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 *Шарандо Артема Андрійовича*
(ПІБ)

академічної групи *122-17-2*
(шифр)

спеціальності *122 Комп'ютерні науки та інформаційні технології*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки та інформаційні технології*
(назва освітньої програми)

на тему: *Розробка інформаційної системи для проведення тестування з використанням технологій Spring, Hibernate*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В.*			
розділів:				
спеціальний	*доц. Кабак Л.В.*			
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)

« » _____ 2020 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-17-2* *Шарандо Артема Андрійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інформаційної системи для проведення тестування з використанням технологій Spring, Hibernate*

затверджена наказом ректора НТУ «ДП» від *26.05.2021 р.* № *275-с*

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***	***14.05.2020 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***28.05.2020 р.***

Завдання видав _____ *доц. Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

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

Дата видачі завдання: *10.02.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *03.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 84 с., 51 рис., 1 табл., 1 додатків, 19 джерел.

Об'єкт розробки: інформаційна система для проведення тестування.

Мета дипломного проекту: автоматизація інформаційної системи для проведення тестування.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми,уточнюється постановка завдання.

Перший розділ присвячено аналізу предметної області, визначенню актуальності завдання та призначення інформаційної системи, розроблена постановка завдання, технологій та програмних засобів.

Другий розділ включає у себе виконання аналізу існуючих рішень, обрання платформи для розробки, проектування та розробку програми, наведення опис алгоритму і структури функціонування програми, наведення характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.Практичне значення полягає у створенні додатку, що дозволить.

Актуальність даного програмного продукту визначається кількістю створених акаунтів та пройдених тестів.

Список ключових слів: ВЕБ-ДОДАТОК, ІНТЕРФЕЙС, ПРОГРАМА, ТЕСТ, КОРИСТУВАЧ.

ABSTRACT

Explanatory note: 84 p., 51 figs., 1 tabl., 1 add., 19 sources.

Development object: virtual digital synthesizer as a standalone application and as a digital audio workstation plugin.

The purpose of the diploma project: automation of information system for testing.

The introduction examines the current state of the problem, specifies the purpose of the qualification work, relevance and the field of its application, specifies the task statement.

In the first section, the analysis of the subject area was carried out, the relevance of the task and the designation of the development was determined, the task statement was developed, the requirements for the software implementation, the technologies and software tools were developed.

The second section provides: the purpose of the program, the description of the applied mathematical methods, the description of the used technologies and programming languages, description of the structure of the program and algorithms of its operation, and detailed description of the work of the developed software product.

In the economic section the complexity of the developed information system is determined, the calculation of the cost of work on the creation of the program is calculated and the time for its creation is calculated.

The practical value is the creation of an application that provides the ability to create music content.

The relevance of the information system is determined by the number of created accounts and passed tests.

Keywords: WEB APPLICATION, INTERFACE, PROGRAM, TEST, USER.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

СКБД – система управління базами даних;

БД – база даних;

API – інтерфейс програмування додатків;

JPA – Java Persistence API;

SQL – Structured Query Language;

UI – інтерфейс користувача;

ІС – інформаційна система;

CSS – Cascading Style Sheets;

HTML – HyperText Markup Language;

JSP – Java Server Pages;

MVC – Модель-Вид-Контролер;

ID – ідентифікатор;

HTTP – HyperText Transfer Protocol;

REST – Representational State Transfer;

JS – JavaScript.

PHP – Hypertext Preprocessor.

Зміст

ВСТУП

У тепершній час web-технології дозволяють створювати велику кількість проектів з різноманітними рішеннями. Кількість та швидкість оновлення технологій дає можливість постійно впроваджувати нові можливості до проекту та куди-більш краще рішення, ніж вони були.

Незважаючи на те, що розроблена достатня кількість програмних продуктів, що дозволяють автоматизувати процес тестування студентів, багато з них мають недоліками, або зайвою функціональністю. Розробка нового продукту, орієнтованого на конкретного користувача, є важливим і актуальним завданням У цьому випускний кваліфікаційної роботи розробляється система автоматизації тестування знань студентів, підсистема студента. Впровадження даної системи дозволити заощаджувати час викладачів і студентів, а також призведе до більш об'єктивного оцінювання знань.

Метою дипломного проекту є створення веб-додатку, який давав би змогу звичайному користувачеві проходити тести різного типу, складності, переглядати результати пройдених тестів у вигляді відсотків.

Беручи це все до уваги було сформовано тему дипломної роботи: «Розробка інформаційної системи для проведення тестування з використанням технологій Spring, Hibernate»

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості с предметної галузі

Предметною областю є тестування знань. Тестування знань призначене насамперед для вирішення завдань діагностики і навчання. Діагностична функція полягає в оцінки знань користувача [1]. Це є основна функція данної предметної області. Навчальна функція полягає в мотивації користувача до активізації роботи по засвоєнню навчального матеріалу. Серед ознаків тестування, як методу, можна виділити такі позитивні риси [1]:

- стандартизована процедура проведення;
- справедливість як в процесі контролю, так і в процесі оцінки, практично виключаючи суб'єктивізм викладача.
- можливість оцінювати знання по всьому курсу в цілому, на відміну від усного та письмового іспиту;
- точність в оцінюванні знань, результат складається залежно від кількості питань та відповідей на них;
- економія, як в плані коштів, так і в плані часу;
- при оцінюванні не витрачаются ресурси;
- мінімізація можливості нечесної демонстрації знань;
- Мінімізація отримання хибного результату, через демонстрацію нечесних знань.

В даний час існує безліч систем, призначених для автоматизації тестування знань студентів, як платних, так і безкоштовних.

Розглянемо програми, які використовують метод тестування для навчання. У таблиці наведено порівняльну характеристику автоматизованих систем данної системи.

№	Назва проекту	Робота через інтернет	Платформа	Ліцензія
1	Система «Синтез»	+	PHP, MySQL	Платна
2	«Конструктор тестів»	-	C++	Безкоштовна
3	«OpenTest 2.0»	+	HTML, PHP, JavaScript	Платна

В ході аналізу програм аналогів були виявлені недоліки у перерахованих вище систем тестування, а саме функціональна перевантаженість, платна ліцензія, а також не всі системи є масштабованими і не працюють через інтернет.

В ході аналізу перерахованих вище систем тестування аналогів, були виявленні такі недоліки, як:

- функціональна перезавантаженність;
- платна ліцензія;
- не всі системи працюють через інтернет;
- не адаптовані до масштабу.

Таким чином можемо зробити висновок, що розробка автоматизовану систему тестування, позбавлену виділених у програм аналогів недоліків є доцільною.

1.2 Призначення розробки та область застосування

Темою бакалаврської дипломної роботи виступає: «Розробка інформаційної системи для проведення тестування з використанням технологій Spring, Hibernate». Головною метою роботи є створення веб-сайту, який представляє собою інструмент для вирішення завдань діагностики і навчання, методом автоматизованої системи тестування [2]

Головними критеріями розроблювального веб-додатку є:

- ✓ Зручність в використанні;
- ✓ Максимальна доступність для користувача;
- ✓ Легкість в освоєнні.

Система призначена для:

- ✓ Створення інструменту тестування та діагностики;
- ✓ Використання автоматизованого тестування для навчання та діагностики.

Система позиціонується як веб-додаток, який дає можливість використовувати ресурси тестування для покращення, оцінки та діагностики своїх знань.

1.3 Підстава для розробки

Відповідно до ОКХ та ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу

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

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

- ОКХ та ОПП за напрямом підготовки «Комп'ютерні науки»;
- Графік навчального процесу та навчальний план;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № _____ від __.__.2020 р;
- завдання на дипломний проект на тему «Розробка інформаційної системи для проведення тестування з використанням технологій Spring, Hibernate»

1.4 Постановка завдання

Даний програмний продукт призначений для автоматизованого тестування знань. Повинні бути присутніми наступні можливості: розмежування рівня прав між адміністратором та користувачем [3]. Що до першої ролі нашої системи, повинні бути такі пункти, як:

- створення, видалення, редагування тесту, доступ до списку всіх користувачів;
- редагування персональної інформації користувача;
- мати можливість змінити статус користaвача на неактивний з метою запобігання подальшого використання ресурсу тестування;
- перегляд результату проходження тестів для побудови статистики та діагностики.

Користувач буде мати такі можливості, як:

- перегляд списку ресурсів тестуання за допомогою пошуку за крітеріями тесту;
- проходження тесту за визначенним часом тесту;
- перегляд інформації, яка була вказана користувачем при регистрації на ресурсі тестувальної системи та результат проходження тестів;
- змінити мову в ході користуванням веб-додатку

1.5 Вимоги до програми або програмного виробу

1.5.1 Вимоги до функціональних характеристик

Вимоги до програми – це проектування автоматизованої інформаційної системи для проведення тестування:

- відображення інформації на сайті зчитаної з таблиць СКБД;
- збереження інформації вказаної користувачем;
- пошук ресурсів тестування за крітеріями;
- підтримка web-браузерів та доступ до програми через нього.

Рівнем доступу будуть виступати:

- адміністратор
- користувач

1.5.2 Вимоги до інформаційної безпеки

Для використання програми потрібно реалізувати такі пункти, як:

- можливість редагувати данні;
- контроль вхідних та вихідних данних;
- захист від надання ресурсу незареєстрованим користувачам;

- збереження цілістності данних;
- блокування користувача з метою запобігання подальшого використання ресурсу тестування.

1.5.3 Вимоги до складу та параметрів технічних засобів

Для забеспечення надійного функіоналу програмного забеспечення, обчислювальна машина, на якій буде експлуатуватися веб-додаток, повинна мати такі мінімальні характеристики:

- процесор класу Intel Core i3 2 ядра 3,8Ггц;
- монітор;
- 300Мб вільного місця на диску;
- клавіатура;
- маніпулятор «миша»;
- не менше 1Гб ОЗУ.

1.5.4 Вимоги до інформаційної та програмної сумісності

Для інформаційної та програмної сумісності необхідна наявність наступних програм та систем:

- операційна система Microsoft Windows 7/8/10;
- Web-браузер Google Chrome, Firefox, Opera.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Функціональне призначення веб додатку полягає для вирішення завдань діагностики і навчання [4]. Розроблений в ході кваліфікаційної роботи веб-додаток має наступні функції та призначення:

- розділення функціоналу адміністратора та користувача
- перегляд вже існуючих тестів для подальшого проходження
- система блокування адміністратором активних користувачів
- сортування тестів за параметрами (складність, кількість питань, предметна область)
- зручний та зрозумілий інтерфейс регістрації та логіну
- перегляд користувачем свого профілю з його інформацією та результатами проходження раніше пройдених тестів
- зміна мови інтерфейса користувача

Головним вікном є сторінка для регестрації користувача та логіну вже інсуючих. Після проходження верифікації аккаунту, користувач має доступ до свого профілю, перелік тестів для подальшого проходження.

2.2. Опис застосованих математичних методів

Особливості предметної області інформаційної системи, не передбачає використання метаматичних методів, при розробці веб-додатку для вирішення завдань діагностики і навчання за допомогою тестування.

2.3. Опис використаної архітектури та шаблонів проектування

При розробці архітектури даної програми використовувалося середовище Intellij Idea Ultimate Edition [5].

В якості основних фреймворків використовується Spring Boot та Hibernate. Spring Boot – це один з найпопулярніших фреймворків, для створення додатків на основі Spring. Він дозволяє створювати веб-додаток, з мінімальними налаштуваннями та кількістю кода [6].

Hibernate - бібліотека для мови програмування Java, призначена для вирішення завдань об'єктно-реляційного відображення (ORM). Дозволяє скоротити обсяги низкоуровневого програмування при роботі з реляційними базами даних; може використовуватися як в процесі проектування системи класів і таблиць «з нуля», так і для роботи з уже існуючою базою [7].

Додатково використовується бібліотека Lombok [8] - проект по додаванню додаткової функціональності в Java за допомогою зміни вихідного коду перед Java компіляцією.

Maven - інструмент для автоматизації збирання проектів. З ним працюють в основному Java-розробники, хоча є плагіни для інтеграції з C / C ++, Ruby, Scala, PHP і іншими мовами [9].

JavaScript - мова програмування [10], яка дозволяє вам створити динамічно оновлюваний контент, управляє мультимедіа, анімує зображення. Java Server Pages представляє технологію [11], яка дозволяє створювати динамічні веб-сторінки. Спочатку JSP (разом з Сервлетами) на зорі розвитку Java EE були домінуючим підходом до веб-розробки на мові Java. І хоча в даний час вони поступилося своїм місцем іншій технології - JSF, проте JSP продовжують широко використовуватися.

2.4. Опис структури системи та алгоритмів її функціонування

Структура диплома сформована за стандартною структурою веб-додатку на Java, з використанням фреймворку Spring-boot [12]. Для налаштування присутні файли фреймворку, які дозволяють зібрати програму в тому чи іншому форматі, а також окремі пакети для структурованності, в яких зберігаються класи для створення додатку. На рис. 2.1 наведена структура файлів проекту.

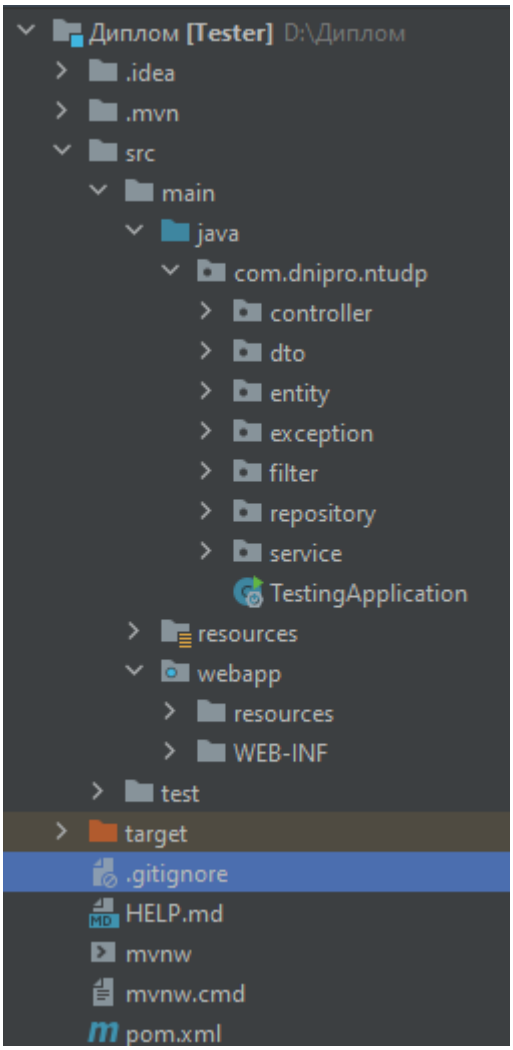

Рис 2.1

Опис файлів та структури проекту:

Controller - це клас, призначений для безпосередньої обробки запитів від клієнта і повернення результатів. На рис. 2.2 наведені контолери, які використовуються для відправки та отримання даних через інтерфейс користувача.

