Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Система онлайн тестування з програмування на основі редагування програмного коду»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: студент _2_ курсу, групи _1КІ-21м_
напряму підготовки (спеціальності)
123 – «Комп'ютерна інженерія»
(шифр і назва напряму підготовки, спеціальності)
Нагорний С.М.
(прізвище та ініціали)
Керівник к.т.н, доц кафедри ОТ
к.т.н, Черняк О. І.
(прізвище та ініціали)
Опонент к.т.н, доц кафедри ПЗ
Чорноволик Г.О.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
(прізвище та ініціали)
«03» 12 2022 р

Вінниця ВНТУ 2022

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри
обчислювальної техніки
_____ О.Д. Азаров
"15" 09 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту **Нагорному Сергію Михайловичу**

1 Тема роботи «Система онлайн тестування з програмування на основі редагування програмного коду» керівник роботи Черняк Олександр Іванович к. т, н., професор кафедри ОТ, затверджено наказом вищого навчального закладу від 15.09.2022 року № 205 - А.

2 Строк подання студентом роботи 19.12.2022 р.

3 Вихідні дані до роботи: проведення тестування з програмування методом редагування рядків з кодом, засоби - середовище розробки Intellij IDEA, мови програмування Java та JavaScript, фреймворки Spring та React.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз існуючих способів та засобів створення систем, розробка програмного забезпечення, тестування системи, економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): технічне завдання, графічні зображення роботи програми.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1— Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Черняк О. І. к. т. н., професор кафедри ОТ	24.09.2022	19.12.2022
4	Небава М. І, к. е. н, професор кафедри ЕПВМ	24.11.2022	18.12.2022

7 Дата видачі завдання 24.09.2022.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі роботи	01.09.22	вик.
2	Огляд технологій для створення серверної частини	05.09-12.09.22	вик.
3	Огляд технологій для створення клієнтської частини	12.09-20.09.22	вик.
4	Розробка серверної частини	21.09-10.10.22	вик.
5	Розробка клієнтської частини	11.10-25.10.22	вик.
6	Проведення юніт тестування програмного забезпечення	25.10-03.11.22	вик.
7	Розрахунок економічної частини	04.11-11.11.22	вик.
8	Оформлення пояснювальної записки та ілюстративного матеріалу	11.11-16.10.22	вик.
9	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	17.11-19.11.22	вик.
10	Підписи супроводжувальних документів у керівника, опонента, нормоконтролера	19.11-22.11.22	вик.
11	Перевірка «антиплагіат»	22.11-23.11.22	вик.
12	Попередній захист	24.11.22	вик.

Студент Нагорний Сергій Михайлович

Керівник к.т.н., доц. Черняк Олександр Іванович

АНОТАЦІЯ

УДК 004.51

Нагорний С. М. Система онлайн тестування з програмування на основі редагування програмного коду. Магістерська кваліфікаційна робота зі спеціальності 123 – комп'ютерна інженерія, освітня програма – комп'ютерна інженерія. Вінниця: ВНТУ, 2022. 136с.

На укр.мові. Бібліогр.: 25 назв; рис.: 11 ; табл. 10.

Магістерська робота присв'ячена розробці інформаційної системи тестування студентів з програмування на основі редагування програмного коду. В роботі був проведений аналіз відомих методів розробки систем тестування студентів з програмування на основі редагування програмного коду.

У ході роботи було розроблено систему тестування студентів з програмування на основі редагування програмного коду. В магістерській роботі було виконано економічні розрахунки по визначенню доцільності розробки нового програмного продукту.

Ключові слова: інформаційна система, тестування студентів, мова програмування, редагування коду.

ANNOTATION

Nagornyi S. M. Online testing system for programming based on software code editing. Master's qualification work on specialty 123 - computer engineering, educational program - computer engineering. Vinnytsia: VNTU, 2022. 136p.

In the Ukrainian language. Bibliography: 25 titles; Fig.: 11; table 10.

The master's thesis is devoted to the development of an information system for testing students in programming based on the editing of software code. The paper analyzed the well-known methods of developing systems for testing students in programming based on editing the software code.

In the course of the work, a system of testing students in programming was developed based on editing the software code. In the master's thesis, economic calculations were performed to determine the feasibility of developing a new software product.

Keywords: information system, student testing, programming language, code editing.

ЗМІСТ

ВСТУП

Завданням даної дипломної роботи є розробка системи онлайн тестування з програмування на основі редагування програмного коду, що дає можливість викладачам більш ефективно перевірити знання студентів з мов програмування.

Освіта — це можливість. Здатність знати і захищати свої права та обов'язки, здатність володіти необхідною інформацією. Освіта збагачує наш світогляд, дає нові можливості для розвитку, кращого життя, це можливість зробити світ кращим. Бути освіченим означає не просто закінчити навчальний заклад, а отримати знання, які знадобляться в житті. Освіта забезпечує швидкий процес розвитку людини.

Сучасне суспільство живе в епоху інформаційних технологій. Вже у найближчому майбутньому інформаційні технології в Україні можуть стати першочерговою з економічної точки зору галуззю. Таким чином, постає важливе завдання розвитку професійних та кваліфікованих фахівців, що будуть конкурентоспроможними на світовому ринку, здатними на розробку інноваційних технологічних рішень. Однак зараз на ринку майже немає доступних систем для тестування знань з мов програмування методом редагування коду.

Актуальність дослідження пов’язана з збільшенням кількості бажаючих працювати розробником програмного забезпечення. Проте чим більше бажаючих, тим більше потрібно приділяти уваги їх рівню знань. Саме тому потрібен продукт, за допомогою якого викладачі зможуть швидко і ефективно перевірити рівень знань студентів з мови програмування. Це дає можливість більш інтерактивно проводити тестування з програмування.

Об’єктом дослідження є процес підвищення ефективності перевірки знань студентів з мов програмування.

Предметом дослідження є методи та засоби для підвищення ефективності перевірки знань студентів з мов програмування.

Мета роботи полягає в підвищенні ефективності перевірки знань студентів з мов програмування шляхом створення системи для тестування знань з мов програмування методом редагування рядків коду.

Для досягнення даної мети у дипломній роботі потрібно вирішити такі задачі:

— виявити та вибрати найбільш ефективні та доцільні технології для розробки системи;

— вибрати інструментарій для розробки онлайн системи;

— розробити програмне забезпечення;

— провести тестування розробленого програмного продукту;

— дослідити ефективність системи;

Наукова новизна цієї роботи полягає у створенні системи тестування знань з мов програмування редагуванням рядків методом коментування та роз коментування рядків з кодом.

Практична цінність полягає в створенні програмного забезпечення, що буде цінним інструментом для викладачів під час проведення оцінювання знань студентів з мов програмування.

1.АНАЛІЗ ІСНУЮЧИХ СПОСОБІВ ТА ЗАСОБІВ СТВОРЕННЯ СИСТЕМ

1.1. Мова програмування Java

Java — це об’єктно-орієнтована мова програмування загального призначення, що базується на класах, розроблена для меншої кількості залежностей реалізації. Це обчислювальна платформа для розробки додатків. Тому Java є швидкою, безпечною та надійною. Вона широко використовується для розробки додатків Java у ноутбуках, центрах обробки даних, ігрових приставках, комп’ютерах, мобільних телефонах тощо.

