateAnswersForQuestion(int questionId, int userId, List<int>
chosenROIds)
        {
            var grade = CountGradeForQuestion(questionId, chosenROIds);

            return grade;
        }

        private void SaveFiledAnswer(int userId, int questionId, int testId, string
fileName, string value)
        {
            var generatedFileName = CreateFileName(userId, testId, questionId);
            var fileExtension = Path.GetExtension(fileName);
            var filePath = string.Format(@"..\Diploma\FiledAnswers\{0}{1}",
generatedFileName, fileExtension);

            var spl = value.Split('/')[1];
            var format = spl.Split(';')[0];
            var validBase64String = value.Replace($"",
string.Empty);

            File.Create(filePath).Close();
            File.WriteAllBytes(filePath, Convert.FromBase64String(validBase64String));

            uow.UserAnswers.Create(new UserAnswer
            {
                QuestionId = questionId,
                UserId = userId,
                Value = filePath
            });

            uow.Save();
        }

        private void SaveOpenAnswer(int userId, int questionId, string answerValue)
        {
            var answer = new UserAnswer
            {
                UserId = userId,
                QuestionId = questionId,
                Value = answerValue
            };

            uow.UserAnswers.Create(answer);

            uow.Save();
        }

        private void SaveUsualAnswer(int userId, int questionId, List<int> chosenROIds)
        {
            foreach (var roId in chosenROIds)
            {
                var responseOptionById = uow.ResponseOptions.Get(ro => ro.Id == roId);
                if (responseOptionById != null)
```

```
                {
                    var userAnswer = new UserAnswer
                    {
                        QuestionId = questionId,
                        ResponseOptionId = responseOptionById.Id,
                        UserId = userId,
                        Value = responseOptionById.Value
                    };

                    uow.UserAnswers.Create(userAnswer);
                }
            }
        }

        private string CreateFileName(int userId, int testId, int questionId)
        {
            return string.Format("{0}_{1}_{2}", userId, testId, questionId);
        }

        private decimal CountGradeForQuestion(int questionId, List<int>chosenROIds)
        {
            decimal grade = 0;

            var question = uow.Questions.Get(q => q.Id == questionId);

            if (!question.IsFileQuestion && !question.IsOpenQuestion)
            {
                if(chosenROIds.Count == 0)
                {
                    return grade;
                }
                else
                {
                    foreach (var roId in chosenROIds)
                    {
                        var rightSimpleAnswer = uow.RightSimpleAnswers.Get(rsa =>
rsa.ResponseOptionId == roId);

                        if (rightSimpleAnswer == null)
                        {
                            return grade;
                        }
                        else
                        {
                            grade += rightSimpleAnswer.Grade;
                        }
                    }
                }
            }

            return grade;
        }
    }
}
```

AccountService.cs

```
using DataAccess;
using System;
using DiplomaServices.Models;
using DiplomaServices.Interfaces;

namespace DiplomaServices.Services.AccountManagment
{
```

```
    public class AccountService : IAccountService
    {
        #region private members

        private readonly IUnitOfWork uow;

        private readonly IUserService userService;

        #endregion

        public AccountService(IUnitOfWork uow, IUserService userService)
        {
            this.uow = uow;
            this.userService = userService;
        }
        public int CreateAccount(CreateAccountModel account)
        {
                if (uow.Users.GetSeveral(u => u.Login == account.Login).Count > 0)
                {
                    throw new Exception(string.Format("User with login {0} already exists!",
account.Login));
                }
                else
                {
                    return userService.CreateUser(account);
                }
            }
        }
    }
```

AuthService.cs

```
using DataAccess;
using System;
using DiplomaServices.Models;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;

namespace DiplomaServices.Services.AccountManagment
{
    public class AuthService : IAuthService
    {
        #region

        private readonly IJWTManagmentService jwtService;

        private readonly IUnitOfWork uow;

        private readonly MapperService mapper;

        private readonly PasswordHasher passwordHasher;