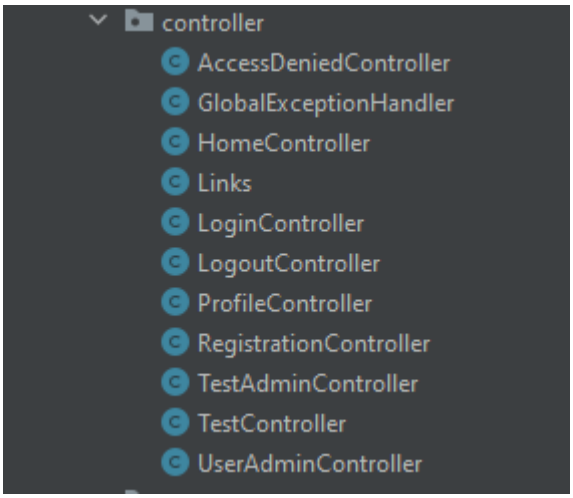

Рис 2.2

DTO (Data Transfer Object) - один із шаблонів проектування, що використовується для передачі даних між прикладними підсистемами. На рис. 2.3 наведені класи, які використовуються для отримання та передачі данних між інтерфейсом користувача та контролером.

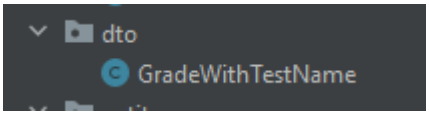

Рис 2.3

Entity - це легковесний зберігаємий об'єкт бізнес-логіки (постійний об'єкт домену). Основна програмна сутність цього класу - entity, який так само може використовувати додаткові класи, які можуть бути використані як вспомогальні класи або для збереження стану сутності (рис. 2.4).

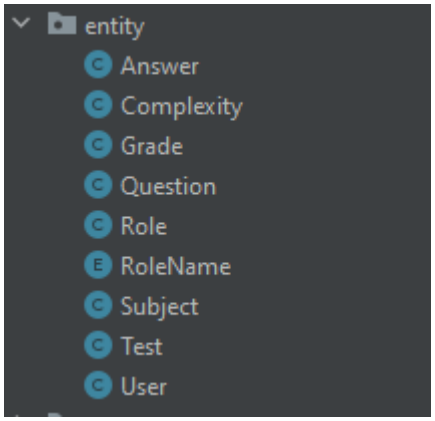

Рис 2.4

Exception - це проблема (помилка), яка призводить до завершення виконання програми. Виключення можуть бути пов'язані з багатьма випадками. Тому були сформовані кастомні Exception, для розуміння, яка помилка виникла в окремому випадку. На рис. 2.5 відображенні кастомні помилки, які були сформовані в ході аналізу інформаційної структури.

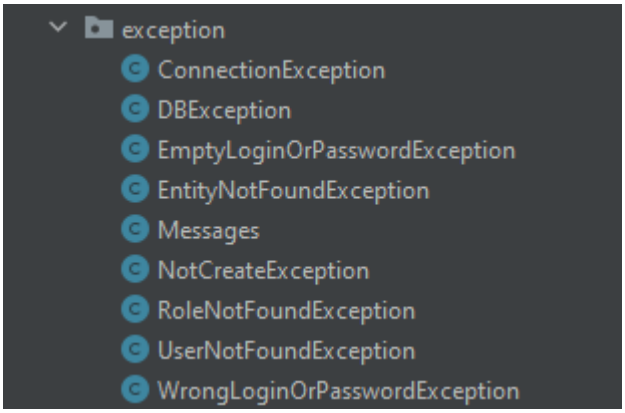

Рис 2.5

Filter - фільтр, відповідно до специфікації, це Java-код, придатний для повторного використання і який дозволяє перетворити зміст HTTP-запитів, HTTP-відповідей і інформацію, що міститься в заголовках HTML. Сервлетний фільтр займається попередньою обробкою запиту, перш ніж той потрапляє в

сервлет або наступною обробкою відповіді, що виходить з сервлета. На рис. 2.6 зображені власно виробленні фільтри.

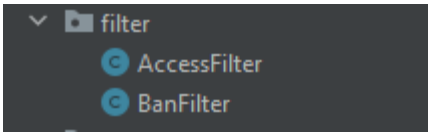

Рис 2.6

Repository - це інтерфейс фреймворка Spring Data надає набір стандартних методів JPA для роботи з БД. Завдяки інтерфейсу JPA, багато примітивних запросів до бази даних, вже імплементовані (влаштовані) в цей інтефейс. На основі наших entity, були сформовані ці інтерфейси для отримання данних з сутності бази данних.

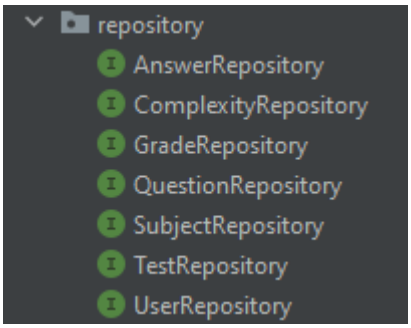

Рис 2.7

Service – це клас, призначений для роботи з даними між БД та контролером та додаванням деякої бізнес-логіки. Іншими словами, це обгортка з бізнес процесами нашого repository, який працює лише з базою даних. Для опису поведінки наших класів сервісів, було сформовано інтерфейси з сигнатурою метода, яка відображає, що має роботи той чи інший метод.
На основі цих інтерфейсів, було прописано імплементацію (реалізацію) сервісів, які вже мають чіткий алгоритм виконання методів, для подальшого

використання їх у контролерах. На рис. 2.8 представлені всі інтерфейси та імплементації сервісів.

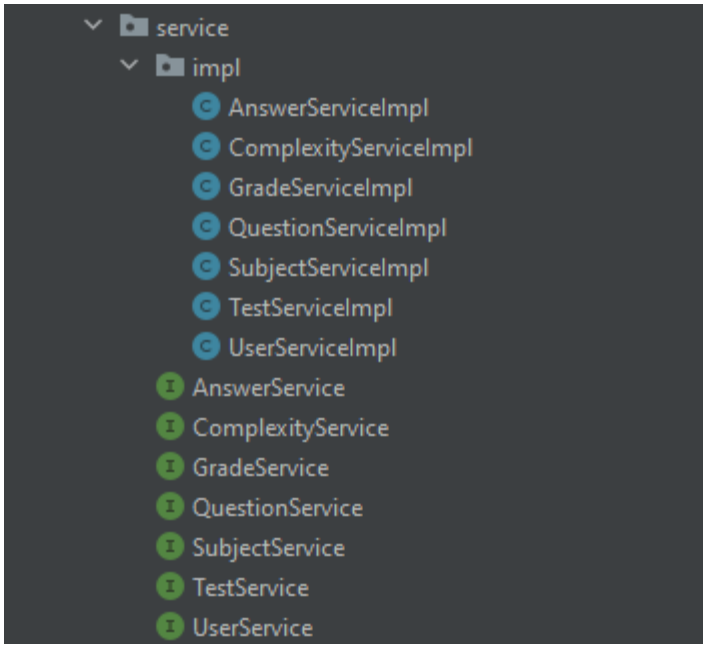

Рис 2.8

TestingAplication – наш головний клас, який відповідає за сбірку нашого додатку та запуску. Насамперед наш клас запуску дивиться налаштування додатку, які ми вказали у файлі application.properties та намагається збудувати проект. Саме завдяки фреймворку Spring-Boot і її анотаціям (Component, Repository, Service і тд.) наш класс запуску розуміє які класи для чого були створені, формує її у правильній послідовності, та в цілому контролює процес створення та знищення наших класів. На рис. 2.9 відображено як цей клас виглядає у структурі проекту.

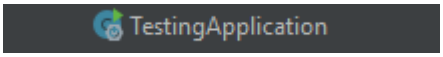

Рис 2.9

resources – це програмні ресурси нашого веб-додотку, які потрібні для

його виконання. Наприклад в файлі application.properties описані всі налаштування програми, веб інформації, бази даних. data.sql – це файл, який виконується при запуску веб-додатку і якщо деяка інформація відсутня в БД, то спрацює скрипт, який відновить всю потрібну інформацію, для подальшого використання її у ході процесса використання програми користувачем. Також веб-додаток має локалізацію і можна змінити мову в ході використання веб-додатку, за англійскої мови на російську та навпаки. Саме файли ресурсу дозволяють реалізувати локалізацію веб-додатку, де прописані обидва варіанти можливості відображення інформації залежно від обраної мови користувачем програми. Вся структура ресурсних файлів відображена на рис. 2.10.

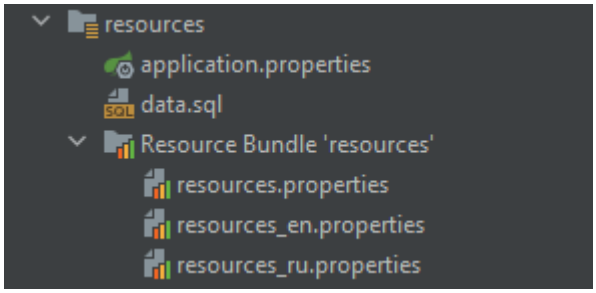

Рис 2.10

webapp – це пакет, в якому збережені файли, які потрібні для зміни вигляду, додавання функціоналу відображення до веб-додатку. На рис. 2.11 відображено структура, де зберігаються картинки, стилі сайту та декілька функцій написаних на JavaScript.

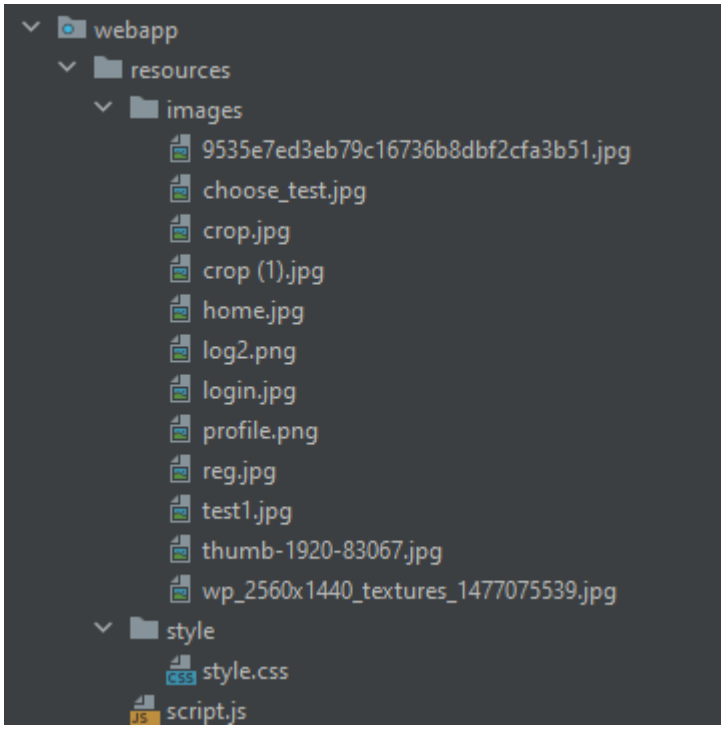

Рис 2.11

WEB-INF - спеціальний каталог, який містить усі речі, пов’язані з програмою, яких немає в корені документа програми. Вузол WEB-INF не є частиною загальнодоступного дерева документів програми. У цьому пакеті збережені файли формату JSP, тобто файли відображення елементів на сторинках додатку.

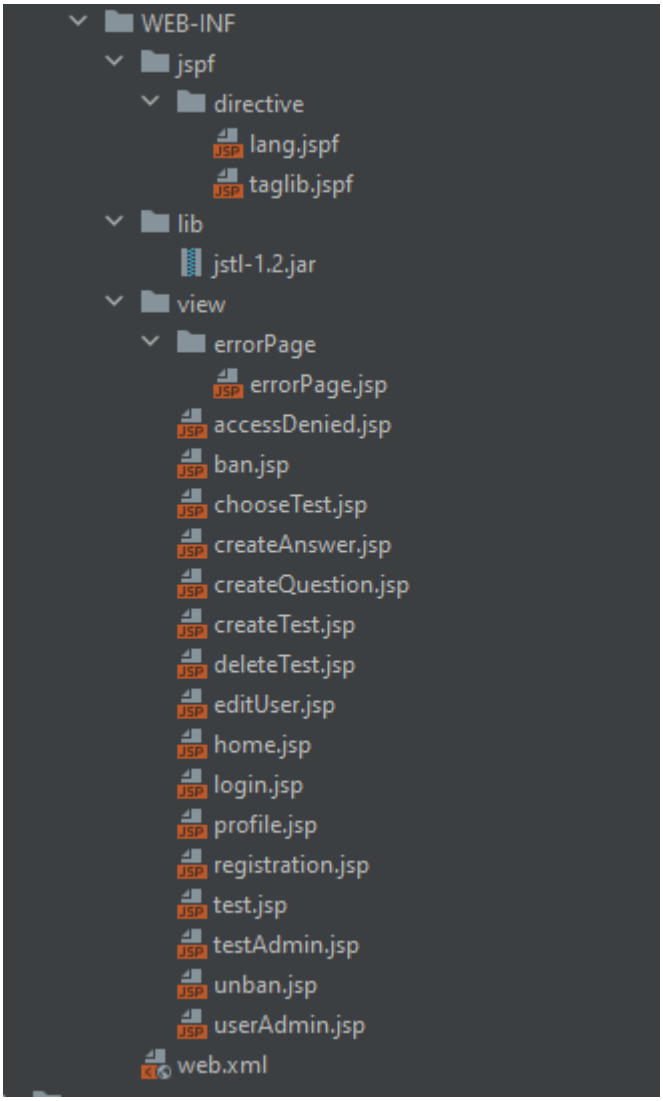

Рис 2.12

В ході аналізу інформаційної системи для проведення тестування, було принято рішеня використовувати MySQL – (СКБД) вільна система керування реляційними базами даних [15]. Після аналізу предметної області, були сформовані сутності в БД та відносини між ними [13-14]. Загальна структура бази даних зображена на рис. 2.13.