Java — це багатоплатформна, об’єктно-орієнтована та мережево-орієнтована мова. Це одна з найбільш використовуваних мов програмування. Java також використовується як обчислювальна платформа. Вона вважається однією зі швидких, безпечних і надійних мов програмування, якій надає перевагу більшість організацій для створення своїх проектів [1].

Область застосування Java:

— використовується для розробки додатків Android;

— допомагає створювати корпоративне програмне забезпечення;

— широкий вибір мобільних Java-додатків;

— наукові обчислювальні програми;

— використовується для Big Data Analytics;

— програмування апаратних пристроїв на Java;

— використовується для серверних технологій, таких як Apache, JBoss, GlassFish тощо [2].

Розглянувши Рисунок 1.1 можна зробити висновки, що лідирує у даній статистиці Python, а от Java посідає друге місце. PYPL Index створюється шляхом аналізу того, як часто мовні навчальні посібники шукаються на Google.

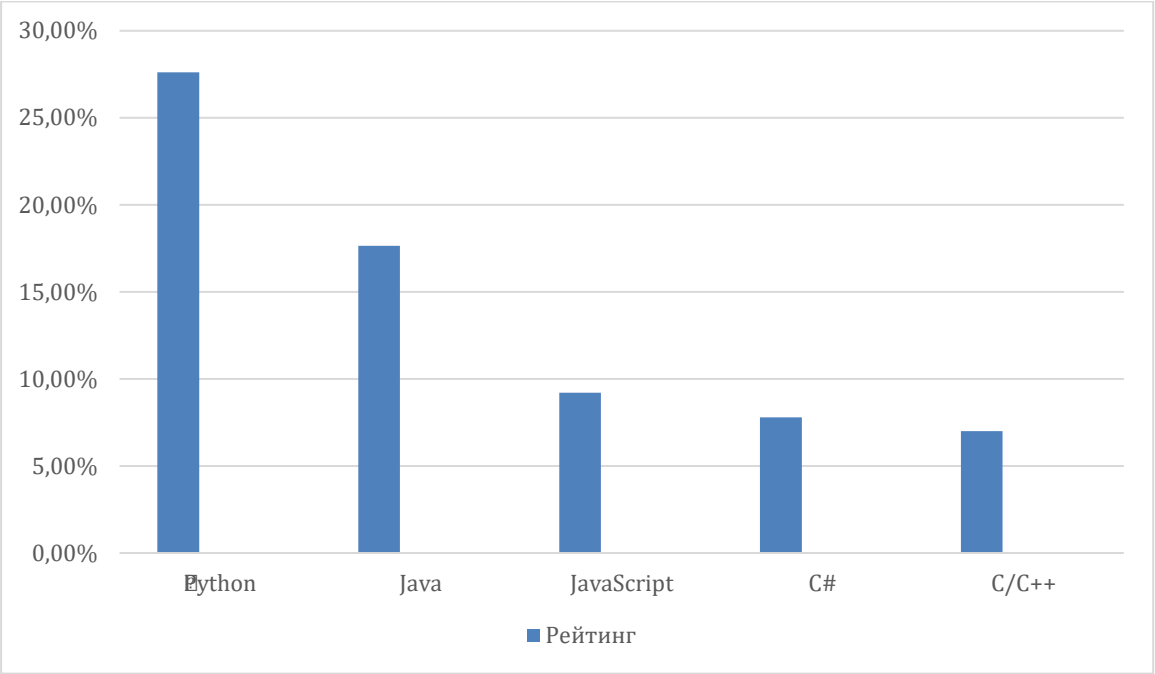

Рисунок 1.1 — Топ найкращих мов програмування за статистикою PYPL Index

Java була побудована з філософією «напиши один раз, запусти будь-де» (WORA). Javacode, який ми пишемо на одній платформі (операційній системі), працюватиме на інших платформах без жодних змін. Для запуску Java використовується абстрактна машина під назвою Java Virtual Machine (JVM). JVM виконує байт-код Java. Потім центральний процесор виконує JVM. Оскільки всі JVM працюють однаково, той самий код працює і в інших операційних системах, що робить Java незалежною від платформи.

Існують різні стилі програмування. Об'єктно-орієнтований підхід є одним із популярних стилів програмування. В об’єктно-орієнтованому програмуванні складну задачу поділяють на менші набори шляхом створення об’єктів. Це робить наш код придатним для багаторазового використання, має переваги дизайну та полегшує обслуговування коду. Багато мов програмування, зокрема Java, Python і C++, мають об’єктно-орієнтовані функції [3].

Попередні версії Java критикували за повільність. Однак зараз справи йдуть зовсім інакше. Нові JVM значно швидші. І процесор, який виконує JVM, також стає все потужнішим. Зараз Java є однією з найшвидших мов

програмування. Добре оптимізований код Java майже такий же швидкий, як мови нижчого рівня, такі як C/C++, і набагато швидший, ніж Python, PHP тощо.

Платформа Java надає різні функції для захисту програм Java. Нижче наведено високорівневі функції, які підтримує Java:

— забезпечує безпечну платформу для розробки та запуску програм;

— має автоматичне керування пам'яттю, зменшує пошкодження пам'яті та вразливості;

— забезпечує безпечний зв'язок, захищаючи цілісність і конфіденційність переданих даних.

Однією з причин широкого використання Java є наявність величезної стандартної бібліотеки. Середовище Java має сотні класів і методів у різних пакетах, щоб допомогти розробникам програмного забезпечення. Наприклад, java.lang — для розширених функцій рядків, масивів тощо. java.util — для структур даних, регулярних виразів, функцій дати та часу тощо. java.io — для файлового введення/виведення, обробки винятків тощо [4].

Отже можна виділити такі особливості мови програмування Java:

— не залежить від платформи;

— об'єктно-орієнтована мова;

— швидка;

— безпечна;

— велика стандартна бібліотека.

1.2. Фреймворк Spring

Spring Framework — це фреймворк із відкритим вихідним кодом, який забезпечує підтримку інфраструктури для розробки програм Java. Один із найпопулярніших фреймворків Java Enterprise Edition (Java EE), Spring допомагає розробникам створювати високопродуктивні програми, використовуючи звичайні старі об’єкти Java (POJO).

Фреймворк — це велика частина попередньо визначеного коду, до якої розробники можуть додавати код для вирішення проблеми в певній області.

Існує багато популярних фреймворків Java, включаючи Java Server Faces (JSF), Maven, Hibernate, Struts і Spring [5].

Spring Framework, випущений у червні 2003 року Родом Джонсоном за ліцензією Apache 2.0, розміщено на SourceForge.

Дізнавшись визначення Spring Framework, розглянемо далі переваги його використання. До них можна віднести наступне:

— spring дозволяє розробникам розробляти програми корпоративного класу за допомогою POJO;

— spring організована за модульним принципом;

— spring використовує деякі з існуючих технологій, як-от кілька фреймворків ORM, фреймворків журналювання, таймерів JEE, Quartz і JDK та інших технологій перегляду;

— тестування програми, написаної за допомогою Spring, просте, оскільки код, що залежить від середовища, переміщується в цю структуру;

— добре розроблений веб-фреймворк MVC, який надає чудову альтернативу веб-фреймворкам, таким як Struts або іншим переробленим або менш популярним веб-фреймворкам

— spring надає зручний API для перекладу специфічних для технології винятків (наприклад, JDBC, Hibernate або JDO) у узгоджені, неперевірені винятки;

— легкі контейнери IoC мають тенденцію бути легкими, особливо в порівнянні, наприклад, з контейнерами EJB, що корисно для розробки та розгортання програм на комп’ютерах з обмеженою пам’яттю;

— spring забезпечує узгоджений інтерфейс керування транзакціями, який можна масштабувати до локальної транзакції і масштабувати до глобальних транзакцій [6].