        #endregion
        public AuthService(IJWTManagmentService jwtService, IUnitOfWork uow)
        {
            this.jwtService = jwtService;
            this.uow = uow;

            mapper = new MapperService();
            passwordHasher = new PasswordHasher();
        }
        public AuthResponseModel SignIn(SignInModel model)
        {
```

```
            var user = uow.Users.Get(u => u.Login == model.Login);
            if (user == null)
            {
                throw new Exception("User was not found!");
            }
            else
            {
                if (!user.IsEmailVerified)
                {
                    throw new Exception("User email is not verified!");
                }
                else
                {
                    var hashedTrial =
passwordHasher.GetEncodedInfoWithSaltExisting(model.Password, user.Salt);

                    if (hashedTrial != user.Password)
                    {
                        throw new Exception("Password isn't valid! Please, check your
input!");
                    }
                    else
                    {
                        model.Password = user.Password;

                        return jwtService.CreateToken(mapper.Map<SignInModel,
CreateTokenModel>(model));
                    }
                }
            }
        }

        public void SignOut(SignOutModel model)
        {
            jwtService.RemoveToken(mapper.Map<SignOutModel, RemoveTokenModel>(model));
        }

        public string RefreshAccessToken()
        {
            return jwtService.RefreshAccessToken();
        }
    }
}
```

EmailService.cs

```
using DataAccess;
using DiplomaServices.Interfaces;
using System;
using System.Net;
using System.Net.Mail;

namespace DiplomaServices.Services.AccountManagment
{
    public class EmailService : IEmailService
    {
        #region private fields

        private const string SENDER_EMAIL = "testonsender@gmail.com";

        private const string SENDER_PASSWORD = "helloit'sme";

        private readonly IUnitOfWork uow;
```

```
        private string confirmationLink;

        #endregion

        public EmailService(IUnitOfWork uow)
        {
            this.uow = uow;
        }

        public void SendMessage(string applicantEmail, string message, string
confirmationLink)
        {
            SmtpClient smtp = new SmtpClient();
            smtp.Port = 587;
            smtp.Host = "smtp.gmail.com"; //for gmail host
            smtp.EnableSsl = true;
            smtp.UseDefaultCredentials = false;
            smtp.Credentials = new NetworkCredential(SENDER_EMAIL, SENDER_PASSWORD);
            smtp.DeliveryMethod = SmtpDeliveryMethod.Network;

            if (message == null)
            {
                var messageContent = CreateMessage(applicantEmail, null, confirmationLink);
                smtp.Send(messageContent);

                return;
            }
            else
            {
                var messageContent = CreateMessage(applicantEmail, message, null);
                smtp.Send(messageContent);

                return;
            }


        }
        public void SetConfirmationLink(string confirmationLink)
        {
            this.confirmationLink = confirmationLink;
        }
        public void SetEmailAsVerified(int userId)
        {
            var user = uow.Users.Get(u => u.Id == userId);
            if (user != null)
            {
                user.IsEmailVerified = true;

                uow.Users.Update(user);
                uow.Save();
            }
            else
            {
                throw new Exception("user wasn't found!");
            }
        }
        public string GenerateConfirmationToken()
        {
            return Guid.NewGuid().ToString().Replace("-", string.Empty);
        }

        private MailMessage CreateMessage(string applicantEmail, string messageText, string
confirmationLink)
```

```
        {
            MailMessage message = new MailMessage();

            message.From = new MailAddress(SENDER_EMAIL);
            message.To.Add(new MailAddress(applicantEmail));
            message.Subject = "You have one more message from TestOn portal! ";
            message.IsBodyHtml = true; //to make message body as html

            if (messageText == null && confirmationLink != null)
            {
                message.Body = GetConfirmationMessageText(confirmationLink);
            }
            else
            {
                message.Body = messageText;
            }

            return message;
        }
        private string GetConfirmationMessageText(string confirmationLink)
        {
            return string.Format("Hello! Please, tab on this link {0} to verify your email
and then go back to the website. " +
                "Don't share this link with third parties! Have a good day :)",
confirmationLink);
        }
    }
}
```

JWTManagmentService.cs