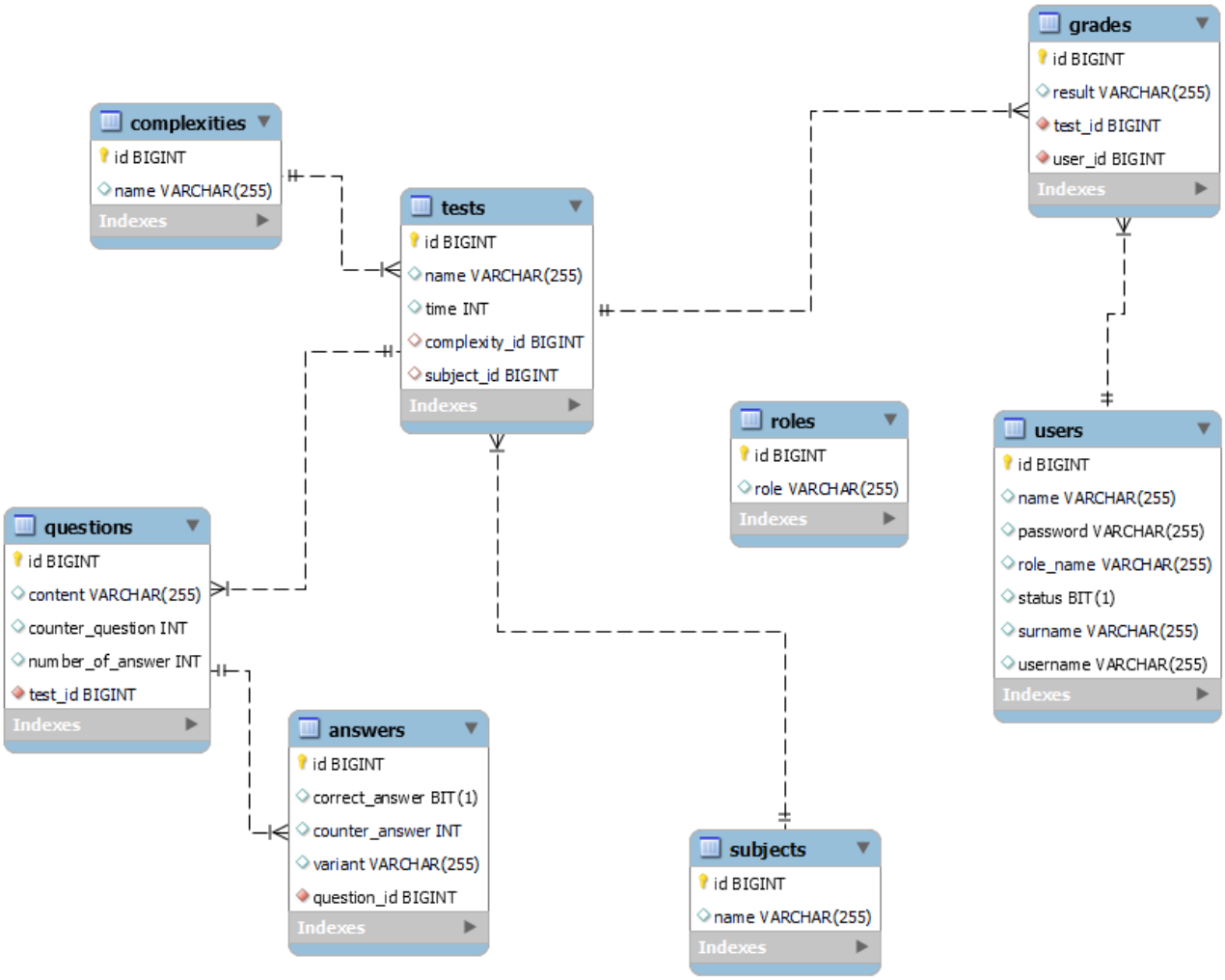

Рис 2.13

Структура сутностей бази даних:

Users (рис. 2.14) – таблиця користувачів, яка частково формується з даних вказаними користувачем при реєстрації. При формуванні об’єкту користувача, присвоюється роль user та status – 1 (користувач зараз активний та не блокований адміністратором)

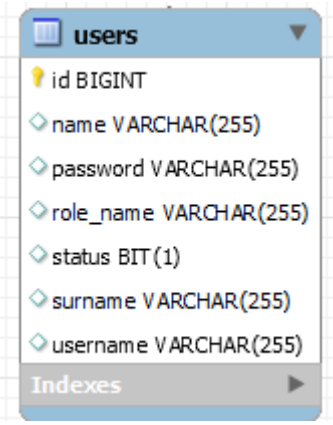

Рис 2.14

Roles (рис. 2.15) – таблиця можливих ролей користувачів веб-додатку. Зараз при запуску програми, в базу даних записується дві ролі: user, admin.

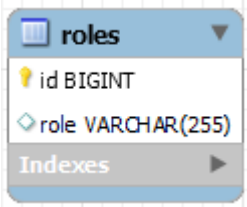

Рис 2.15

Tests (рис. 2.16) – таблиця в якій зберігається загальні відомості тесту для проходження.

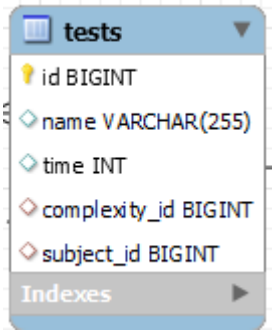

Рис 2.16

Complexities (рис. 2.17) - таблиця яка зберігає можливі ступеня складності тесту. Співвідношення до таблиці tests – OneToMany. В базі даних збережено 3 складності: Easy, Medium, Hard.

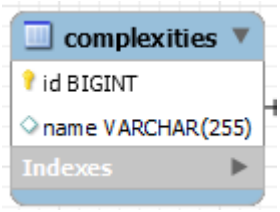

Рис 2.17

Subjects (рис. 2.18) – таблиця яка зберігає предметну область тесту. Співвідношення до таблиці tests – OneToMany. В базі даних збережено 4 предметної області: Math, English, Programming, Ukrainian.

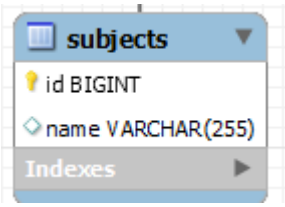

Рис 2.18

Questions (рис. 2.19) - таблиця яка зберігає список питань, підв'язані до тесту. Співвідношення до таблиці tests – ManyToOne, тобто один тест зберігає список питань.

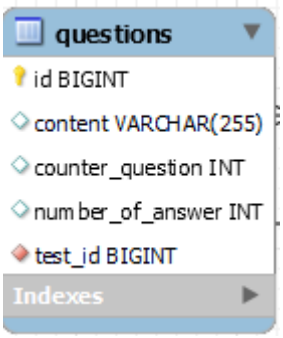

Рис 2.19

Answers (рис. 2.20) - таблиця яка зберігає список відповідей, підв'язані до питання. Співвідношення до таблиці questions – ManyToOne, тобто одне питання зберігає список відповідей.

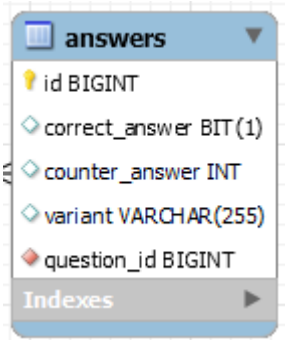

Рис 2.20

Grades (рис. 2.21) – таблиця яка зберігає результат проходження тесту користувачем. Співвідношення до таблиці tests – ManyToOne, до таблиці users – ManyToOne. Тобто у одного користувача до кожного тесту є свій результат.

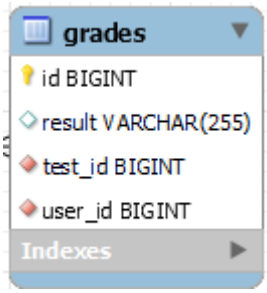

Рис 2.21

Загальний алгоритм роботи веб-додатку інформаційної системи для проведення тестування зображений на рис. 2.22.

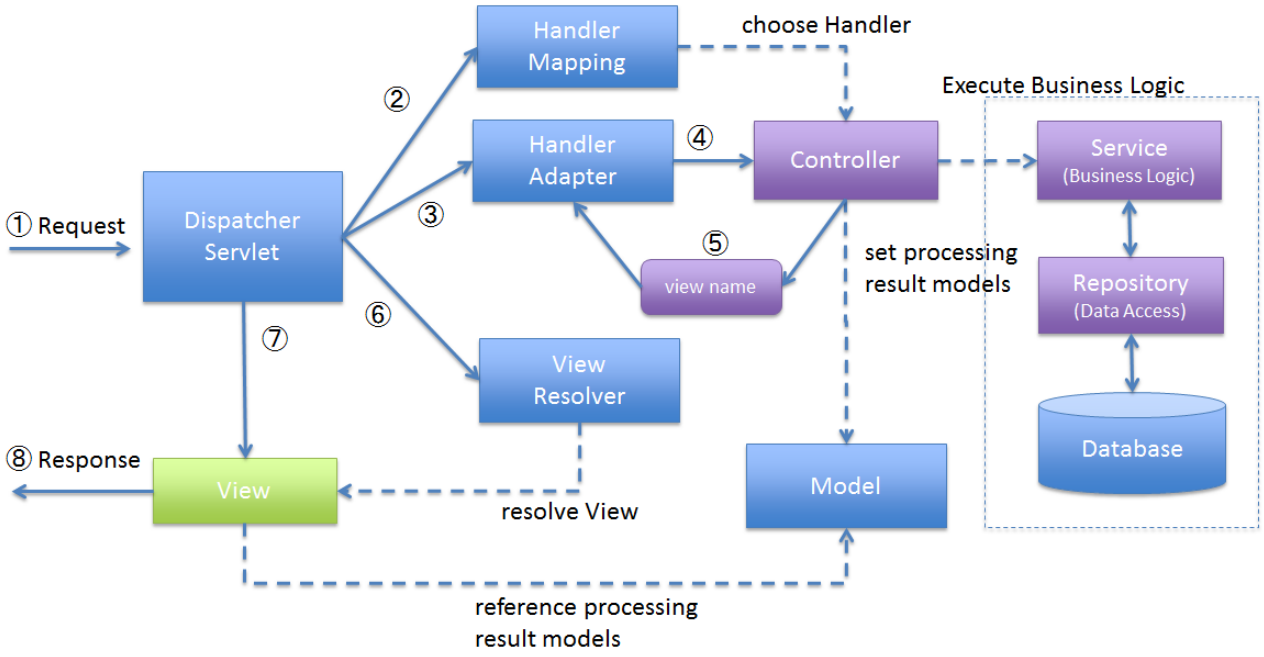

Рис. 2.22

Вся логіка роботи Spring MVC побудована навколо DispatcherServlet, який приймає і обробляє всі HTTP-запити (з UI) і відповіді на них [16]. Нижче наведена послідовність подій, відповідна вхідному HTTP-запиту:

- Після отримання HTTP-запиту, DispatcherServlet звертається до інтерфейсу HandlerMapping, який визначає, який Контролер повинен бути

викликаний, після чого, відправляє запит в потрібний Контролер. Контролер приймає запит і викликає відповідний службовий метод, заснований на GET або POST. Викликаний метод визначає дані Моделі, засновані на певній бізнес-логікою і повертає в DispatcherServlet ім'я (View);

- За допомогою інтерфейсу ViewResolver DispatcherServlet визначає, який Вид потрібно використовувати на підставі отриманого імені;
- Після того, як Вид (View) створений, DispatcherServlet відправляє дані Моделі у вигляді атрибутів в Вид, який в кінцевому підсумку відображається в браузері.

Всі вищезгадані компоненти [17], а саме, HandlerMapping, Controller і ViewResolver, є частинами інтерфейсу WebApplicationContext extends ApplicationContext, з деякими додатковими особливостями, необхідними для створення web-додатків.

2.5. Обгрунтування та організація вхідних та вихідних даних програми

В якості вхідних даних програмного комплексу є дані:

- отримані через інтерфейс користувача (логін, пароль, ім'я, прізвище, відповіді до тесту і тд.);
- отримані через інтерфейс адміністратора (вся інформація теста, змінена інформація користувача і тд.);
- отримані з бази даних, які були вже створенні при збірці веб-додатку (адміністратор, ролі, складності тесту, предметна область тесту);

Вихідними даними є сторінка з відображеною інформацією, які були

витягнуті з бази даних, трансформована додатком у той вигляд, яка вона повина буди на той чи іншої сторінці користувача.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Веб-додаток може використовуватись на різних пристроях, де є можливість використання браузеру. Але на цей час, програма на локальному хостінгу та не адаптивна до всіх пристроїв, крім: персонального компьютеру, ноутбуку.

Для користування потрібно мати один з цих браузерів: Google Chrome, Mozilla, Opera, Edge і тд. До параметрів пристрою, додадок не є вибагливим.

2.6.2. Використані програмні засоби

Для роботи програми необхідні такі програмні засоби:

- встановлене середовище виконання Intellij Idea;
- встановлена система керування базами даних MySQL Server.

2.6.3. Виклик та завантаження програми

Для виклику програми потрібно запустити файл TestingApplication, після того як веб-додаток буде разгорнуто, потрібно відкрити браузер та прописати у графі пошук такий URL: http://localhost:8082. Після того, як користувач буде перенаправлений до сайту, потрібно зареєструватися у системі за допомогою

полів вводу даних реєстрації. Далі натиснути кнопку Login та ввести дані, які були вказані при реєстрації раніше користувачем.

2.6.4. Опис інтерфейсу користувача

Робота з програмою починається з головної сторінки за адресою http://localhost:8082/ (рис. 2.23). Користувач має можливість створити аккаунт або увійти в вже створений обліковий запис. Для того щоб зареєструвати свій аккаунт, клінт має натиснути кнопку «Регистрация» (рис. 2.24).