Перевага використання лише POJO полягає в тому, що нам не потрібен продукт-контейнер EJB, наприклад сервер додатків, але ми можемо використовувати лише надійний контейнер сервлетів, наприклад Tomcat, або інший комерційний продукт.

Незважаючи на те, що кількість пакунків і класів є значною, доведеться турбуватися лише про ті, які потрібні, та ігнорувати решту.

Крім того, за допомогою JavaBeanstyle POJO стає легше використовувати ін’єкцію залежностей для ін’єкції тестових даних.

Якщо говорити коротко про Spring Framework, то це просто контейнер для впровадження залежностей. Це допомагає швидше та зручніше створювати програму Java. Впровадження залежностей — це техніка програмування, яка робить клас незалежним від його залежностей. Це досягається шляхом відокремлення використання об’єкта від його створення. Це допомагає дотримуватися принципів інверсії залежностей і єдиної відповідальності SOLID [7].

Розглянемо детально Рисунок 1.2 на якому зображено архітектуру Spring Framework.

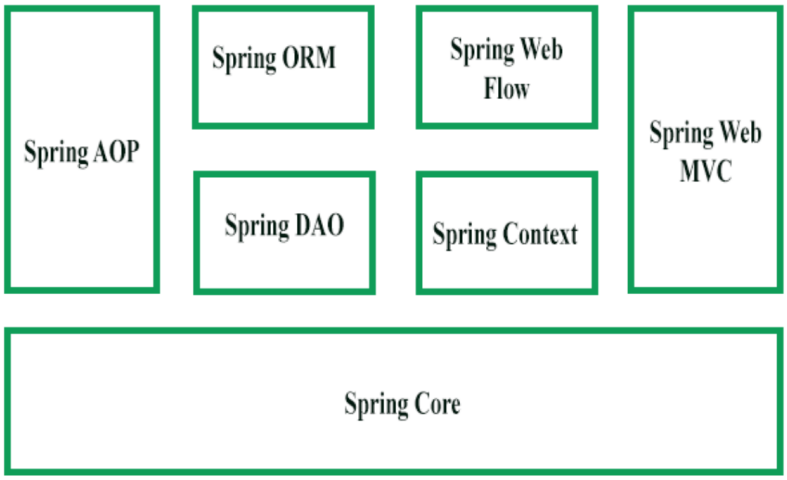

Рисунок 1.2 — Архітектура Spring Framework

Як ми бачимо на Рисунку 1.2 Spring Framework складається із семи модулів. Це модулі — Spring Core, Spring AOP, Spring Web MVC, Spring DAO, Spring ORM, Spring context і Spring Web flow. Ці модулі надають різні платформи для розробки різних корпоративних програм; наприклад, ми можемо

використовувати модуль Spring Web MVC для розробки програм на основі MVC [7].

1.2.1. Spring Core Module

Він є основним компонентом фреймворку Spring, надає контейнер IoC. Існує два типи реалізацій контейнера Spring, а саме фабрика компонентів і контекст програми. Фабрика компонентів визначається за допомогою інтерфейсу org.springframework.beans.factory. BeanFactory і діє як контейнер для компонентів. Фабричний контейнер Bean дозволяє відокремити конфігурацію та специфікацію залежностей від логіки програми. У Spring framework фабрика Bean діє як центральний контейнер IoC, який відповідає за створення екземплярів об’єктів програми. Він також налаштовує та збирає залежності між цими об’єктами. Існує багато реалізацій інтерфейсу BeanFactory. Клас XmlBeanFactory є найпоширенішою реалізацією інтерфейсу BeanFactory. Це дозволяє виразити об’єкт для створення програми та усунути взаємозалежності між об’єктами програми.

1.2.2. Spring AOP Module

Подібно до об’єктно-орієнтованого програмування (ООП), яке розбиває програми на ієрархію об’єктів, AOP розбиває програми на аспекти або проблеми. Модуль Spring AOP дозволяє реалізувати проблеми або аспекти в програмі Spring у Spring AOP, аспектами є звичайні компоненти Spring або звичайні класи, анотовані анотацією @Aspect. Ці аспекти допомагають керувати транзакціями та відстежувати помилки програми. Наприклад, керування транзакціями необхідне в банківських операціях, таких як переказ суми з одного рахунку на інший. Модуль Spring AOP забезпечує рівень абстракції керування транзакціями, який можна застосувати до API транзакцій.

1.2.3. Spring ORM Module

Модуль Spring ORM використовується для доступу до даних із баз даних у програмі. Він надає API для роботи з базами даних за допомогою JDO, Hibernate та iBatis. Spring ORM підтримує DAO, що забезпечує зручний спосіб створення наступних ORM-рішень на основі DAO:

— просте декларативне керування транзакціями;
— прозора обробка винятків;
— потокобезпечні, легкі шаблонні класи;
— класи підтримки DAO;
— управління ресурсами.

1.2.4. Spring Web MVC Module

Модуль Web MVC Spring реалізує архітектуру MVC для створення веб-додатків. Він розділяє код моделі та компонентів представлення веб-додатку. У Spring MVC, коли запит генерується з браузера, він спочатку надходить до класу DispatcherServlet (Front Controller), який відправляє запит до контролера (класу SimpleFormController або класу AbstractWizardformController), використовуючи набір відображень обробників. Контролер витягує та обробляє інформацію, вбудовану в запит, і надсилає результат до класу DispatcherServlet у формі об’єкта моделі. Нарешті, клас DispatcherServlet використовує класи ViewResolver для надсилання результатів у представлення, яке відображає ці результати користувачам [8].

1.2.5. Spring Web Flow Module

Модуль Spring Web Flow є розширенням модуля Spring Web MVC. Інфраструктура Spring Web MVC надає контролери форми, такі як клас SimpleFormController і клас AbstractWizardFormController, для реалізації попередньо визначеного робочого процесу. Spring Web Flow допомагає визначити файл XML або клас Java, який керує робочим процесом між різними сторінками веб-програми. Розглянемо наступні переваги Spring Web Flow:

— потік між різними інтерфейсами користувача програми чітко забезпечується визначенням веб-потоку в XML-файлі;

— визначення веб-потоку допомагають віртуально розділити програму на різні модулі та повторно використовувати ці модулі в багатьох ситуаціях.

1.2.6. Spring Web DAO Module

Пакет DAO у структурі Spring забезпечує підтримку DAO за допомогою технологій доступу до даних, таких як JDBC, Hibernate або JDO. Цей модуль представляє рівень абстракції JDBC, усуваючи потребу в наданні виснажливого кодування JDBC. Він також надає програмні та декларативні класи керування транзакціями. Пакет Spring DAO підтримує гетерогенне підключення до бази даних Java і відображення O/R, що допомагає Spring працювати з кількома технологіями доступу до даних.