```
using DataAccess;
using DataAccess.Authentification;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using DiplomaServices.Models;
using DiplomaServices.Interfaces;

namespace DiplomaServices.Services.AccountManagment
{
    public class JWTManagmentService : IJWTManagmentService
    {
        #region private members

        private readonly IUnitOfWork uow;

        #endregion

        public JWTManagmentService(IUnitOfWork uow)
        {
            this.uow = uow;
        }

        public AuthResponseModel CreateToken(CreateTokenModel createTokenModel)
        {
            var identity = GetIdentity(createTokenModel.Login, createTokenModel.Password);

            if (identity == null)
            {
                throw new Exception("Invalid username or password.");
```

```
            }

            var encodedJwt = GenerateAccessToken();

            var user = uow.Users.Get(u => u.Login == identity.Name);

            AuthResponseModel response = new AuthResponseModel
            {
                AccessToken = encodedJwt,
                RefreshToken = GenerateRefreshToken(),
                UserLogin = user.Login,
                UserName = user.Name,
                UserId = user.Id.ToString(),
                IsEmailVerified = user.IsEmailVerified,
                Role = user.Role
            };

            user.AccessToken = response.AccessToken;
            uow.Users.Update(user);
            uow.Save();

            return response;
        }

        public string RefreshAccessToken()
        {
            return GenerateAccessToken();
        }

        public void RemoveToken(RemoveTokenModel removeTokenModel)
        {
            var user = uow.Users.Get(u=>u.Id == removeTokenModel.UserId);
            user.AccessToken = null;
            uow.Users.Update(user);
            uow.Save();
        }

        private ClaimsIdentity GetIdentity(string userName, string password)
        {
            var users = uow.Users.GetSeveral(x => x.Login == userName && x.Password ==
password);
            if (users != null && users.Count < 2)
            {
                var claims = new List<Claim>
                {
                    new Claim(ClaimsIdentity.DefaultNameClaimType, users[0].Login),
                    new Claim(ClaimsIdentity.DefaultRoleClaimType, users[0].Password)
                };
                ClaimsIdentity claimsIdentity =
                new ClaimsIdentity(claims, "Token", ClaimsIdentity.DefaultNameClaimType,
                    ClaimsIdentity.DefaultRoleClaimType);
                return claimsIdentity;
            }
            else
            {
                throw new KeyNotFoundException("User wasn't found!");
            }
        }
        private string GenerateAccessToken()
        {
            var jwt = new JwtSecurityToken(
                   issuer: AuthOptions.ISSUER,
                   audience: AuthOptions.AUDIENCE,
```

```
                    notBefore: DateTime.UtcNow,
                    expires: DateTime.UtcNow.Add(TimeSpan.FromMinutes(AuthOptions.LIFETIME)),
                    signingCredentials: new
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));

            return new JwtSecurityTokenHandler().WriteToken(jwt);
        }
        private string GenerateRefreshToken()
        {
            return Guid.NewGuid().ToString().Replace("-", string.Empty);
        }
    }
}
```

PasswordHasher.cs

```
using Microsoft.AspNetCore.Cryptography.KeyDerivation;
using System;
using System.Security.Cryptography;
using System.Text;
using DiplomaServices.Models;

namespace DiplomaServices.Services.AccountManagment
{
    public class PasswordHasher
    {
        #region

        private string salt;

        #endregion

        public PasswordHashedInfo GetEncodedInfoWithSaltGenerated(string originalPassword)
        {
            GenerateSalt();

            string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
            password: originalPassword,
            salt: Encoding.ASCII.GetBytes(this.salt),
            prf: KeyDerivationPrf.HMACSHA1,
            iterationCount: 10000,
            numBytesRequested: 256 / 8));

            return new PasswordHashedInfo { Salt = salt, PasswordHash = hashed };
        }

        public string GetEncodedInfoWithSaltExisting(string originalPassword, string
saltExisting)
        {
            string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
            password: originalPassword,
            salt: Encoding.ASCII.GetBytes(saltExisting),
            prf: KeyDerivationPrf.HMACSHA1,
            iterationCount: 10000,
            numBytesRequested: 256 / 8));

            return hashed;
        }
        private void GenerateSalt()
        {
            byte[] salt = new byte[128 / 8];
            using (var rng = RandomNumberGenerator.Create())
```