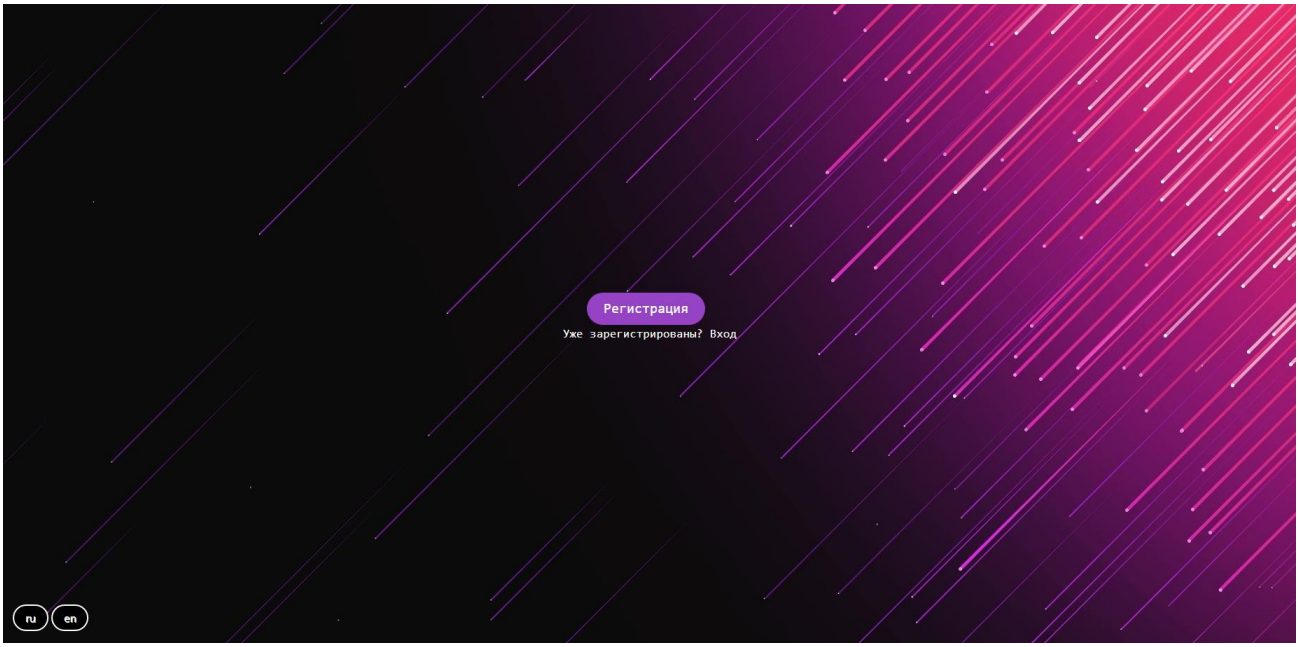

Рис. 2.23

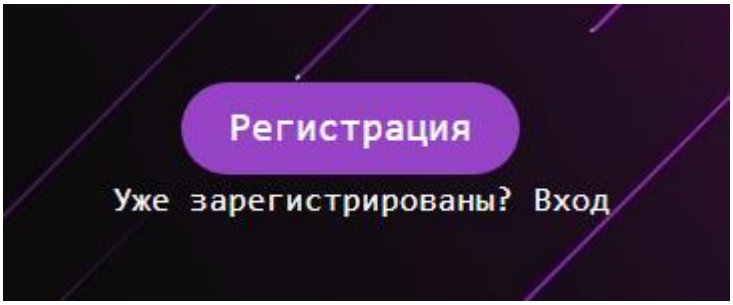

Рис. 2.24

Для того щоб змінити мову використання програмного забеспечення, потрібно натиснути на клавішу обраної мови: «en» (English) або «ru» (Російська) (Рис 2.25).

Рис. 2.25

На рис. 2.26 зображена стартова сторінка, зміненої локалізації на анлійську мову.

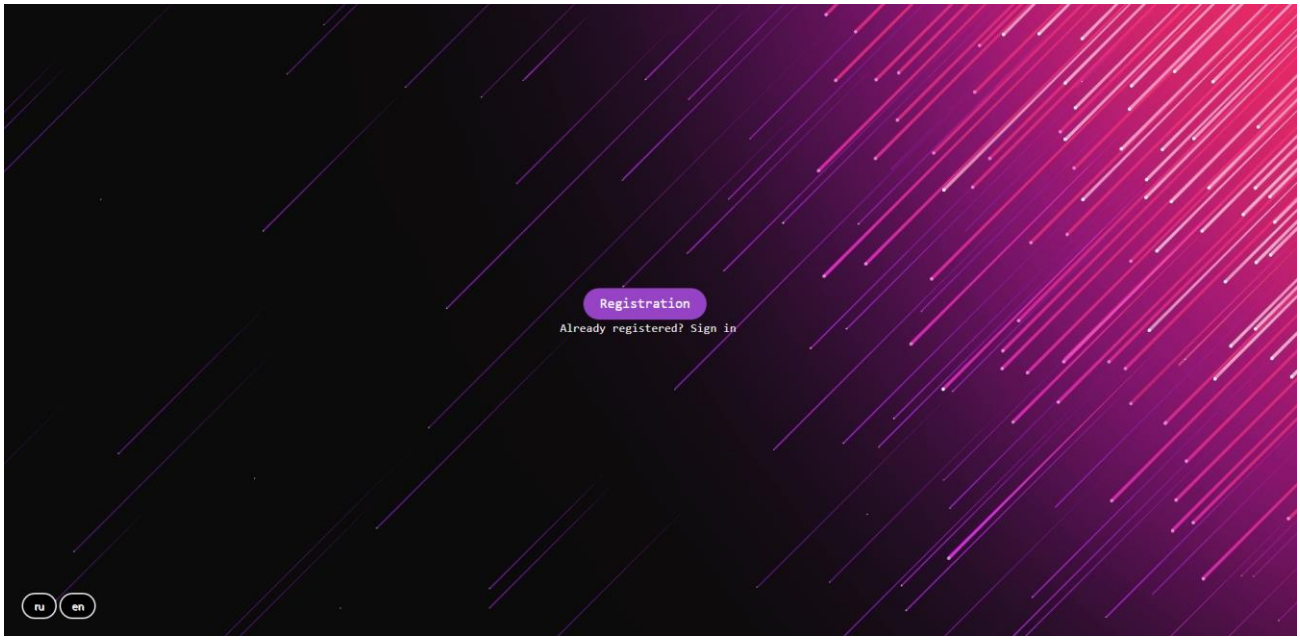

Рис. 2.26

Якщо користувач натискає зареєструватися, він переходить на сторінку, де йому потрібно ввести деяки дані для подальшого їх використання в аутентифікації облікового запису. На рис. 2.27 представлена сторінка, з такими полями для вводу даних:

- ім’я користувача;
- прізвище;
- логін (username);
- пароль.

Потрібно також зазначити, що логін користувача має бути унікальним, за для унікальності аккаунта.

Після того як клінт виконав всі вимоги реєстрації, потрібно натиснути кнопку «Зарегестрироваться» або «Registration», залежно від вибраної мови клієнтом. Всі дані будуть передані до контролера, який в свою чергу запише всю інформацію до бази даних в таблицю users, де зберігаються всі данні облікових записів користувачів.

Рис. 2.27

Якщо обліковий запис користувача вже створений, потрібно натиснути кнопку «Вход» (Login) (рис. 2.28). У подальших діях, клієнт має ввести свої дані, котрі вказував при формуванні свого облікового запису у поля логіну та паролю. Після того як умови було виконаті, користувач має натиснути увійти. Програма зчитує вказані дані з полів, передає до контролеру, який в свою чергу намагається перевірити за логіном, чи інсує такий користувач. Якщо алгоритм знаходить такого користувача, то перевіряється пароль котрий був введений клієнтом та отриманий з раніше знайденого користувача у базі на рівність.

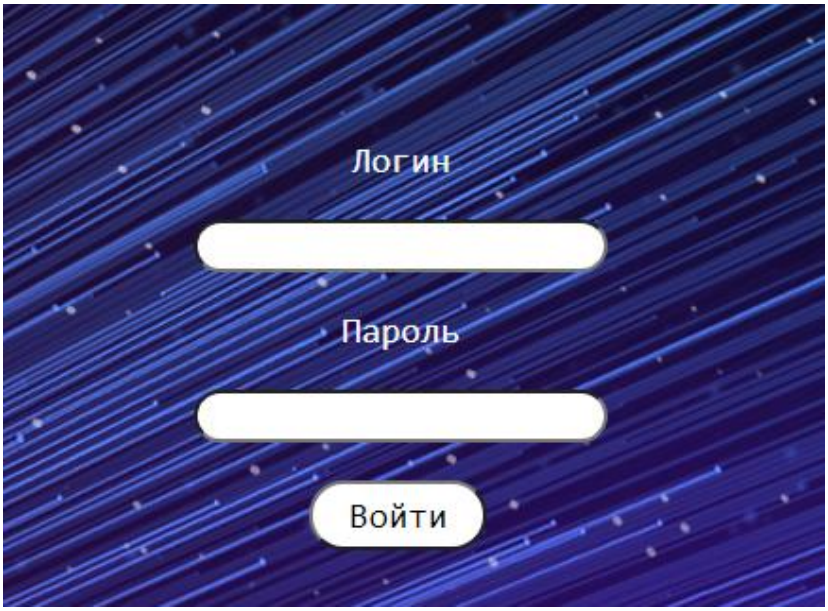

Рис.2.28

Все ж якщо умови не були виконаті коректно, користувач буде повідомлений про це повідомленням, залежно від помилки. Наприклад некоректний логін або пароль (рис. 2.29).

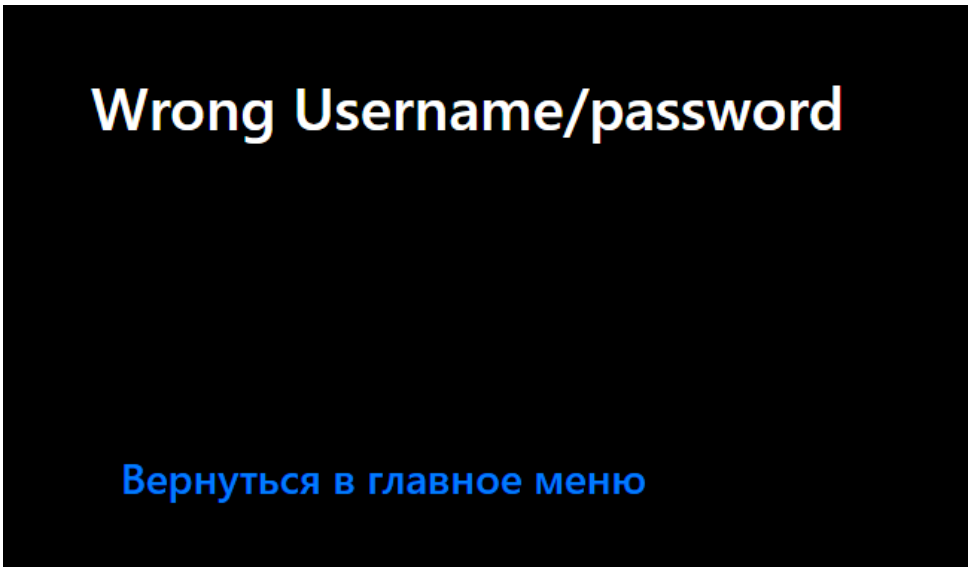

Рис. 2.29

Після того як користувач заходить у свій аккаунт, він переходить до головної сторінки (рис. 2.30), де він має можливість перейти до проходження тесту або ж перейти до свого профілю натиснувши іконку профілю на сторінці.

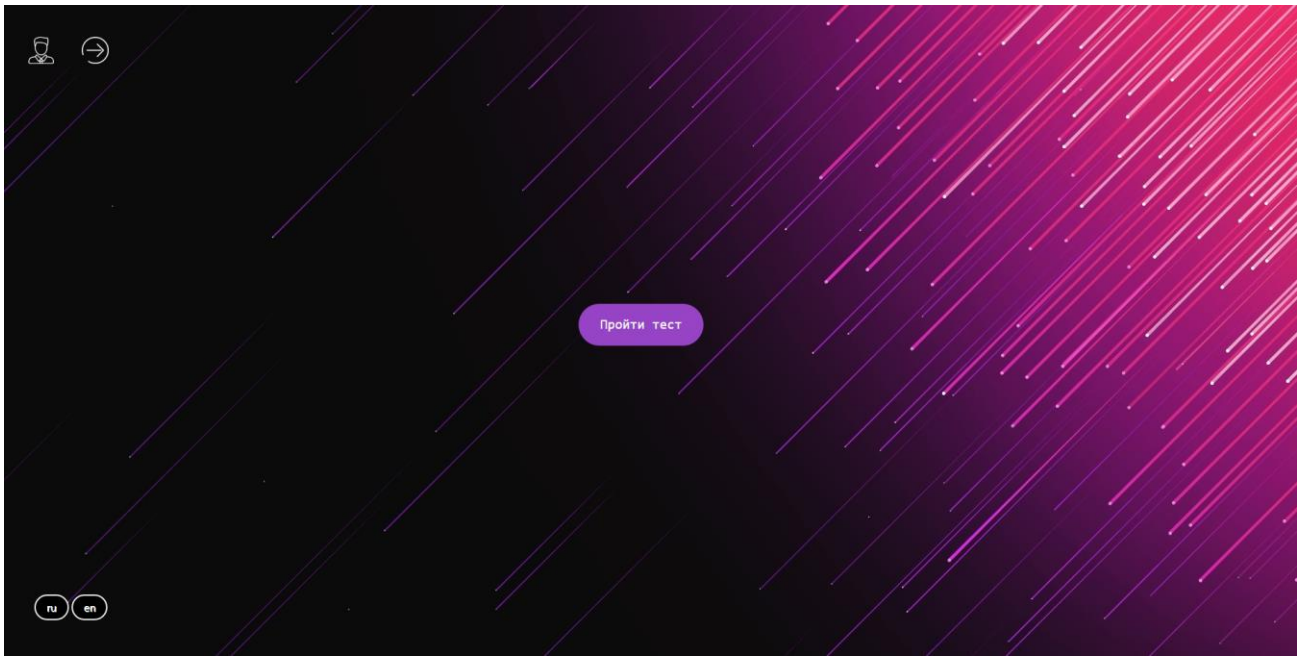

Рис 2.30

Якщо користувач натискає пройти тест, то його перенаправляє до сторінки вибору тестів. Контролер знаходить завдяки методам та алгоритму всю потрібну інформацію з бази даних та підставляє її до сторінки відображення. Як вказано на рис. 2.31, зараз у таблиці знаходиться два теста для проходження.

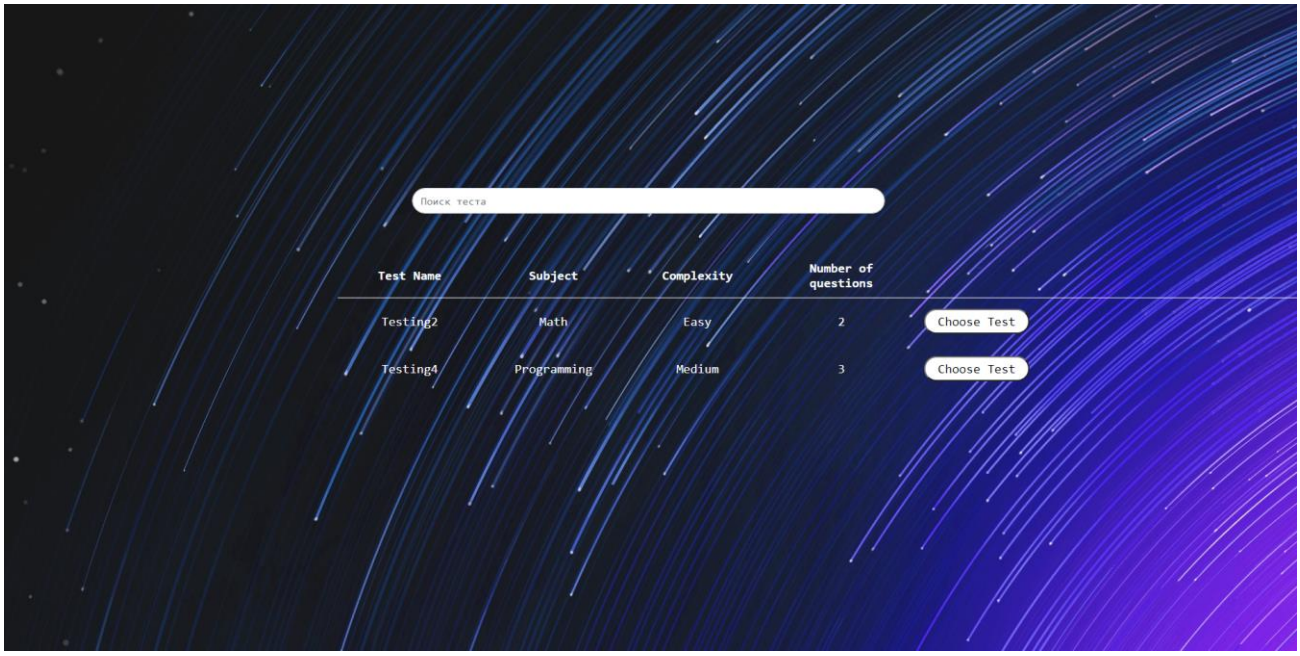

Рис. 2.31

На рис. 2.32 відображенно детальніше поля тесту на сторінці, а саме:

– назва тесту;
– предметна область;
– складність;
– кількість питань.