Для легкого та швидкого доступу до ресурсів бази даних платформа Spring надає абстрактні базові класи DAO. Для кожної технології доступу до даних, яка підтримується фреймворком Spring, доступні кілька реалізацій. Наприклад, у JDBC клас JdbcDaoSupport і його методи використовуються для доступу до примірника DataSource і попередньо налаштованого примірника JdbcTemplate. Потрібно буде просто розширити клас JdbcDaoSupport і надати зіставлення з фактичним екземпляром DataSource у конфігурації контексту програми, щоб отримати доступ до програми на основі DAO.

1.2.7. Spring Application Context Module

Контекстний модуль Spring Application базується на модулі Core. Контекст програми org.springframework.context.ApplicationContext є інтерфейсом BeanFactory. Цей модуль походить від пакета org.springframework.beans, а також підтримує такі функції, як інтернаціоналізація (I18N), перевірка, розповсюдження подій і завантаження ресурсів. Контекст програми реалізує інтерфейс MessageSource і надає програмі функції обміну повідомленнями[8].

Коротко кажучи, Spring framework — це проста, легка та надійна структура для створення сучасних додатків на основі Java. Він надає багато функціональних можливостей у модульній формі, і ви можете використовувати ті, які вам дійсно потрібні, замість того, щоб тягнути їх усі.

1.3. Spring Security

Spring Security — це фреймворк, який надає різні функції безпеки, такі як: автентифікація, авторизація для створення захищених програм Java Enterprise. Це підпроект Spring framework, який розпочав у 2003 році Бен Алекс. Пізніше, у 2004 році, він був випущений під ліцензією Apache як Spring Security 2.0.0. Він долає всі проблеми, які виникають під час створення додатків із безпекою, а також керує новим серверним середовищем для програми. Цей фреймворк націлений на дві основні сфери застосування: автентифікацію та авторизацію.

Автентифікація – це процес визначення та ідентифікації користувача, який хоче отримати доступ.

Авторизація — це процес, який дозволяє повноваженням виконувати дії в програмі.

Ми можемо застосувати авторизацію для веб-запиту, методів і доступу до окремого домену [9].

Spring Security Framework підтримує широкий спектр моделей автентифікації. Ці моделі або надаються третіми сторонами, або сама структура. Spring Security підтримує інтеграцію з усіма цими технологіями.

— заголовки автентифікації HTTP BASIC;
— заголовки автентифікації дайджесту HTTP;
— обмін сертифікатами клієнта HTTP X.509;
— LDAP (протокол легкого доступу до каталогу);
— автентифікація на основі форми;
— автентифікація OpenID;
— автоматична автентифікація "запам'ятати мене";
— kerberos;

— JOSSO (єдиний вхід з відкритим вихідним кодом Java);

— AppFuse;

— AndroMDA;

— Mule ESB;

— DWR (прямий веб-запит)

Принадність цього фреймворку полягає в його гнучкій природі автентифікації для інтеграції з будь-яким програмним рішенням. Іноді розробники хочуть інтегрувати його із застарілою системою, яка не відповідає жодним стандартам безпеки, тут Spring Security працює добре.

Розглянемо далі функції Spring Security [10]. До них належать такі:

— LDAP - це відкритий прикладний протокол для підтримки та доступу до інформаційних служб розподіленого каталогу через Інтернет-протокол;

— єдиний вхід - ця функція дозволяє користувачеві отримати доступ до кількох програм за допомогою одного облікового запису;

— JAAS LoginModule - це Pluggable Authentication Module, реалізований на Java;

— базова аутентифікація доступу яка використовується для надання імені користувача та пароля під час виконання запиту через мережу;

— автентифікація дайджест-доступу - ця функція дозволяє нам зробити процес аутентифікації більш безпечним, ніж автентифікація з базовим доступом;

— автентифікація веб-форм обліковими даними користувача з веб-браузера;

— авторизація дозволяє розробникам визначати політики доступу до ресурсів;

— локалізація ПЗ, що дозволяє нам створювати інтерфейс користувача будь-якою мовою;

— HTTP авторизація.

Spring security має архітектуру, яка відокремлює автентифікацію від авторизації, і має стратегії та точки розширення для обох. Підсистема автентифікації відповідає за дійсність облікових даних клієнта. Наприклад, ім'я

користувача або пароль. Нижче на рисунку 1.3 наведено схему архітектури Spring Security.

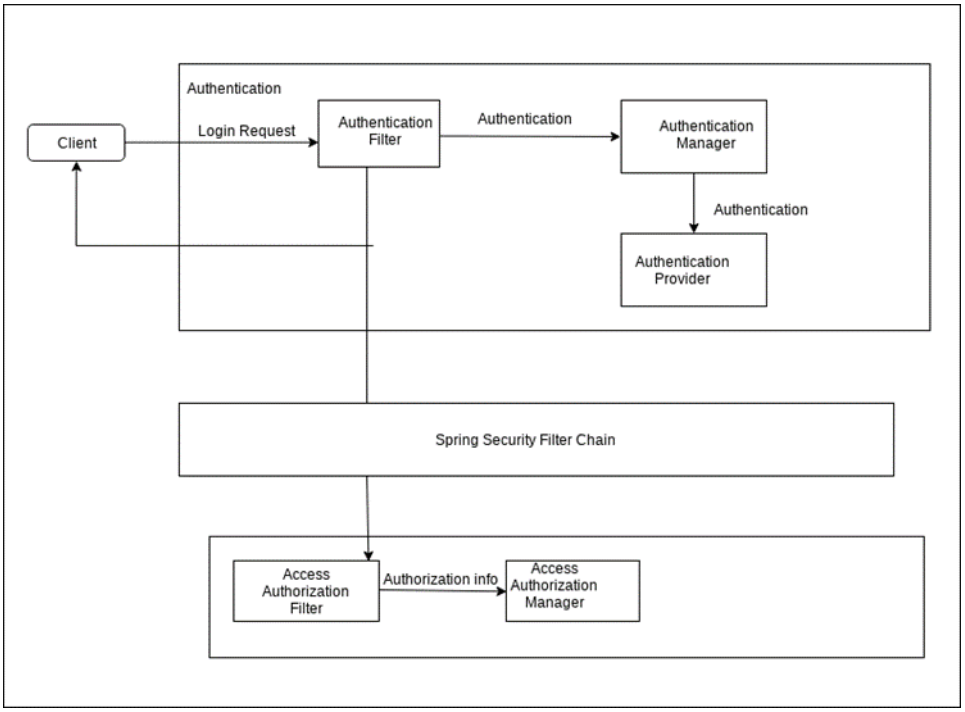

Рисунок 1.3 — Архітектура Spring Security

Те, що наведено на Рисунку 1.3 допомагає нам отримати уявлення про архітектуру Spring Security та її роботу. Система надає різні методи безпеки для веб-додатків, застосовуючи конфігурації Java. Це свого роду фреймворк Java, який допомагає впроваджувати додатки корпоративного рівня. Крім того, кожна веб-програма потребує належного захисту, оскільки вона, як правило, відкривається різними користувачами через різні браузери. Крім того, він пропонує різноманітні функції та переваги, що дозволяє веб-додаткам захищатися від зовнішніх загроз, якщо такі виникають.

1.4. JWT токен

JWT або веб-токени JSON найчастіше використовуються для ідентифікації автентифікованого користувача. Вони видаються сервером автентифікації та споживаються клієнт-сервером (для захисту його API).