```
            {
                rng.GetBytes(salt);
            }

            this.salt = new string(Encoding.ASCII.GetChars(salt));
        }
    }
                                                    }
```

UserService.cs

```
using DataAccess;
using DataAccess.Entities;
using DiplomaServices.Interfaces;
using DiplomaServices.Mapping;
using DiplomaServices.Models;
using DiplomaServices.Pagination;
using System.Collections.Generic;
using System.Linq;

namespace DiplomaServices.Services.AccountManagment
{
    public class UserService : IUserService
    {
        #region private members

        private readonly IUnitOfWork uow;

        private readonly MapperService mapper;

        private readonly PasswordHasher passwordHasher;

        #endregion

        public UserService(IUnitOfWork uow)
        {
            this.uow = uow;
            mapper = new MapperService();
            passwordHasher = new PasswordHasher();
        }
        public int CreateUser(CreateAccountModel model)
        {
            var user = mapper.Map<CreateAccountModel, User>(model);

            var hashInfo = passwordHasher.GetEncodedInfoWithSaltGenerated(user.Password);

            user.Password = hashInfo.PasswordHash;
            user.Salt = hashInfo.Salt;

            uow.Users.Create(user);
            uow.Save();

            return user.Id;
        }
        public CreateAccountModel GetUserById(int id)
        {
            return mapper.Map<User, CreateAccountModel>(uow.Users.Get(u => u.Id == id));
        }
        public IEnumerable<CreateAccountModel> GetAll()
        {
            return mapper.Map<IQueryable<User>,
IEnumerable<CreateAccountModel>>(uow.Users.GetAll()).ToList();
        }
```

```
        public CheckIfUserExistsResponseModel CheckIfUserExists(string login)
        {
            var user = uow.Users.Get(u => u.Login == login);
            var response = new CheckIfUserExistsResponseModel();
            response.Login = login;

            if (user == null)
            {
                response.ErrorMessage = "User was not found!";
                response.IsFound = false;

                return response;
            }

            if (!user.IsEmailVerified)
            {
                response.ErrorMessage = "User email is not verified!";
                response.IsFound = false;

                return response;
            }

            response.IsFound = true;

            return response;
        }

        public PagedResponse<List<UserModel>> GetUsersPaginated(PaginationFilter filter)
        {
            var totalCount = uow.Users.GetAll().Count();

            var pagedDataFromDb = uow.Users.GetPaginated(filter.PageNumber,
filter.PageSize);
            var pagedDataForResponse = new List<UserModel>();
            foreach (var user in pagedDataFromDb)
            {
                pagedDataForResponse.Add(mapper.Map<User, UserModel>(user));
            }

            return new PagedResponse<List<UserModel>>(pagedDataForResponse,
filter.PageNumber, filter.PageSize, totalCount);
        }
    }
}
```

MapperProfile.cs

```
using AutoMapper;
using DataAccess.Entities;
using DataAccess.Entities.Answers;
using DataAccess.Entities.TestEntities;
using DiplomaServices.Models;

namespace DiplomaServices.Mapping
{
    public class MapperProfile : Profile
    {
        public MapperProfile()
        {
            CreateMap<SignInModel, CreateTokenModel>()
                    .ReverseMap();
```

```
            CreateMap<SignOutModel, RemoveTokenModel>()
                    .ReverseMap();