Рис. 2.32

Всі поля використовуються у фільтрації пошука теста (рис 2.33). Тобто якщо користувач, наприклад, бажає пройти тест по конкретній предметної області, він має вказати початок або цілком слово.

Алгоритм фільтрації таблиці:

```
function tableSearch () {
    var phrase = document.getElementById ('search-text');
    var table = document.getElementById ('test-table');
    var regPhrase = new RegExp (phrase.value, 'i');
    var flag = false;
    for (var i = 1; i <table.rows.length; i ++) {
        flag = false;
        for (var j = table.rows [i] .cells.length - 1; j> = 0; j--) {
            flag = regPhrase.test (table.rows [i] .cells [j] .innerHTML);
            if (flag) break;
        }
        if (flag) {
            table.rows [i] .style.display = "";
        } Else {
            table.rows [i] .style.display = "none";
        }
    }
}
```

Рис. 2.33

Якщо користувач визначився з тестом, він має натиснути обрати тест напроти самого тесту.

Після того як клієнт натискає клавішу вибору теста, з сторінки передається запит до контролера та він зберігає id тесту.

Користувач вибрав тест для подальшого проходження та його перенаправляє до сторінки з тестом (рис 2.39).

Контроллер отримавши запит на тест, знаходить за id, збережений раніше, тест та передає у потрібні місця всі данні, а саме:

- назва тесту;
- час
- всі питання
- всі відповіді до питань

Алгоритм часу виконюється таким чином, що якщо час добігаю кінця, то користувача перенаправяє на домашню сторінку сайта, а на сторінці профілю можна побачити результат тесту, без тих питань на котрі не була дана відповідь.

Код алгоритму :

```
$(document).ready(function (e) {
   var $worked = $("#worked");
   function update() {
      var myTime = $worked.html();
      var ss = myTime.split(":");
      var dt = new Date();
      dt.setHours(0);
      dt.setMinutes(ss[0]);
      dt.setSeconds(ss[1]);
      var dt2 = new Date(dt.valueOf() + 1000);
      var temp = dt2.toTimeString().split(" ");
      var ts = temp[0].split(":");
      $worked.html(ts[1]+":"+ts[2]);
      setTimeout(update, 1000);
   }
   setTimeout(update, 1000);
});
```

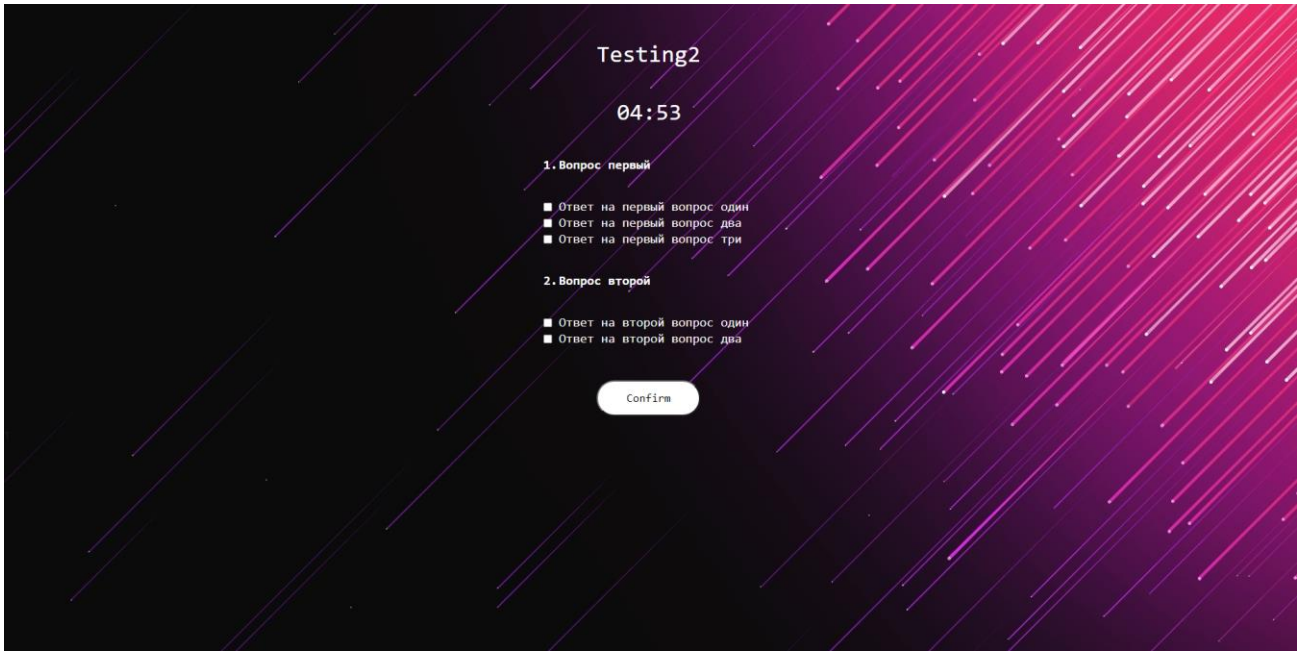

Рис. 2.34

Користувач обирає відповіді, контролер завдяки алгоритмам розуміє які відповіді були правильними, рахує все та видає результат у відсоток.
Для того щоб подивитися результат всіх пройдених тестів, потрібно на головній сторінці натиснути іконку профілю (рис 2.35).

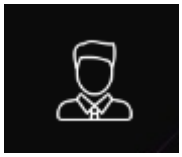

Рис. 2.35

На сторінці профілю відображено всі пройдені тесту користувачем, який зараз знаходиться під цим акаунтом, логін та ім’я користувача (рис. 2.36)

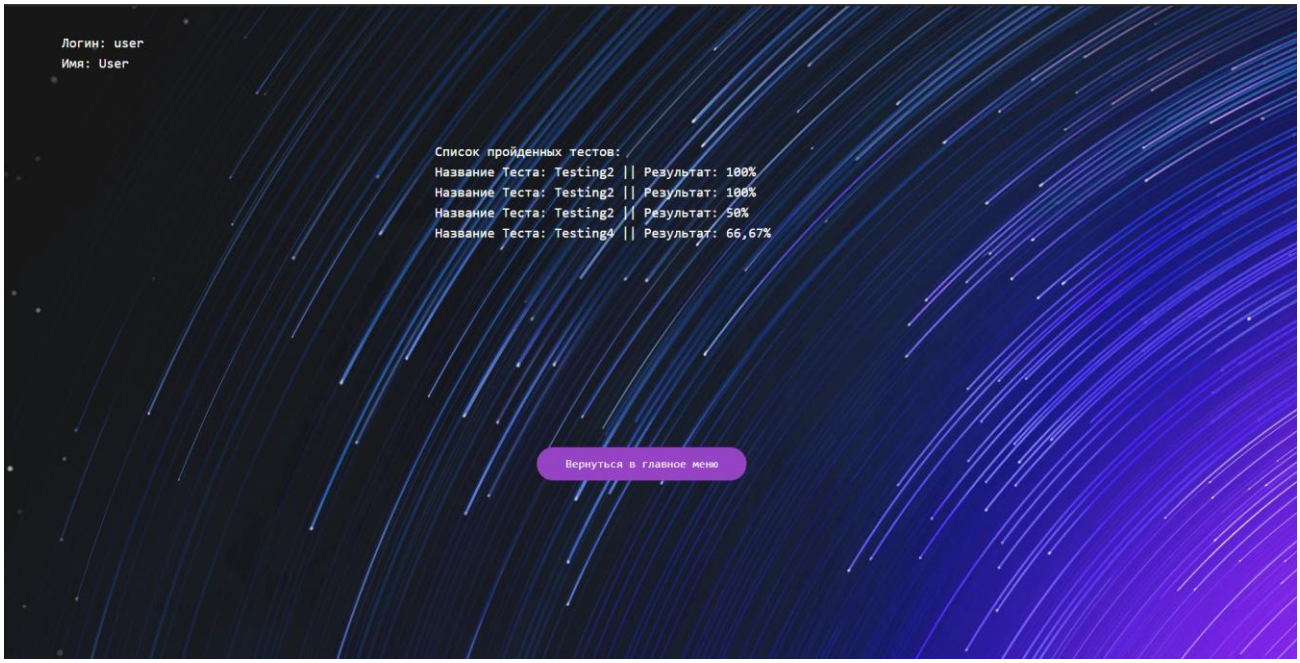

Рис. 2.36

Якщо користувач бажає змінити акаунт, створити новий або просто вийти з нього, то на головній сторінці є кнопка Log out (рис 2.37). Після її натискання, сесія користувача буде завершена і видалена з HttpSession.

Рис. 2.37

Веб-додаток має іншу роль користувачів – admin. Він буде створений в кількості один в базі на початку виконання програми. Для того щоб увійти до акаунту адміністратора, потрібно ввести його дані у форму логіна. Контролер перевіряє чи є такий користувач, бачe що в нього є роль admin, та надає йому статус адміністратора у сесії. На рис. 2.38 відображена головна сторінка, де є

кнопки котрі бачить тільки адміністратор.

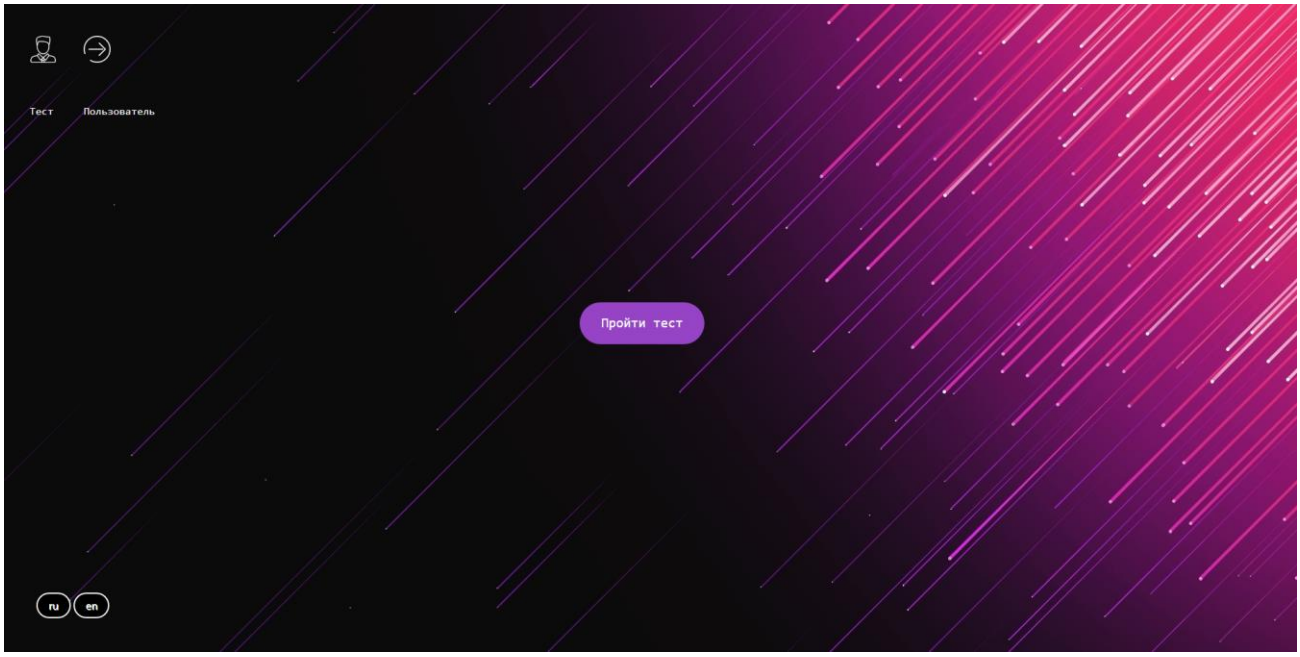

Рис. 2.38

Одна з можливостей адміністратора це адміністрування тестами за допомогою сторінки «Тест». Реалізовані дві можливості: створення тесту та видалення (рис. 2.39).

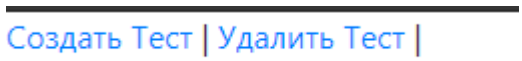

Рис. 2.39

Створення тесту реалізовано через отримання данів через поля, конвертацію даних до потрібного формату за допомогою методів та алгоритмів, а далі запису до бази даних. На рис. 2.40 відображено всі поля, котрі необхідно заповнити на цьому епапі для формування теста. А саме:

- назва теста;
- предметна область;
- складність;

- час виконання;
- кількість питань.

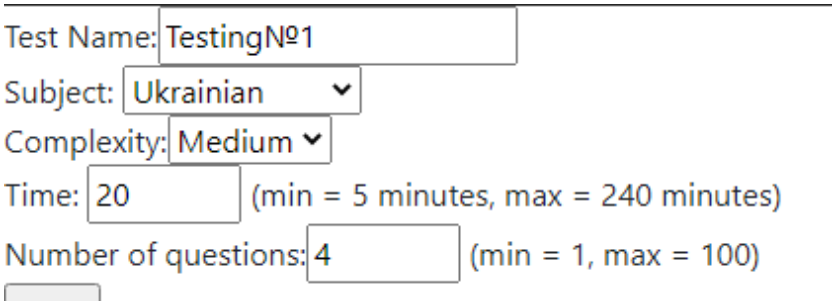

Рис. 2.40

На рис. 2.41 позначено поля для заповнення питання тесту. Кількість полів питань визначається з попередньо введених даних. Після того, як користувач вкаже всі потрібні дані і натисне «Next», дані конвертуються і додаються до об’єкту тест, який був частково заповнений на пепередній сторінці.