Веб-токен JSON — це відкритий галузевий стандарт, який використовується для обміну інформацією між двома об’єктами, як правило, клієнтом (наприклад, зовнішнім інтерфейсом програми) і сервером (сервером програми).

Вони містять об’єкти JSON, які містять інформацію, якою потрібно поділитися. Кожен JWT також підписується за допомогою криптографії (хешування), щоб гарантувати, що вміст JSON (також відомий як претензії JWT) не може бути змінений клієнтом або зловмисною стороною [11].

Наприклад, коли ми входимо в обліковий запис Google, Google видає JWT, який містить такі претензії JSON, які зображені на рисунку 1.4.

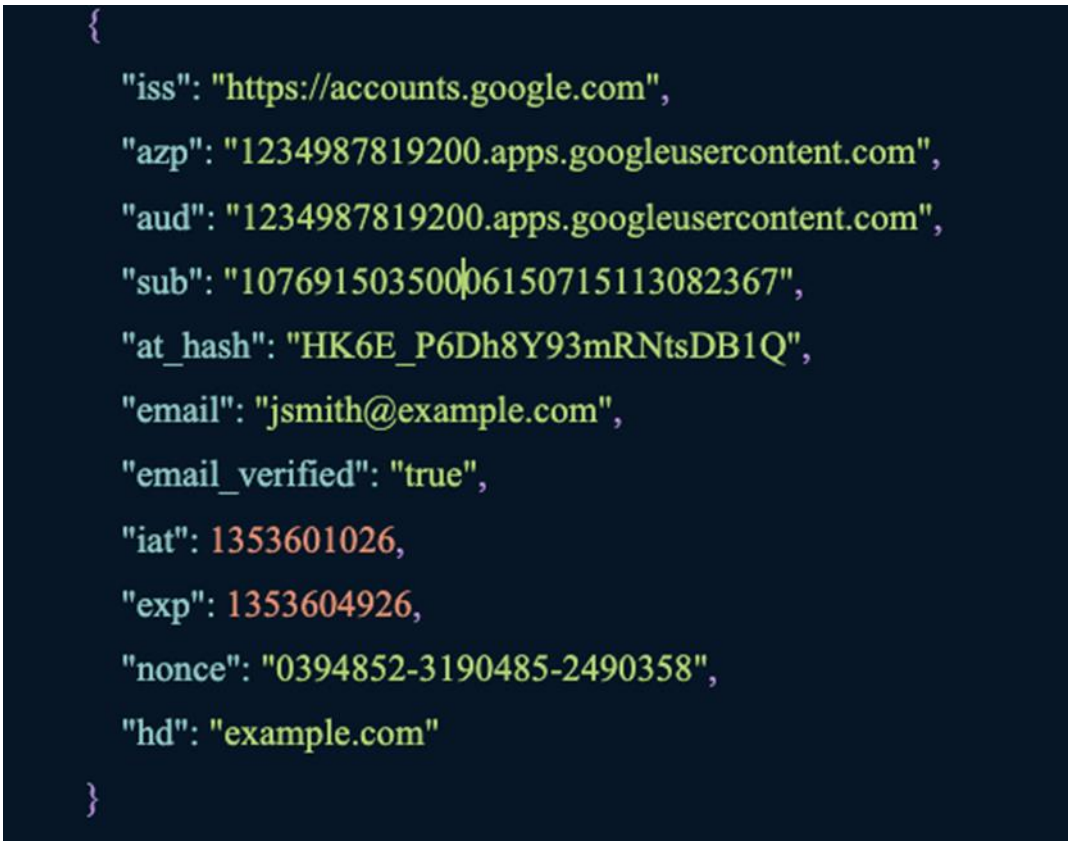

Рисунок 1.4 — Приклад JWT

Отже, використовуючи наведену вище інформацію на рисунку 1.4, клієнтська програма, яка використовує вхід за допомогою Google, точно знає, хто є кінцевим користувачем.

Тоді виникає таке питання, чому сервер аутентифікації не може просто надіслати інформацію як звичайний об’єкт JSON і чому йому потрібно перетворити її на «токен».

Якщо сервер автентифікації надсилає його як звичайний JSON, API клієнтської програми не матимуть можливості перевірити, чи вміст, який вони отримують, правильний. Зловмисник може, наприклад, змінити ідентифікатор користувача, і API програми не зможуть дізнатися, що це сталося.

Через цю проблему безпеки сервер автентифікації повинен передавати цю інформацію таким чином, щоб її могла перевірити клієнтська програма, і саме тут з’являється концепція «токена».

Простіше кажучи, токен — це рядок, який містить деяку інформацію, яку можна безпечно перевірити. Це може бути випадковий набір буквено-цифрових символів, які вказують на ідентифікатор у базі даних, або це може бути закодований JSON, який клієнт може самостійно перевірити (відомий як JWT) [11].

Надалі розглянемо структуру JWT, яка складається з кількох частин. Заголовок (Header): складається з алгоритму підпису, який використовується та типу токена, який в даному випадку здебільшого є «JWT». Корисне навантаження (Payload): корисне навантаження містить претензії або об’єкт JSON. Підпис (Signature): Рядок, створений за допомогою криптографічного алгоритму, який можна використовувати для перевірки цілісності корисного навантаження JSON.

Коротше кажучи, JWT використовуються як безпечний спосіб автентифікації користувачів і обміну інформацією.

Як правило, закритий ключ, або секрет, використовується емітентом для підпису JWT. Одержувач JWT перевірить підпис, щоб переконатися, що токен не було змінено після того, як його підписав емітент. Неавтентифікованим джерелам важко вгадати ключ підпису та спробувати змінити претензії в JWT.

Дана структура може виглядати так, як показано на рисунку 1.5.

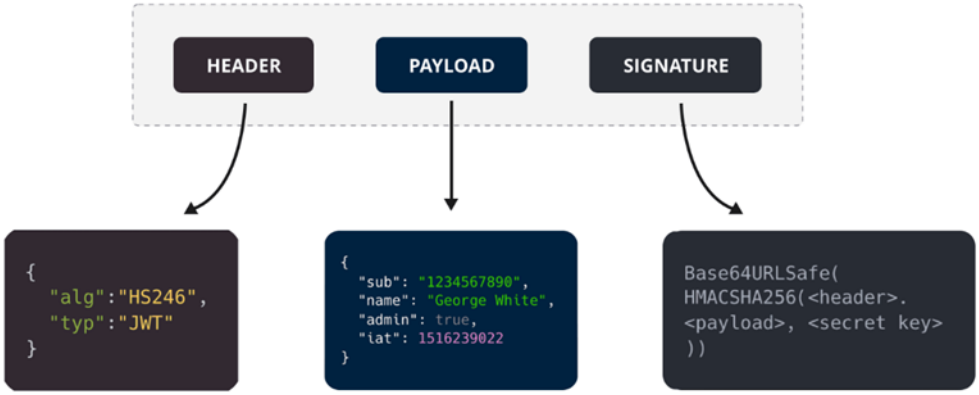

Рисунок 1.5 — Структура JWT

Однак не всі алгоритми підписання однакові. Наприклад, деякі алгоритми підписання використовують секретне значення, яке спільно використовується між емітентом і стороною, яка перевіряє JWT. Інші алгоритми використовують відкритий і закритий ключ. Закритий ключ відомий лише емітенту, тоді як відкритий ключ може бути широко поширеним. Відкритий ключ можна використовувати для перевірки підпису, але лише закритий ключ можна використовувати для створення підпису. Це безпечніше, ніж спільний секрет, оскільки закритий ключ має існувати лише в одному місці.