            CreateMap<CreateAccountModel, User>()
                .ForMember(u => u.Login, cam => cam.MapFrom(src => src.Login))
                .ForMember(u => u.Password, cam => cam.MapFrom(src => src.Password))
                .ForMember(u => u.Name, cam => cam.MapFrom(src => src.Name))
                .ForMember(u => u.Fathername, cam => cam.MapFrom(src => src.Fathername))
                .ForMember(u => u.Surname, cam => cam.MapFrom(src => src.Surname))
                .ForMember(u => u.Role, cam => cam.MapFrom(src => src.Role))
                .ReverseMap();

            CreateMap<CreateCourseModel, Course>()
                .ReverseMap();

            CreateMap<CreateQuestionModel, Question>()
                .ReverseMap();

            CreateMap<CreateResponseOptionModel, ResponseOption>()
                .ReverseMap();

            CreateMap<CreateRightAnswerModel, RightSimpleAnswer>()
                .ReverseMap();

            CreateMap<UserModel, User>()
               .ReverseMap();
        }
    }
}
```

MapperService.cs

```
using AutoMapper;
using System.Linq;

namespace DiplomaServices.Mapping
{
    public class MapperService
    {
        private readonly IMapper mapper;

        public MapperService()
        {
            mapper = ConfigureMapper();
        }

        private IMapper ConfigureMapper()
        {
            var config = new MapperConfiguration(cfg =>
            {
                cfg.AddProfile<MapperProfile>();
            });

            return config.CreateMapper();
        }

        public TDestination Map<TSource, TDestination>(TSource source)
        {
            return mapper.Map<TSource, TDestination>(source);
        }

        public TDestination Map<TSource, TDestination>(TSource source, TDestination
destination)
        {
```

```
            return mapper.Map(source, destination);
        }

        public IQueryable<TDestination> ProjectTo<TDestination>(IQueryable source)
        {
            return mapper.ProjectTo<TDestination>(source);
        }
    }
}
```

AuthFilter.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using System;
using System.Text;

namespace DiplomaAPI.Filters
{
    public class AuthFilter : Attribute, IAuthorizationFilter
    {
        public void OnAuthorization(AuthorizationFilterContext context)
        {
            var accessToken = context.HttpContext.Request.Headers["Authorization"]
                .ToString()
                .Replace("Bearer ", "");
            var accessTokenFromSession = new
string(Encoding.ASCII.GetChars(context.HttpContext.Session.Get("accessToken")));

            if (context.HttpContext.Session.Get("userId") == null ||
                accessTokenFromSession != accessToken)
            {
                context.Result = new UnauthorizedResult();
            }
        }

        public void OnAuthorizationExecuted(AuthorizationFilterContext context)
        {

        }
    }
}
```