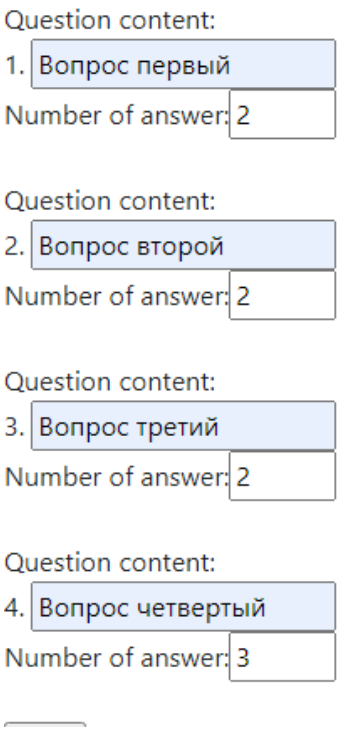

Рис. 2.41

Останій етап формування тесту, це заповнення полів відповідей, до всіх питань. Поряд с текстовою формою, є checkbox, який за стандартом має значеня – false. Тобто ця відповідь не є правильною. Для вибору яка відповідь буде правильною, потрібно натиснути на флаг поля и змінити статус на активний (рис. 2.42)

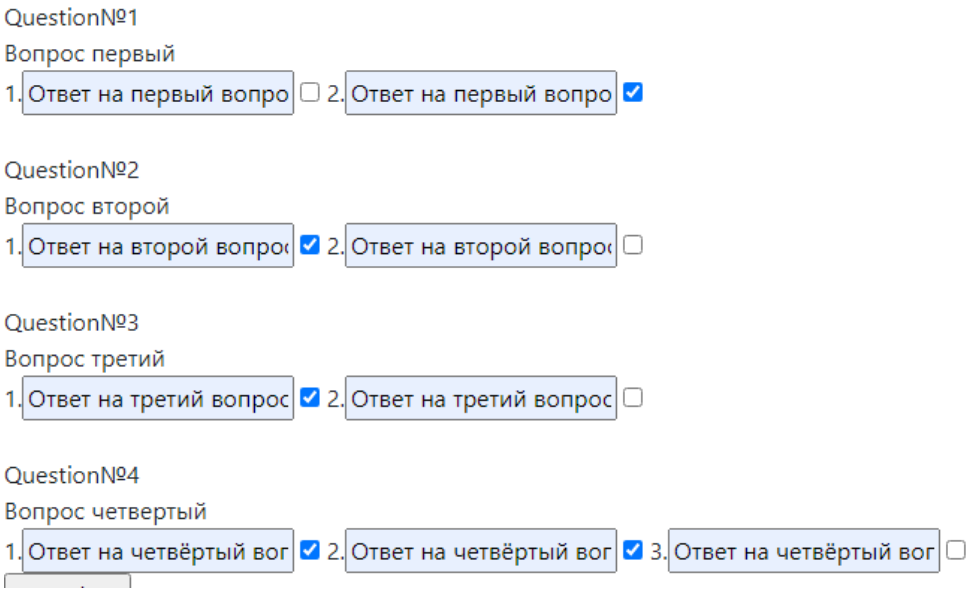

Рис. 2.42

Друга частина адміністрування теста є видалення об’єкта тест с бази даних (рис. 2.43). Контролер передає до сторінки всі знайдені об’єкти теста у вигляді списку назв самих тестів. Після вибора, потрібно натиснуту «Delete». С сторінки передається id вибраного теста, до контролера, який в свою чергу передає до метода видалення id теста, котрий необхідно видалити. Всі данні пов’язані з цим тестом будуть видалені. Тобто: сам тест, його питання, відповіді та результат у профіля користувачів, якщо попередньо він був пройденим ним.

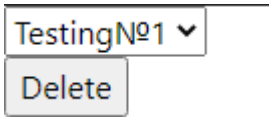

Рис. 2.43

Інший об’єкт для адміністрування є – користувачи. Щоб перейти до сторінки з адмініструванням користувачів, потрібно натиснути кнопку «Пользователь», під залогіненим акаунтом адміністратора, на головній сторінці (рис. 2.44).

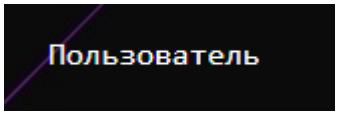

Рис. 2.44

Адміністратор може заблокувати користувачів, розблокувати та змінити данні користувача (рис. 2.45).

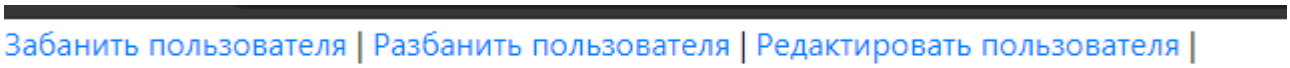

Рис. 2.45

На сторінці блокування є перелік користувачів (рис. 2.46), котрі не є адміністраторами. Список користувачів витягується за бази даних інтерфейсом UserRepository, далі передається до сервісу, котрий в свою чергу передає дані вже отфільтрованих користувачів за параметрами, котрі необхідно враховувати при блокуванні.

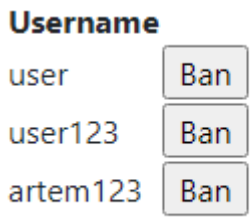

Рис. 2.46

Якщо користувач спробує зайти під заблокаваним аккаунтом, спрацює фільтр веб-додатка. Цей фільтр перевіряє на статус акаунта, та якщо він заблокований, користувача перенаправляє до сторінки з помилкою (рис. 2.47).

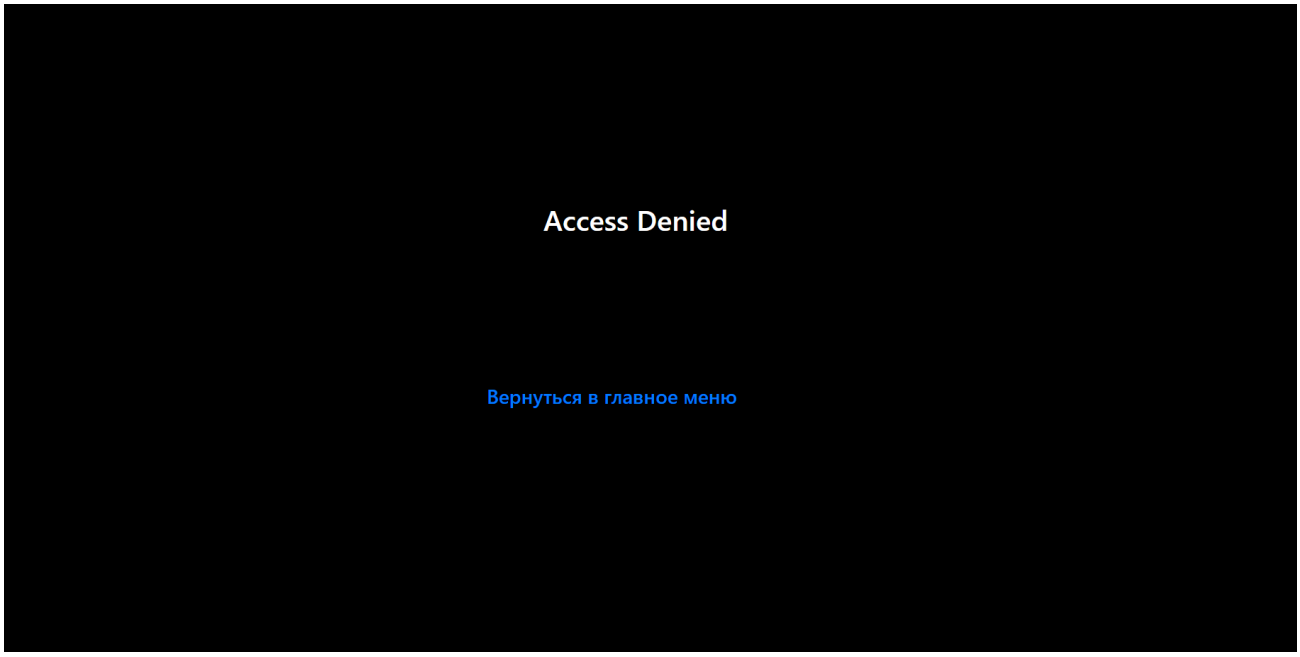

Рис. 2.47

Адміністратор також може разблокувати користувача, за допомогою сторінки «Разбанить пользователя» (рис. 2.48). Для цього з бази даних береться алгоритмом список користувачів, котрі мають статус заблоковані. Id вибраного користувача передається до метода зміни статуса на активний та акаунт знов буде відновлений.

Рис. 2.48

Для зміни даних користувачів, адміністратор може перейти до сторінки «Редактировать пользователя» (рис. 2.49). Весь список користувачів відображається на сторінці з їх полями. Всі зміни полів програма зчитує та вносить до кожного користувача, залежно від змінених полів цього об’єкта.

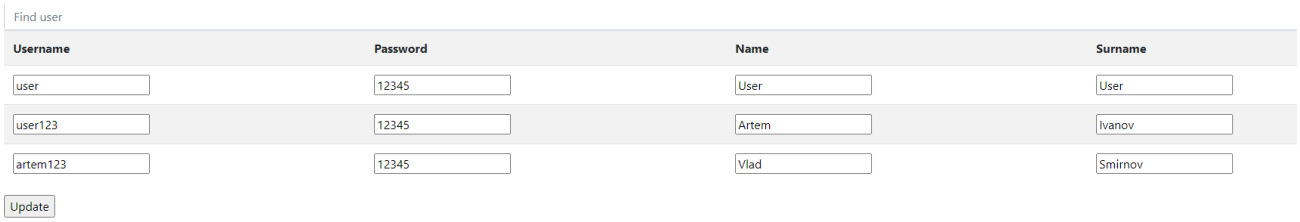

Рис 2.49

Для більш зручнішого використання зміни інформації користувачів, був доданий пошук. Він працює за всіма полями, котрі є у сутності user.

Алгоритм пошуку та фільтрації:

```
function filtr(cells) {
   var input, filter, table, tr, td, i;
   input = document.getElementById("myInput");
   filter = input.value.toUpperCase();
   table = document.getElementById("info-table");
   tr = table.getElementsByTagName("tr");
   for (i = 0; i < tr.length; i++) {
      td = tr[i].getElementsByTagName("td")[0];
      if (td) {
         if (td.innerHTML.toUpperCase().indexOf(filter) > -1) {
            tr[i].style.display = "";
         } else {
            tr[i].style.display = "none";
         }
      }
   }
}
```

РОЗДІЛ 3

ЕКОНОМІЧНИЙ РОЗДІЛ

Вихідні дані розробки програмного забезпечення:

а) передбачуване число операторів – 2235;

б) коефіцієнт складності програми – 1,75;

в) коефіцієнт кореляції програми в ході її розробки – 0,06;

г) середня годинна заробітна плата програміста, грн/год – 93,56;

д) коефіцієнт кваліфікації програміста, обумовлений від стажу – 1,2;

е) вартість машино-години ЕОМ, грн/год – 4,2.

3.1. Визначення трудомісткості розробки програмного забезпечення

Трудомісткість розробки ПЗ можна розрахувати за формулою:

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

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 60 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі,

t_a – витрати праці на розробку блок-схеми алгоритму,

t_n – витрати праці на програмування по готовій блок-схемі,

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ,

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C(1 + p), \text{ людино-годин,} \qquad (3.2)$$

де q – передбачуване число операторів,

C – коефіцієнт складності програми,

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 2235 \cdot 1{,}7 \cdot (1 + 0{,}07) = 4065{,}47$$

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

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,} \qquad (3.3)$$

де B, яке дорівнює 1,2, – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k, яке дорівнює 1,2, – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності.

$$t_u = \frac{4065{,}47 \cdot 1{,}2}{80 \cdot 1{,}2} = 50{,}82,$$ людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25) \cdot k}; \quad (3.4)$$

$$t_a = \frac{4065{,}47}{23 \cdot 1{,}2} = 147{,}3,$$ людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25) \cdot k}, \quad (3.5)$$

$$t_n = \frac{4065{,}47}{22 \cdot 1{,}2} = 154,$$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k}; \quad (3.6)$$

$$t_{отл} = \frac{4065{,}47}{4 \cdot 1{,}2} = 846{,}98$$, людино-годин,

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}4 \cdot t_{отл}; \quad (3.7)$$

$$t_{отл}^{к} = 1{,}4 \cdot 846{,}98 = 1185{,}77$$, людино-годин.

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15..20) \cdot k}; \quad (3.9)$$

$$t_{др} = \frac{4065,47}{17 \cdot 1,2} = 199,29$$, людино-годин.

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

$$t_{до} = 0,75 \cdot t_{д}; \quad (3.10)$$

$$t_{до} = 0,75 \cdot 199,29 = 149,47$$, людино-годин.

$$t_{д} = 199,29 + 149,47 = 348,76$$, людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

$$t_{д} = 60 + 50,82 + 147,3 + 154 + 846,98 + 348.76 = 1607,86$$, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1202,69 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного для налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин,

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1607{,}86 \cdot 93{,}56 = 150431{,}38, \text{ грн.}$$

$З_{МВ}$ – вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot С_{МЧ}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

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

$$З_{МВ} = 846{,}98 \cdot 4{,}2 = 3557{,}32, \text{ грн,}$$

$$К_{ПО} = 150431{,}38 + 3557{,}32 = 153988{,}7, \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес}, \tag{3.14}$$

де B_k – число виконавців,

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

$$T = \frac{1607{,}86}{1 \cdot 176} \approx 9{,}14 \text{ міс.}$$

Висновки: час розробки даного програмного забезпечення складає 1607,86 людино-годин. Таким чином, очікувана тривалість розробки складе 9,14 місяця при 40 годинному робочому тижні (місячний фонд робочого часу 176 годин), а витрати на створення програмного забезпечення складатимуть 150431,38 грн.

ВИСНОВКИ

Головною метою роботи є створення веб-сайту, який представляє собою інструмент для вирішення завдань діагностики і навчання, методом автоматизованої системи тестування.

Система тестування може бути дуже корисною для діагностики та покращення навчання. Що може сприяти покращенню продуктивності користувача та ефективності сприйняття інформації. При розробці даної системи необхідно використовувати нові технології та підходи. Це може сприяти покращенню користувацького досвіду та технічної підтримки додатку в майбутньому.

Головними критеріями розроблювального веб-додатку є:

- ✓ Зручність в використанні.
- ✓ Максимальна доступність для користувача.
- ✓ Легкість в освоєнні.