Через це серверу не потрібно зберігати базу даних з інформацією, необхідною для ідентифікації користувача. Для розробників це чудова новина — сервер, який видає JWT, і сервер, який перевіряє його, не повинні бути однаковими.

Використання JWT має переваги порівняно з простими веб-токенами (SWT) і маркерами мови розмітки безпеки (SAML). До його переваг можна віднести наступне:

— більш компактний - JSON є менш детальним, ніж XML, тому, коли він закодований, JWT менший, ніж маркер SAML;

— більш безпечний - JWT можуть використовувати для підпису пару відкритих/приватних ключів у формі сертифіката X.509, JWT також може бути симетрично підписаний спільним секретом за допомогою алгоритму HMAC;

— більш поширений - аналізатори JSON поширені в більшості мов програмування, оскільки вони відображаються безпосередньо на об’єкти;

— легший в обробці - JWT використовується в масштабі Інтернету, його легше обробляти на пристроях користувача, особливо мобільних.

Якщо говорити про безпеку JWT, то веб-підпис JSON (JWS) за своєю суттю не забезпечує безпеки. Безпека або вразливість служби випливає з реалізації. Найпоширеніші вразливості, створені реалізацією JWT/JWS, легко уникнути, але не завжди очевидні. Деякі з найпоширеніших вразливостей виникають через реалізації, які дозволяють значенням заголовка JWT керувати перевіркою JWT.

Якщо функція обробки JWT сліпо дотримується типу алгоритму, оголошеного в заголовку, а для заголовка «alg» встановлено значення «none», JWT буде оброблено як дійсний без будь-якого захисту, який забезпечує підпис. Це дозволяє агентам-атакам встановлювати значення вимог у тілі JWT відповідно до їхніх цілей. Захищена реалізація функції обробки допустиме лише строгий набір значень заголовків і перевірятиме лише JWT, який відповідає цим обмеженням. На даний момент це добре відомий експлойт, і більшість сучасних бібліотек JWT і допоміжних функцій більше не повинні бути вразливими до нього.

Навіть коли JWT підписано, все одно є деякі підводні камені, про які слід знати. Існує три основні категорії алгоритмів підписів, які широко використовуються: алгоритм «немає», симетричні алгоритми та асиметричні алгоритми. Якщо послуга, яка використовує JWT/JWS, дозволяє поєднання різних алгоритмів, функція обробки повинна захищатися від загального експлойту, що використовується проти асинхронних реалізацій JWS.

Розглянемо приклад двох алгоритмів шифрування, коду автентифікації повідомлення на основі хешу (HMAC) і Рівест-Шаміра-Адлемана (RSA). HMAC використовує спільний секретний ключ для формування та перевірки підпису. RSA, асиметричний алгоритм, використовує закритий ключ для формування підпису та відкритий ключ для перевірки підпису. Система, яка використовує

алгоритм RSA, підписуватиме приватний ключ і перевірятиме відкритий ключ. Якщо JWT було підписано за допомогою алгоритму HMAC і загальнодоступного ключа RSA як вхідних даних, то систему-одержувач можна обдурити, спробувавши використати наявний відкритий ключ RSA як спосіб перевірки підпису за допомогою алгоритму HMAC. Цей вид атаки працює, коли система-одержувач дозволяє заголовку JWT управляти автентифікацією маркера. У цьому випадку зловмисник оголосив відкритий ключ асиметричної схеми спільним секретним ключем схеми симетричного підпису [11].

Отже, при ретельному розгляді використання JWT може бути легким методом керування та захисту ресурсів

1.5. Мова розмітки гіпертексту HTML

HTML означає мову розмітки гіпертексту. Він складається з ключових слів і команд, які веб-дизайнери використовують для створення веб-сайтів.

Гіпертекст — це текст із посиланнями, на які читачі можуть просто клацнути, щоб перейти на іншу сторінку чи іншу частину сторінки. Тим часом мова розмітки використовує теги або звичайний текст зі спеціальними позначками для визначення розділів сторінки, таких як верхні та нижні колонтитули, та інших елементів, включаючи таблиці та зображення.

HTML вважається одним із трьох основних інструментів у створенні веб-сторінок: HTML забезпечує структуру або те, як текст, зображення тощо відображатимуться на веб-сайті. CSS (каскадні таблиці стилів) встановлює візуальні властивості цих елементів, такі як кольори, формат і макет.

Щоб написати HTML, знадобиться текстовий редактор, наприклад Notepad, Brackets або Atom. Редактори HTML гарантують, що код чистий і функціональний. Вони допомагають зменшити кількість помилок шляхом автоматичної вставки тегів та інших загальних елементів або через налагодження.

HTML є мовою за замовчуванням для веб-сайтів і веб-документів. Це допомагає браузеру зрозуміти структуру та стиль документа або файлів для

перегляду в Інтернеті. Це дозволяє веб-сторінкам розміщувати аудіо, відео, електронні таблиці та інші програми. Це також полегшує навігацію всередині веб-сторінок або між веб-сайтами за допомогою гіпертексту.

Крім того, виробники веб-сайтів можуть використовувати HTML для створення форм для замовлення продуктів, бронювання або пошуку інформації. Таким чином, HTML є основним будівельним блоком для побудови бренду та ведення сайту електронної комерції або онлайн-бізнесу на основі передплати.[4]

Основними компонентами документа HTML є теги та елементи. Вони повідомляють браузеру, як відображати ваш вміст. Теги починаються і закінчуються кутовими дужками або знаками «менше» і «більше». Букви між ними називаються вмістом елемента.

Декларація типу документа (DTD) <!DOCTYPE html> відображається на початку або в самому верху документа HTML. Він повідомляє браузеру, яка версія HTML була використана для створення сторінки.

Кореневий елемент HTML <html>, який записаний під DTD, діє як "основний контейнер", який містить усі інші елементи. Він може вказати мову документа HTML. Наприклад, <html lang="en-US"> означає, що сторінка написана американською англійською.

Head <head>, який можна знайти між <html> і <body>, містить метадані, що описують інформацію про сторінку. Вони включають:

— <title> або загальна тема веб-сторінки, він окремий від тегу заголовка, який відображається в тілі, але має збігатися;

— <style> визначає, як елементи мають відображатися у браузері - це стосується кольору заголовка, вирівнювання тексту, кольору фону основної частини тощо;

— <link> вказує на ресурси (тобто іншу веб-сторінку або зовнішню таблицю стилів), пов’язані зі сторінкою HTML;

— <meta> містить ключові слова, автора та опис сторінки;

— <base> посилається на URL-адресу за умовчанням.

— Body <body> - це основна частина документа, яка містить інформацію, яку браузер відображає на екрані. Він може включати наступне:

— заголовок містить назву сайту, логотип, основну навігацію та панель пошуку;

— основний вміст охоплює назву або заголовок статті, зміст статті, дату публікації, автора тощо;

— бічні панелі відображають віджети та вторинну навігацію, наприклад архіви за категоріями чи датами;

— нижній колонтитул пропонує контактну інформацію, соціальні посилання, авторські права та третинну навігацію.