UnitOfWork.cs

```
using DataAccess.Repositories;
using System;
using DataAccess.Interfaces.Repositories;

namespace DataAccess
{
    public class UnitOfWork: IUnitOfWork
    {
        #region privateMembers

        private AppContext context;

        private IUserRepository userRepository;
        private ICourseRepository courseRepository;
        private IUserAnswerRepository userAnswersRepository;
        private ITestRepository testRepository;
        private IQuestionRepository questionRepository;
        private IRightSimpleAnswerRepository rightSimpleAnswerRepository;
        private IResponseOptionRepository responseOptionRepository;
```

```
private IUsersCoursesRepository usersCoursesRepository;
private IUsersTestsRepository usersTestsRepository;
private ITestResultRepository testResultRepository;

private bool isDisposed;

#endregion

public IUserRepository Users
{
    get
    {
        if (userRepository == null)
            userRepository = new UserRepository(context);

        return userRepository;
    }
}

public ICourseRepository Courses
{
    get
    {
        if (courseRepository == null)
            courseRepository = new CourseRepository(context);

        return courseRepository;
    }
}

public ITestRepository Tests
{
    get
    {
        if (testRepository == null)
            testRepository = new TestRepository(context);

        return testRepository;
    }
}

public IQuestionRepository Questions
{
    get
    {
        if (questionRepository == null)
            questionRepository = new QuestionRepository(context);

        return questionRepository;
    }
}

public IResponseOptionRepository ResponseOptions
{
    get
    {
        if (responseOptionRepository == null)
            responseOptionRepository = new ResponseOptionRepository(context);

        return responseOptionRepository;
    }
}
```

```
public IRightSimpleAnswerRepository RightSimpleAnswers
{
    get
    {
        if (rightSimpleAnswerRepository == null)
            rightSimpleAnswerRepository = new RightSimpleAnswerRepository(context);

        return rightSimpleAnswerRepository;
    }
}

public IUserAnswerRepository UserAnswers
{
    get
    {
        if (userAnswersRepository == null)
            userAnswersRepository = new UserAnswerRepository(context);

        return userAnswersRepository;
    }
}

public IUsersCoursesRepository UsersCourses
{
    get
    {
        if (usersCoursesRepository == null)
            usersCoursesRepository = new UsersCoursesRepository(context);

        return usersCoursesRepository;
    }
}

public IUsersTestsRepository UsersTests
{
    get
    {
        if (usersTestsRepository == null)
            usersTestsRepository = new UsersTestsRepository(context);

        return usersTestsRepository;
    }
}

public ITestResultRepository TestResults
{
    get
    {
        if (testResultRepository == null)
            testResultRepository = new TestResultRepository(context);

        return testResultRepository;
    }
}

public UnitOfWork(AppContext context)
{
    this.context = context;
    isDisposed = false;
}

public virtual void Dispose(bool disposing)
{
```

```
            if (!isDisposed)
            {
                if (disposing)
                {
                    context.Dispose();
                }

                isDisposed = true;
            }
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        public void Save()
        {
            context.SaveChanges();
        }

    }
}
```

AppContext.cs

```
using DataAccess.Entities;
using DataAccess.Entities.Answers;
using DataAccess.Entities.ManyToManyEntities;
using DataAccess.Entities.TestEntities;
using Microsoft.EntityFrameworkCore;

namespace DataAccess
{
    public class AppContext : DbContext
    {
        #region Db Sets

        public DbSet<User> Users { get; set; }
        public DbSet<Course> Courses { get; set; }
        public DbSet<RightSimpleAnswer> RightSimpleAnswers { get; set; }
        public DbSet<Question> Questions { get; set; }
        public DbSet<UserAnswer> UserAnswers { get; set; }
        public DbSet<ResponseOption> ResponseOptions { get; set; }
        public DbSet<Test> Tests { get; set; }
        public DbSet<UsersCourses> UsersCourses { get; set; }
        public DbSet<UsersTests> UsersTests { get; set; }
        public DbSet<TestResult> TestResults { get; set; }

        #endregion

        public AppContext(DbContextOptions<AppContext> options) : base(options)
        {
            Database.EnsureCreated();
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<UsersCourses>()
            .HasKey(uc => new { uc.CourseId, uc.UserId });

            modelBuilder.Entity<UsersTests>()
            .HasKey(ut => new { ut.TestId, ut.UserId });
        }
```

```
    }
}
```

AuthOptions.cs

```
using Microsoft.IdentityModel.Tokens;
using System.Text;

namespace DataAccess.Authentification
{
    public class AuthOptions
    {
        public const string ISSUER = "TestOnServer"; // издатель токена
        public const string AUDIENCE = "TestOnClient"; // потребитель токена
        const string KEY = "eda9bced-f0f6-44cb-9e5c-ad4751f3428e";   // ключ для шифрации
        public const int LIFETIME = 1; // время жизни токена - 30 минут (6*5)
        public static SymmetricSecurityKey GetSymmetricSecurityKey()
        {
            return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));
        }
    }
}
```