Система призначена для:

- ✓ Створення інструменту тестування та діагностики.
- ✓ Використання автоматизованого тестування для навчання та діагностики.

Система позиціонується як веб-додаток, який дає можливість використовувати ресурси тестування для покращення, оцінки та діагностики своїх знань.

Визначено трудомісткість розробленої інформаційної системи (1607,86 людино-годин), проведений підрахунок вартості роботи по створенню програми (150431,38 грн) та розраховано час на його створення (9,14 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційна система тестування. URL: https://www.effective-group.ru/services/testing.html
2. Життєвий цикл інформаційної системи. URL: https://sites.google.com/site/metodsybd/blok-5-etapy-ziznennogo-cikla/5-4-testirovanie-is
3. Вимоги до створення сайту. URL: https://likiweb.ru/blog/trebovaniya-k-saitu
4. Алгоритми створення сайту. URL: https://cetera.ru/about/articles/requirements-to-create-site/
5. Алгоритми створення сайту. URL: https://stud.com.ua/97612/informatika/struktura_dodatku
6. Spring-Boot. URL: https://spring.io/projects/spring-boot
7. Hibernate. URL: https://hibernate.org/orm/
8. Lombok. URL: https://projectlombok.org/
9. Maven. URL: https://maven.apache.org/
10. JavaScript. URL: https://learn.javascript.ru/
11. Java Server Pages. URL https://metanit.com/java/javaee/3.1.php
12. Базова структура веб-додатку на Java. URL https://tproger.ru/translations/building-a-web-app-with-java-servlets/
13. Нормалізація баз даних. URL: https://habr.com/ru/post/254773/
14. Структура баз даних. URL: https://www.internet-technologies.ru/articles/rukovodstvo-po-razrabotke-struktury-i-proektirovaniyu-bazy-dannyh.html
15. MySQL. URL: https://www.hostinger.com.ua/rukovodstva/shto-takoje-mysql/
16. Опис праці веб-додатку. URL: https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami

17. Model-View-Controller. URL: https://habr.com/ru/post/181772/
18. Розрахунок вартості машино-години ЕВМ studbooks.net. URL: https://studbooks.net/1786806/geografiya/raschet_stoimosti_mashino_chasa
19. Заробітня плата програмістів в Україні ДОУ. URL: https://jobs.dou.ua/salaries/#period=dec2020&city=Dnipro&title=Junior%20Software%20Engineer&language=Java&spec=&exp1=0&exp2=1

ДОДАТОК А

КОД ПРОГРАМИ

AccessDeniedController.class

```
@Controller
@RequestMapping({"/accessDenied",})
public class AccessDeniedController {
   @GetMapping
   public String accessDenied() {
      return "accessDenied";
   }
}
```

GlobalExceptionHandler.class

```
@Slf4j
@ControllerAdvice
public class GlobalExceptionHandler {

   @ExceptionHandler(Exception.class)
   @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
   public String handleException(Exception e, HttpServletRequest request) {
      log.error(e.getMessage(), e);
      request.setAttribute(EX, SOMETHING_WENT_WRONG);
      return ERROR_PAGE;
   }

   @ExceptionHandler(NoHandlerFoundException.class)
   @ResponseStatus(HttpStatus.NOT_FOUND)
   public String handle404Exception(NoHandlerFoundException e, HttpServletRequest request) {
      log.error(e.getMessage(), e);
      request.setAttribute(EX, PAGE_NOT_FOUND_404);
      return ERROR_PAGE;
   }

   @ResponseStatus(HttpStatus.NOT_FOUND)
   @ExceptionHandler(EntityNotFoundException.class)
   public String handleEntityNotFoundException(HttpServletRequest request, Exception e) {
      log.error(e.getMessage(), e);
      request.setAttribute(EX, ENTITY_NOT_FOUND);
      return ERROR_PAGE;
   }

   @ResponseStatus(HttpStatus.UNAUTHORIZED)
   @ExceptionHandler(WrongLoginOrPasswordException.class)
   public String handleWrongLoginOrPasswordException(HttpServletRequest request, Exception e) {
      log.error(e.getMessage(), e);
      request.setAttribute(EX, e.getMessage());
      return ERROR_PAGE;
   }

   @ResponseStatus(HttpStatus.UNAUTHORIZED)
   @ExceptionHandler(EmptyLoginOrPasswordException.class)
```

```
    public String handleEmptyLoginOrPasswordException(HttpServletRequest request, Exception e) {
        log.error(e.getMessage(), e);
        request.setAttribute(EX, e.getMessage());
        return ERROR_PAGE;
    }

}
```

HomeController.class

```
@Controller
@RequestMapping({"/", "/home"})
public class HomeController {

    @GetMapping
    public String homePage() {
        return "home";
    }

    @PostMapping
    public String getLang(@RequestParam String lang, HttpSession session) {
        session.setAttribute(LANG, lang);
        return "redirect:/home";
    }

}
```

Links.class

```
public class Links {

    //Lang
    public static final String LANG = "lang";

    //User
    public static final String USER_ID = "userId";
    public static final String USERNAME = "username";
    public static final String PASSWORD = "password";
    public static final String NAME = "name";
    public static final String SURNAME = "surname";
    public static final String USER = "user";
    public static final String STATUS = "status";

    //Status
    public static final String IS_LOGGED = "isLogged";
    public static final String USER_ROLE = "userRole";
    public static final String IS_ADMIN = "isAdmin";

    //Test
    public static final String TEST = "test";
    public static final String TEST_ID = "test_id";
    public static final String TIME = "time";
    public static final String NUMBER_Q = "numberQ";
    public static final String SUBJECT = "subject";
```

```
    public static final String COMPLEXITY = "complexity";

    //Question
    public static final String CONTENT = "content";
    public static final String NUMBER_A = "numberA";

    //Answer
    public static final String CORRECT_ANSWER = "correctAnswer";

    //List
    public static final String LIST = "list";
    public static final String SUBJECTS = "subjects";
    public static final String COMPLEXITIES = "complexities";
    public static final String QUESTIONS = "questions";
    public static final String TESTS = "tests";
    public static final String USERS = "users";

}
```

LoginController.class

```
@Slf4j
@Controller
@RequestMapping("/login")
@RequiredArgsConstructor
public class LoginController {

    private final UserService userService;

    @GetMapping
    public String LoginPage() {
        return "login";
    }

    @PostMapping
    public String loginUser(HttpServletRequest request, HttpSession session) {
        String username = request.getParameter(USERNAME);
        String password = request.getParameter(PASSWORD);

        if (username == null || password == null || username.isEmpty() || password.isEmpty()) {
            throw new EmptyLoginOrPasswordException("Login/password cannot be empty");
        }
        User user = userService.findByUsername(username.toLowerCase());

        if (user != null && user.getUsername().equals(username.toLowerCase()) &&
user.getPassword().equals(password)) {
            String userRole = user.getRoleName();
            String lang = session.getAttribute(LANG).toString();
            session.setAttribute(USERNAME, username);
            session.setAttribute(NAME, user.getName());
            session.setAttribute(IS_LOGGED, user);
            session.setAttribute(STATUS, user.getStatus());
            session.setAttribute(USER_ROLE, userRole);
            session.setAttribute(LANG, lang);
```

```
            log.info("User " + user.getUsername() + " logged as " + userRole.toLowerCase());
            if (user.getRoleName().equals(RoleName.ADMIN.getName())) {
                session.setAttribute(IS_ADMIN, RoleName.ADMIN.getName());
            }

            return "redirect:/home";
        } else {
            throw new WrongLoginOrPasswordException("Wrong Username/password");
        }
    }
}
```

LogoutController.class

```
@Slf4j
@RequiredArgsConstructor
@Controller
@RequestMapping("/logout")
public class LogoutController {

    @GetMapping
    public String logout(HttpSession session){
        if(session != null){
            log.debug("Session " + session.getId() + " is over");
            session.invalidate();
        }
        return "redirect:/home";
    }
}
```

ProfileController.class

```
@RequiredArgsConstructor
@Controller
@RequestMapping("/profile")
public class ProfileController {

    private final UserService userService;
    private final TestService testService;

    @GetMapping
    public ModelAndView profilePage(HttpSession session){
        User user = userService.findByUsername(session.getAttribute(USERNAME).toString());
        ModelAndView modelAndView = new ModelAndView("profile");
        modelAndView.addObject(LIST, testService.findAllByUserId(user.getId()));
        return modelAndView;
    }
}
```

RegistrationController.class

```
@Slf4j
@Controller
@RequestMapping("/registration")
@RequiredArgsConstructor
public class RegistrationController {

   private final UserService userService;

   @GetMapping
   public String showRegistration() {
      return "registration";
   }

   @PostMapping
   public String register(@RequestParam Map<String, String> request) {
      User user = userService.create(request.get(USERNAME), request.get(PASSWORD), request.get(NAME),
request.get(SURNAME));
      log.trace("New user has registered --> " + user);
      return "redirect:/home";
   }
}
```

TestAdminController.class

```
@RequiredArgsConstructor
@Controller
@RequestMapping("/testAdmin")
public class TestAdminController {

   private final TestService testService;
   private final SubjectService subjectService;
   private final QuestionService questionService;
   private final AnswerService answerService;
   private final ComplexityService complexityService;

   private List<Question> questions;
   private List<Test> tests;

   @GetMapping
   public String testAdminPage(){
      return "testAdmin";
   }

   @GetMapping("createTest")
   public ModelAndView testAdminCreatePage(){
```

```
    ModelAndView modelAndView = new ModelAndView("createTest");
    modelAndView.addObject(SUBJECTS, subjectService.findAll());
    modelAndView.addObject(COMPLEXITIES, complexityService.findAll());
    return modelAndView;
}

@PostMapping("createTest")
public String testAdminCreate(HttpServletRequest request, HttpSession session){
    Test test = testService.create(request.getParameter(NAME),
            Integer.parseInt(request.getParameter(TIME)),
            Long.parseLong(request.getParameter(SUBJECT)),
            Long.parseLong(request.getParameter(COMPLEXITY)));
    session.setAttribute(TEST_ID, test.getId());
    session.setAttribute(NUMBER_Q, request.getParameter(NUMBER_Q));
    return "redirect:/testAdmin/createQuestion";
}

@GetMapping("createQuestion")
public ModelAndView questionAdminCreatePage(){
    return new ModelAndView("createQuestion");
}

@PostMapping("createQuestion")
public String questionAdminCreate(HttpServletRequest request, HttpSession session){
    questions = questionService.createListOfQ(Integer.parseInt(session.getAttribute(NUMBER_Q).toString()),
            request,
            CONTENT,
            NUMBER_A,
            Long.parseLong(session.getAttribute(TEST_ID).toString()));
    return "redirect:/testAdmin/createAnswer";
}

@GetMapping("createAnswer")
public ModelAndView answerAdminCreatePage(){
    ModelAndView modelAndView = new ModelAndView("createAnswer");
    modelAndView.addObject(QUESTIONS, questions);
    return modelAndView;
}

@PostMapping("createAnswer")
public String answerAdminCreate(HttpServletRequest request){
    answerService.createListOfAnswer(questions, request);
    return "redirect:/home";
}

@GetMapping("deleteTest")
public ModelAndView testAdminDeletePage(){
    ModelAndView modelAndView = new ModelAndView("deleteTest");
    tests = testService.findAll();
```

```
        modelAndView.addObject(TESTS, tests);
        return modelAndView;
    }

    @PostMapping("deleteTest")
    public String testAdminDelete(HttpServletRequest request){
        testService.deleteById(request.getParameter(TEST));
        return "redirect:/home";
    }
}
```

TestController.class

```
@RequiredArgsConstructor
@Controller
@RequestMapping("/chooseTest")
public class TestController {

    private final TestService testService;
    private Long testId;
    private List<Question> questions;

    @GetMapping
    public ModelAndView testPage(){
        ModelAndView modelAndView = new ModelAndView("chooseTest");
        modelAndView.addObject(TESTS, testService.findAll());
        return modelAndView;
    }

    @PostMapping
    public String test(HttpServletRequest request){
        testId = Long.parseLong(request.getParameter(TEST_ID));
        return "redirect:/chooseTest/test";
    }

    @GetMapping("test")
    public ModelAndView testPassingPage(){
        ModelAndView modelAndView = new ModelAndView("test");
        Test test = testService.findById(testId);
        questions = test.getQuestionList();
        for (Question question: questions){
            for(Answer answer: question.getAnswerList()){
                modelAndView.addObject(CORRECT_ANSWER + answer.getId(), answer.getCorrectAnswer());
            }
        }

        modelAndView.addObject(QUESTIONS, questions);
        modelAndView.addObject(NAME, test.getName());
```

```
        modelAndView.addObject(TIME,test.getTime());
        return modelAndView;
    }

    @PostMapping("test")
    public String testPassing(HttpServletRequest request, HttpSession session){
       testService.passingTest(request, session, questions, testId);
        return "redirect:/home";
    }
}
```

UserAdminController.class

```
@Controller
@RequestMapping("/userAdmin")
@RequiredArgsConstructor
public class UserAdminController {

    private final UserService userService;

    @GetMapping
    public String userAdminPage(){
        return "userAdmin";
    }

    @GetMapping("ban")
    public ModelAndView userAdminBanPage(){
        ModelAndView modelAndView = new ModelAndView("ban");
        modelAndView.addObject(USERS, userService.findAllActive());
        return modelAndView;
    }

    @PostMapping("ban")
    public String userAdminBan(HttpServletRequest request){
        userService.statusUser(request.getParameter(USER_ID), false);
        return "redirect:/userAdmin";
    }

    @GetMapping("unban")
    public ModelAndView userAdminUnBanPage(){
        ModelAndView modelAndView = new ModelAndView("unban");
        modelAndView.addObject(USERS, userService.findAllInactive());
        return modelAndView;
    }

    @PostMapping("unban")
    public String userAdminUnban(HttpServletRequest request){
        userService.statusUser(request.getParameter(USER), true);
```

```
        return "redirect:/userAdmin";
    }

    @GetMapping("edit")
    public ModelAndView userAdminEditPage(){
        ModelAndView modelAndView = new ModelAndView("editUser");
        modelAndView.addObject(USERS, userService.findAllByUserRole());
        return modelAndView;
    }

    @PostMapping("edit")
    public String userAdminEdit(HttpServletRequest request){
        userService.update(userService.findAllByUserRole(), request);
        return "redirect:/userAdmin";
    }
}
```

GradeWithTestName.class

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class GradeWithTestName {

    private String result;

    private String testName;

}
```

Answer.class

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "answers")
public class Answer {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String variant;

    private Integer counterAnswer;
```

```
    private Boolean correctAnswer;

    @ManyToOne
    @JoinColumn(name="question_id", nullable=false)
    private Question question;

}
```