HTML є легкою та простою мовою, яку можна легко зрозуміти та змінити. Тому розглянемо переваги HTML, до них можна віднести наступні пункти:

— з HTML дуже легко зробити ефективну презентацію, оскільки він має багато тегів форматування;

— це мова розмітки, тому вона забезпечує гнучкий спосіб дизайну веб-сторінок разом із текстом;

— це полегшує програмістам додавати посилання на веб-сторінки (за тегом прив’язки html), що підвищує інтерес користувача до перегляду;

— він не залежить від платформи, оскільки його можна відображати на будь-якій платформі, як-от Windows, Linux, Macintosh тощо;

— це полегшує програмісту додавати графіку, відео та звук до веб-сторінок, що робить їх більш привабливими та інтерактивними;

— HTML є мовою, яка не враховує регістр, що означає, що ми можемо використовувати теги як у нижньому, так і у верхньому регістрі.

Також можна виділити і недоліки HTML хоч їх і не так багато. До них ми можемо віднести наступне:

— HTML може створювати лише статичні веб-сторінки, для динамічних веб-сторінок потрібно використовувати інші мови;

— щоб створити просту веб-сторінку, необхідно написати велику кількість коду;

— функція безпеки погана [12].

Отже, HTML є основою веб-сторінок, використовується для розробки веб-сторінок шляхом структурування веб-сайтів і веб-програм.

1.6. Каскадні таблиці стилів CSS

Каскадні таблиці стилів, які люблять називати CSS, — це просто розроблена мова, призначена для спрощення процесу створення веб-сторінок, які виглядають презентабельно. CSS дозволяє застосовувати стилі до веб-сторінок. Що ще важливіше, CSS дозволяє робити це незалежно від HTML-коду, з якого складається кожна веб-сторінка. Він описує, як має виглядати веб-сторінка: він визначає кольори, шрифти, інтервали та багато іншого. Коротше кажучи, ми можемо зробити так, щоб наш веб-сайт виглядав як завгодно. CSS дозволяє розробникам і дизайнерам визначати його поведінку, зокрема те, як елементи розташовуються у браузері.

Тоді як HTML використовує теги, CSS використовує набори правил. CSS простий у вивченні та розумінні, але він забезпечує потужний контроль над представленням документа HTML [13].

Розглянемо наступні переваги CSS:

— CSS економить час: ми можемо написати CSS один раз і повторно використовувати той самий аркуш на кількох HTML-сторінках;

— просте обслуговування: щоб внести глобальні зміни, можна просто змінити стиль, і всі елементи на всіх веб-сторінках оновлюються автоматично;

— пошукові системи: CSS вважається чистою технікою кодування, що означає, що пошуковим системам не доведеться намагатися «прочитати» його вміст;

— стилі кращі за HTML: CSS має набагато ширший набір атрибутів, ніж HTML, тому ми можемо надати своїй HTML-сторінці набагато кращий вигляд порівняно з атрибутами HTML;

— офлайн-перегляд: CSS може зберігати веб-додатки локально за допомогою офлайн-кешу, за допомогою цього ми можемо переглядати офлайн-сайти.

Основні характеристики CSS включають правила стилю, які інтерпретуються браузером клієнта та застосовуються до різних елементів у вашому документі.

Правило стилю складається з компонента селектора та компонента блоку оголошення. Селектор використовується для вказівки на компонент HTML, до якого потрібно додати стиль. Усередині блоку оголошень міститься одна або кілька декларацій разом із крапкою з комою. Кожне розміщене оголошення має назву властивості CSS, крапку з комою та значення. Наприклад, колір є властивістю, а значення має червоний колір. Розмір шрифту є властивістю, а 15 пікселів є значенням.

Оголошення CSS закінчується крапкою з комою, а ці блоки оточені фігурними дужками. Селектори CSS – це ті, які використовуються для пошуку елементів HTML, заснованих на імені елемента, ідентифікаторі, атрибуті, класі тощо. Один унікальний елемент буде вибрано за ідентифікатором елемента. Якщо ви бажаєте вибрати окремий елемент із певним ідентифікатором, слід використати функцію #разом із атрибутом id. Якщо ви бажаєте виділити елементи з певним класом, разом з іменем класу слід написати крапку. Універсальний селектор: якщо ви не зацікавлені у виборі елементів певного типу, універсальний селектор просто відповідає імені елемента.

Селектор елемента: ці селектори вибирають елемент на основі назви елемента. Селектор нащадка: коли окремий елемент лежить всередині іншого елемента, він називається селектором нащадка. Селектор ID: цей селектор використовує ідентифікатор елемента HTML, щоб можна було вибрати певний елемент. Селектори класу: вибирає елемент із певним атрибутом класу.

Групування селекторів: добре буде згрупувати селектори, щоб мінімізувати код. Кожен селектор разом із комою слід використовувати для групування селекторів [13].

Отже, CSS надає можливість веб-дизайнеру, щоб можна було внести значні зміни у веб-макет усіх сторінок одного веб-сайту, використовуючи лише один файл. Це допомагає створити легкий і креативний веб-сайт із високою швидкістю реагування, який справляє враження на аудиторію під час показу. Тому це невід'ємна частина сучасних веб-сайтів, яку не можна забувати.

1.7. Мова програмування JavaScript

JavaScript — це легка, кросплатформна та інтерпретована мова програмування, яка також відома як мова сценаріїв для веб-сторінок. Він добре відомий розробкою веб-сторінок, його також використовують багато середовищ без браузерів. JavaScript можна використовувати як для розробки на стороні клієнта, так і для розробки на стороні сервера. JavaScript є одночасно імперативним і декларативним типом мови. JavaScript містить стандартну бібліотеку об'єктів, наприклад Array, Date і Math, а також основний набір елементів мови, як-от оператори, керуючі структури та оператори.

На стороні клієнта: надає об'єкти для керування браузером і його об'єктною моделлю документа (DOM). Наприклад, розширення на стороні клієнта дозволяють програмі розміщувати елементи у формі HTML і реагувати на події користувача, такі як клацання мишею, введення форми та навігація сторінкою. Корисними бібліотеками для клієнта є AngularJS, ReactJS, VueJS та багато інших.

На стороні сервера: він надає об'єкти, що стосуються запуску JavaScript на сервері. Наприклад, якщо розширення на стороні сервера дозволяють додатку спілкуватися з базою даних і забезпечують безперервність інформації від одного виклику програми до іншого або виконують маніпуляції з файлами на сервері. Корисний фреймворк, який сьогодні є найвідомішим, — це node.js[14].

Імперативна мова – у цьому типі мови ми здебільшого турбуємося про те, як це зробити. Вона просто контролює потік обчислень. Підхід до процедурного програмування, об'єктно-орієнтований підхід, відноситься до цього.

Декларативне програмування – у цьому типі мови ми дбаємо про те, як це зробити, в основному тут потрібні логічні обчислення. Тут головна мета — описати бажаний результат без прямого диктування про те, як його отримати, як це робить функція зі стрілкою.

Згідно з нещодавнім опитуванням, проведеним Stack Overflow, JavaScript є найпопулярнішою мовою у світі.