Startup.cs

```
using DataAccess;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.EntityFrameworkCore;
using DataAccess.Repositories;
using AutoMapper;
using FluentValidation.AspNetCore;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using DataAccess.Authentification;
using System;
using DiplomaServices.Mapping;
using DataAccess.Interfaces.Repositories;
using DiplomaServices.Services.AccountManagment;
using DiplomaServices.Interfaces;
using DiplomaServices.Services.TestServices;
using DiplomaServices.Services;
using Microsoft.OpenApi.Models;
using System.Collections.Generic;

namespace Diploma
{
    public class Startup
    {
        private const int SESSION_EXPIRATION_TIME_IN_MINUTES = 60;
        public IConfiguration Configuration { get; }
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }
        public void ConfigureServices(IServiceCollection services)
        {
            string connection = Configuration.GetConnectionString("DefaultConnection");

            services.AddDbContext<DataAccess.AppContext>(options =>
                options.UseSqlServer(connection));
```

```
            services.AddTransient<IUnitOfWork, UnitOfWork>(e => new
UnitOfWork(e.GetService<DataAccess.AppContext>()));
            services.AddTransient<IUserRepository, UserRepository>();
            services.AddTransient<ICourseRepository, CourseRepository>();
            services.AddTransient<IQuestionRepository, QuestionRepository>();
            services.AddTransient<IResponseOptionRepository, ResponseOptionRepository>();
            services.AddTransient<IRightSimpleAnswerRepository,
RightSimpleAnswerRepository>();
            services.AddTransient<ITestRepository, TestRepository>();
            services.AddTransient<IUserAnswerRepository, UserAnswerRepository>();
            services.AddTransient<IUsersCoursesRepository, UsersCoursesRepository>();
            services.AddTransient<IUsersTestsRepository, UsersTestsRepository>();

            services.AddTransient<IAccountService, AccountService>();
            services.AddTransient<IAuthService, AuthService>();
            services.AddTransient<IJWTManagmentService, JWTManagmentService>();
            services.AddTransient<IEmailService, EmailService>();
            services.AddTransient<IUserService, UserService>();
            services.AddTransient<ITestService, TestService>();
            services.AddTransient<ICourseService, CourseService>();
            services.AddTransient<IQuestionService, QuestionService>();
            services.AddTransient<IAnswersService, AnswersService>();

            services.AddMvc()
                    .AddFluentValidation();

            services.AddControllers()
                    .AddNewtonsoftJson();

            var mapperConfig = new MapperConfiguration(mc =>
            {
                mc.AddProfile(new MapperProfile());
            });

            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.AddControllersWithViews()
                    .AddNewtonsoftJson();
```

```
            services.AddSwaggerGen(c =>
            {
                c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
                {
                    Description =
                "JWT Authorization header using the Bearer scheme. \r\n\r\n Enter 'Bearer'
[space] and then your token in the text input below.\r\n\r\nExample: \"Bearer
12345abcdef\"",
                    Name = "Authorization",
                    In = ParameterLocation.Header,
                    Type = SecuritySchemeType.ApiKey,
                    Scheme = "Bearer"
                });

                c.AddSecurityRequirement(new OpenApiSecurityRequirement()
                   {
                     {
                        new OpenApiSecurityScheme
                 {
                           Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                },
                Scheme = "oauth2",
                Name = "Bearer",
                In = ParameterLocation.Header,

            },
                  new List<string>()
                    }
                });
            });
            services.AddSession(options =>
            {
                options.IdleTimeout =
TimeSpan.FromMinutes(SESSION_EXPIRATION_TIME_IN_MINUTES);
            });
        }



        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.UseHttpsRedirection();

            app.UseSwagger(c =>
            {
                c.SerializeAsV2 = true;
            });
            app.UseSwaggerUI(c =>
            {
                c.SwaggerEndpoint("/swagger/v1/swagger.json", "Diploma API");
                c.RoutePrefix = string.Empty;
            });
            app.UseStatusCodePages();
            app.UseRouting();
            app.UseSession();
```

```
            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
            app.UseCors();
        }
    }
}
```

ДОДАТОК Б

Відгук керівника економічного розділу

ДОДАТОК В

Перелік файлів на диску

ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Левдик.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Левдик.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Левдик.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Левдик.ppt	Презентація кваліфікаційної роботи