Complexity.class

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "complexities")
public class Complexity {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy="complexity", cascade = CascadeType.ALL)
    private List<Test> testList;
}
```

Grade.class

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "grades")
public class Grade {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String result;
```

```
    @ManyToOne
    @JoinColumn(name="test_id", nullable=false)
    private Test test;

    @ManyToOne
    @JoinColumn(name="user_id", nullable=false)
    private User user;
}
```

Question.class

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "questions")
public class Question {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String content;

    private Integer counterQuestion;

    @OneToMany(mappedBy="question", cascade = CascadeType.ALL)
    private List<Answer> answerList;

    private Integer numberOfAnswer;

    @ManyToOne
    @JoinColumn(name="test_id", nullable=false)
    private Test test;

}
```

Role.class

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
```

```
@Table(name = "roles")
public class Role {

   @Id
   @Column(name = "id")
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String role;
}
```

RoleName.class

```
@Getter
@AllArgsConstructor
public enum RoleName {
   USER("user"),
   ADMIN("admin");

   private final String name;
}
```

Subject.class

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "subjects")
public class Subject {

   @Id
   @Column(name = "id")
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String name;

   @OneToMany(mappedBy="subject", cascade = CascadeType.ALL)
   private List<Test> testList;
}
```

Test.class

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "tests")
public class Test {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private Integer time;

    @ManyToOne
    @JoinColumn(name="subject_id", nullable=false)
    private Subject subject;

    @ManyToOne
    @JoinColumn(name="complexity_id", nullable=false)
    private Complexity complexity;

    @OneToMany(mappedBy="test", cascade = CascadeType.ALL)
    private List<Question> questionList;

    @OneToMany(mappedBy="test", cascade = CascadeType.ALL)
    private List<Grade> grades;
}
```

User.class

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    private String username;

    private String password;

    private String name;

    private String surname;

    private Boolean status;

    private String roleName;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Grade> grades;

}
```

ConnectionException.class

```
public class ConnectionException extends RuntimeException {
    public ConnectionException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

DBException.class

```
public class DBException extends RuntimeException {

    public DBException(Throwable cause) {
        super(cause);
    }

    public DBException(String message, Throwable cause) {
        super(message, cause);
    }

    public DBException() {

    }
}
```

EmptyLoginOrPasswordException.class

```
public class EmptyLoginOrPasswordException extends RuntimeException{
    public EmptyLoginOrPasswordException(String message) {
        super(message);
```

```
    }
}
```

EntityNotFoundException.class

```
public class EntityNotFoundException extends RuntimeException{
}
```

Messages.class

```
public class Messages {
    public static final String EX = "ex";
    public static final String ERROR_PAGE = "errorPage/errorPage";
    public static final String SOMETHING_WENT_WRONG = "Something went wrong";
    public static final String PAGE_NOT_FOUND_404 = "Page not found (404)";
    public static final String ENTITY_NOT_FOUND = "Entity not found";
    public static final String ERR_CANNOT_OBTAIN_USER_BY_ID = "Cannot obtain a user by its id";

}
```

RoleNotFoundException.class

```
public class NotCreateException extends RuntimeException{
    public NotCreateException(String message) {
        super(message);
    }
}
```

NotCreateException.class

```
public class RoleNotFoundException extends EntityNotFoundException{
}
```

UserNotFoundException.class

```
public class UserNotFoundException extends EntityNotFoundException {
}
```

WrongLoginOrPasswordException.class

```
public class WrongLoginOrPasswordException extends RuntimeException {
    public WrongLoginOrPasswordException(String message) {
        super(message);
    }
```

```
}
```

AccessFilter.class

```
@Slf4j
@Component
public class AccessFilter extends GenericFilterBean {

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws
IOException, ServletException {
      HttpServletRequest httpRequest = (HttpServletRequest) request;
      HttpServletResponse httpResponse = (HttpServletResponse) response;
      String role = (String) httpRequest.getSession().getAttribute("userRole");
      if (role != null && role.equals(RoleName.ADMIN.getName())) {
         filterChain.doFilter(request, response);
      } else {
         httpResponse.sendRedirect("accessDenied");
      }

   }
}
```

BanFilter.class

```
@Slf4j
@Component
public class BanFilter extends GenericFilterBean {

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws
IOException, ServletException {
      HttpServletRequest httpRequest = (HttpServletRequest) request;
      HttpServletResponse httpResponse = (HttpServletResponse) response;
      Object status = httpRequest.getSession().getAttribute("status");
      if (status != null && status.equals(true)) {
         filterChain.doFilter(request, response);
      } else {
         httpResponse.sendRedirect("accessDenied");
      }

   }
}
```

AnswerRepository.interface

```
@Repository
public interface AnswerRepository extends JpaRepository<Answer, Long> {
}
```

ComplexityRepository.interface

```
@Repository
public interface ComplexityRepository extends JpaRepository<Complexity, Long> {
}
```

GradeRepository.interface

```
@Repository
public interface GradeRepository extends JpaRepository<Grade, Long> {

   List<Grade> findAllByUserId(Long userId);
}
```

QuestionRepository.interface

```
@Repository
public interface QuestionRepository extends JpaRepository<Question, Long> {
}
```

SubjectRepository.interface

```
@Repository
public interface SubjectRepository extends JpaRepository<Subject, Long> {
}
```

TestRepository.interface

```
@Repository
public interface TestRepository extends JpaRepository<Test, Long> {
}
```

UserRepository.interface

```
public interface UserRepository extends JpaRepository<User, Long> {

   Optional<User> findByUsername(String username);

   @Query(value = "SELECT * FROM users where users.role_name='user'", nativeQuery = true)
   List<User> findAllByUserRole();
```

```
    @Query(value = "select * from users where users.status=?1 and users.role_name='user' ", nativeQuery = true)
    List<User> findAllByStatus(Boolean status);

}
```

AnswerServiceImpl.class

```
@RequiredArgsConstructor
@Service
public class AnswerServiceImpl implements AnswerService {

    private final AnswerRepository answerRepository;

    public void createListOfAnswer(List<Question> questions, HttpServletRequest request) {
        for (Question question : questions) {
            for (int i = 1; i <= question.getNumberOfAnswer(); i++) {
                String variant = request.getParameter("variant" + question.getCounterQuestion() + i);
                String correctAnswer = request.getParameter("correctAnswer" + question.getCounterQuestion() + i);
                if (correctAnswer == null) {
                    correctAnswer = "false";
                }

                Answer answer = Answer.builder()
                        .variant(variant)
                        .counterAnswer(i)
                        .correctAnswer(Boolean.parseBoolean(correctAnswer))
                        .question(question)
                        .build();
                answerRepository.save(answer);
            }
        }
        questions.clear();
    }
}
```

ComplexityServiceImpl.class

```
@Service
@RequiredArgsConstructor
public class ComplexityServiceImpl implements ComplexityService {

    private final ComplexityRepository complexityRepository;

    public List<Complexity> findAll() {
        return complexityRepository.findAll();
```

```
    }
}
```

GradeServiceImpl.class

```
@Service
@RequiredArgsConstructor
public class GradeServiceImpl implements GradeService {

    private final GradeRepository gradeRepository;

    public Grade create (String result, User user, Test test){
        Grade grade = Grade.builder()
                .result(result)
                .user(user)
                .test(test)
                .build();
        return gradeRepository.save(grade);
    }
}
```

QuestionServiceImpl.class

```
@Service
@RequiredArgsConstructor
public class QuestionServiceImpl implements QuestionService {

    private final QuestionRepository questionRepository;
    private final TestRepository testRepository;

    public List<Question> createListOfQ(int numberOfQ, HttpServletRequest request, String content, String
number, long testId) {
        List<Question> questions = new ArrayList<>();
        Test test = testRepository.findById(testId).orElseThrow(EntityNotFoundException::new);
        for (int i = 1; i <= numberOfQ; i++) {
            Question question = Question.builder()
                    .content(request.getParameter(content + i))
                    .counterQuestion(i)
                    .numberOfAnswer(Integer.parseInt(request.getParameter(number + i)))
                    .test(test)
                    .build();
            questionRepository.save(question);
            questions.add(question);
        }
        return questions;
    }
```

```
    //Method for reducing the remainder to hundredths
    public String percent(double numberOfQ, double numberOfA) {
        DecimalFormat df = new DecimalFormat("###.##");
        int k = 100;
        double result = (numberOfA * k) / numberOfQ;
        return df.format(result);
    }
}
```

SubjectServiceImpl.class

```
@Service
@RequiredArgsConstructor
public class SubjectServiceImpl implements SubjectService {

   private final SubjectRepository subjectRepository;

    public List<Subject> findAll() {
        return subjectRepository.findAll();
    }
}
```

TestServiceImpl.class

```
@Slf4j
@Service
@RequiredArgsConstructor
public class TestServiceImpl implements TestService {

    private final TestRepository testRepository;
    private final SubjectRepository subjectRepository;
    private final ComplexityRepository complexityRepository;
    private final GradeRepository gradeRepository;
    private final QuestionService questionService;
    private final GradeService gradeService;
    private final UserService userService;

    public List<Test> findAll() {
        return testRepository.findAll();
    }

    public Test findById(Long testId) {
        return testRepository.findById(testId).orElseThrow(EntityNotFoundException::new);
    }

    public List<GradeWithTestName> findAllByUserId(Long userId) {
        List<GradeWithTestName> grades = new ArrayList<>();
```

```
        gradeRepository.findAllByUserId(userId).forEach(grade -> grades.add(GradeWithTestName
                .builder()
                .result(grade.getResult())

.testName(testRepository.findById(grade.getTest().getId()).orElseThrow(EntityNotFoundException::new).getNam
e())
                .build()));
        return grades;
    }

    public Test create(String name, int time, long subjectId, long complexityId) {
        Test test = Test.builder()
                .name(name)
                .time(time)
                .subject(subjectRepository.findById(subjectId).orElseThrow(EntityNotFoundException::new))

.complexity(complexityRepository.findById(complexityId).orElseThrow(EntityNotFoundException::new))
                .build();
        return testRepository.save(test);
    }

    public void deleteById(String reqTestId) {
        testRepository.deleteById(Long.parseLong(reqTestId));
    }

    public void passingTest(HttpServletRequest request, HttpSession session, List<Question> questions, Long
testId) {
        int k = 0;
        for (Question question : questions) {
            for (Answer answer : question.getAnswerList()) {
                String correctAnswerS = request.getParameter("correctAnswer" + answer.getId());
                if (Boolean.parseBoolean(correctAnswerS)) {
                    k++;
                }
            }
        }

        String result = questionService.percent(questions.size(), k);
        String username = session.getAttribute("username").toString();

        gradeService.create(result,
                userService.findByUsername(username),
                testRepository.findById(testId).orElseThrow(EntityNotFoundException::new));

        log.info(userService.findByUsername(username).getUsername() + " passed the test " + result + "%");
    }
}
```

UserServiceImpl.class

```
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

   private final UserRepository userRepository;

   public User findByUsername(String username) {
      return
userRepository.findByUsername(username.toLowerCase()).orElseThrow(UserNotFoundException::new);
   }

   public List<User> findAllByUserRole() {
      return userRepository.findAllByUserRole();
   }

   public List<User> findAllActive() {
      return userRepository.findAllByStatus(true);
   }

   public List<User> findAllInactive() {
      return userRepository.findAllByStatus(false);
   }

   public User create(String username, String password, String name, String surname) {
      User user = User.builder()
            .username(username.toLowerCase())
            .password(password)
            .status(true)
            .roleName(RoleName.USER.getName())
            .name(UserServiceImpl.firstLetterUpperCase(name))
            .surname(UserServiceImpl.firstLetterUpperCase(surname))
            .build();
      if (password != null) {
         return userRepository.save(user);
      } else throw new NotCreateException("Can`t create this user");
   }

   @Transactional
   public void statusUser(String requestUserId, boolean status) {
      User user =
userRepository.findById(Long.parseLong(requestUserId)).orElseThrow(UserNotFoundException::new);
      user.setStatus(status);
   }

   @Transactional
   public void update(List<User> users, HttpServletRequest request) {
      for (User user : users) {
```

```
            user.setUsername(request.getParameter("username" + user.getId()));
            user.setPassword(request.getParameter("password" + user.getId()));
            user.setName(request.getParameter("name" + user.getId()));
            user.setSurname(request.getParameter("surname" + user.getId()));
        }
    }

    private static String firstLetterUpperCase(String word) {
        return word.substring(0, 1).toUpperCase() + word.substring(1).toLowerCase();
    }

}
```

AnswerService.interface

```
public interface AnswerService {

    void createListOfAnswer (List<Question> questions, HttpServletRequest request);

}
```

ComplexityService.interface

```
public interface ComplexityService {

    List<Complexity> findAll();
}
```

GradeService.interface

```
public interface GradeService {

    Grade create (String result, User user, Test test);
}
```

QuestionService.interface

```
public interface QuestionService {

    List <Question> createListOfQ(int numberOfQ, HttpServletRequest request, String content, String number, long
testId);

    String percent (double numberOfQ, double numberOfA);
}
```

SubjectService.interface

```
public interface SubjectService {

   List<Subject> findAll();
}
```

TestService.interface

```
public interface TestService {

   Test findById(Long testId);

   List<Test> findAll();

   List<GradeWithTestName> findAllByUserId(Long userId);

   Test create (String name, int time, long subjectId, long complexityId);

   void deleteById(String reqTestId);

   void passingTest(HttpServletRequest request, HttpSession session, List<Question> questions, Long testId);
}
```

UserService.interface

```
public interface UserService {

   User findByUsername(String username);

   List<User> findAllByUserRole();

   List<User> findAllActive();

   List<User> findAllInactive();

   User create(String username, String password, String name, String surname);

   void statusUser(String requestUserId, boolean status);

   void update(List<User> users, HttpServletRequest request);
}
```

TestingApplication

```
@SpringBootApplication
public class TestingApplication {
```

```
    public static void main(String[] args) {
        SpringApplication.run(TestingApplication.class, args);
    }

    @Bean
    FilterRegistrationBean<AccessFilter> accessFilterFilterRegistrationBean() {
        final FilterRegistrationBean<AccessFilter> filterFilterRegistrationBean = new FilterRegistrationBean<>();
        filterFilterRegistrationBean.setFilter(new AccessFilter());
        filterFilterRegistrationBean.addUrlPatterns("/testAdmin", "/createTest", "/deleteTest", "/userAdmin",
"/unban", "/ban");
        return filterFilterRegistrationBean;
    }

    @Bean
    FilterRegistrationBean<BanFilter> banFilterFilterRegistrationBean() {
        final FilterRegistrationBean<BanFilter> filterFilterRegistrationBean = new FilterRegistrationBean<>();
        filterFilterRegistrationBean.setFilter(new BanFilter());
        filterFilterRegistrationBean.addUrlPatterns("/chooseTest", "/test", "/profile");
        return filterFilterRegistrationBean;
    }
}
```