Завдяки прогресу в технології браузера та JavaScript, який перейшов на сервер із Node.js та іншими фреймворками, JavaScript здатний на набагато більше. Ось кілька речей, які ми можемо робити за допомогою JavaScript:

— JavaScript був створений в першу чергу для маніпулювання DOM, попередні веб-сайти були переважно статичними, після створення JS почали створювати динамічні веб-сайти;

— функції в JS є об’єктами, вони можуть мати властивості та методи, як і інший об’єкт та їх можна передати як аргументи в інші функції;

— може працювати з датою та часом;

— виконує перевірку форм, хоча форми створюються за допомогою HTML;

— компілятор не потрібен.

JavaScript вважається легким через те, що він мало використовує ЦП, його легко реалізувати та має мінімалістичний синтаксис. Мінімалістичний синтаксис не має типів даних. Тут усе трактується як об’єкт. Його дуже легко вивчити, оскільки його синтаксис схожий на C++ і Java [14].

Полегшена мова не споживає багато ресурсів ЦП. Це не створює надмірного навантаження на процесор або оперативну пам’ять. JavaScript працює у браузері, навіть якщо він має складні парадигми та логіку, що означає, що він використовує менше ресурсів, ніж інші мови. Наприклад, NodeJs, різновид JavaScript, не тільки виконує швидші обчислення, але й використовує менше ресурсів, ніж його аналоги, такі як Dart або Java.

Крім того, порівняно з іншими мовами програмування, вона має менше вбудованих бібліотек або фреймворків, що є ще однією причиною її легкості.

Однак це призводить до недоліку, оскільки нам потрібно включати зовнішні бібліотеки та фреймворки.

Сьогодні JavaScript може виконуватися не тільки в браузері, але і на сервері, або фактично на будь-якому пристрої, який має спеціальну програму під назвою JavaScript engine.

Браузер має вбудований механізм, який іноді називають «віртуальною машиною JavaScript». Також різні двигуни мають різні «кодові імена» [14].

1.8. Фреймворк React

У наш час світ не може жити без мобільних і веб-додатків. Все оцифровано, від бронювання таксі до замовлення їжі для здійснення банківських операцій. Завдяки ефективним структурам, які забезпечують безперебійну роботу користувача. Однією з таких надійних інтерфейсних бібліотек є React.

React — це бібліотека розробки інтерфейсу користувача на основі JavaScript. Facebook і спільнота розробників з відкритим кодом керують ним. Хоча React є бібліотекою, а не мовою, він широко використовується у веб-розробці. Бібліотека вперше з’явилася в травні 2013 року і зараз є однією з найбільш часто використовуваних інтерфейсних бібліотек для веб-розробки.

React пропонує різні розширення для підтримки всієї архітектури програми, наприклад Flux і React Native, крім простого інтерфейсу користувача [15].

Популярність React сьогодні затьмарила популярність усіх інших інтерфейсних фреймворків розробки. Просте створення динамічних додатків: React полегшує створення динамічних веб-додатків, оскільки вимагає менше кодування та пропонує більше функціональних можливостей, на відміну від JavaScript, де кодування часто стає складним дуже швидко. Покращена продуктивність: React використовує Virtual DOM, завдяки чому швидше створюються веб-додатки. Віртуальний DOM порівнює попередні стани компонентів і оновлює лише ті елементи в реальному DOM, які були змінені,

замість того, щоб оновлювати всі компоненти знову, як це роблять звичайні веб-програми.

Багаторазові компоненти: компоненти є будівельними блоками будь-якої програми React, і одна програма зазвичай складається з кількох компонентів. Ці компоненти мають свою логіку та елементи керування, і їх можна повторно використовувати в усій програмі, що, у свою чергу, значно скорочує час розробки програми.

Односпрямований потік даних: React слідує за односпрямованим потоком даних. Це означає, що під час розробки програми React розробники часто вкладають дочірні компоненти в батьківські компоненти. Оскільки дані передаються в одному напрямку, стає легше виправляти помилки та знати, де виникає проблема в програмі в даний момент.

Легке навчання: React легко освоїти, оскільки він здебільшого поєднує базові концепції HTML і JavaScript з деякими корисними доповненнями. Однак, як і у випадку з іншими інструментами та фреймворками, потрібно витратити деякий час, щоб правильно зрозуміти бібліотеку React. Його можна використовувати для розробки як веб-, так і мобільних додатків: React використовується для розробки веб-додатків, але це ще не все, що він може зробити.

Спеціальні інструменти для легкого налагодження: Facebook випустив розширення Chrome, яке можна використовувати для налагодження програм React. Це робить процес налагодження веб-додатків React швидшим і простішим.

Наведені вище причини виправдовують популярність бібліотеки React і чому її приймає велика кількість організацій і компаній[16].

1.9. Бібліотека Hibernate

Hibernate — це інструмент реляційного відображення об’єктів (ORM) з відкритим вихідним кодом, який надає структуру для відображення об’єктно-орієнтованих моделей домену в реляційні бази даних для веб-додатків.

Об’єктно-реляційне відображення базується на контейнеризації об’єктів і абстракції, яка забезпечує цю здатність. Абстракція дає змогу звертатися до об’єктів, отримувати доступ до них і маніпулювати ними, не враховуючи, як вони пов’язані зі своїми джерелами даних.

Інфраструктура Hibernate ORM керує зіставленням класів Java з таблицями бази даних і типів даних Java з типами даних SQL і забезпечує запити та пошук.

Hibernate — це служба об’єктно-реляційної стійкості та запитів з відкритим кодом для будь-якої програми Java. Hibernate зіставляє класи Java з таблицями бази даних і з типів даних Java на типи даних SQL і звільняє розробника від більшості поширених завдань програмування, пов’язаних із збереженням даних.

Hibernate — це платформа Java, яка спрощує розробку програми Java для взаємодії з базою даних. Це легкий інструмент ORM (Object Relational Mapping) з відкритим кодом. Hibernate реалізує специфікації JPA (Java Persistence API) для збереження даних[17].

Інструмент ORM спрощує створення даних, маніпулювання ними та доступ до них. Це техніка програмування, яка відображає об’єкт на дані, що зберігаються в базі даних.

Інструмент ORM внутрішньо використовує JDBC API для взаємодії з базою даних.

Java Persistence API (JPA) — це специфікація Java, яка надає певну функціональність і стандарт для інструментів ORM. Пакет javax.persistence містить класи та інтерфейси JPA.

Нижче наведено переваги Hibernate framework:

— відкритий і легкий, інфраструктура Hibernate має відкриту програму за ліцензією LGPL і є легкою;

— швидка продуктивність, оскільки кеш використовується внутрішньо в сплячому фреймворку;

— незалежний запит до бази даних, тож не потрібно буде писати специфічні запити до бази даних;

— автоматичне створення таблиці, інфраструктура Hibernate надає можливість автоматично створювати таблиці бази даних;

— спрощує комплексне об’єднання, отримувати дані з кількох таблиць легко в середовищі Hibernate;

— надає статистику запитів і статус бази даних.

Hibernate підтримує кеш-пам’ять запитів і надає статистику про статус запитів і бази даних [17].

2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Створення проекту за допомогою Spring Boot

Щоб швидко створити проект на базі Spring Boot з базовими налаштуваннями, необхідним функціоналом можна використати сервіс для створення Spring проектів. Переходимо на сайт start.spring.io/ та бачимо перед собою сторінку з великим вибором технологій (рис. 2.1).

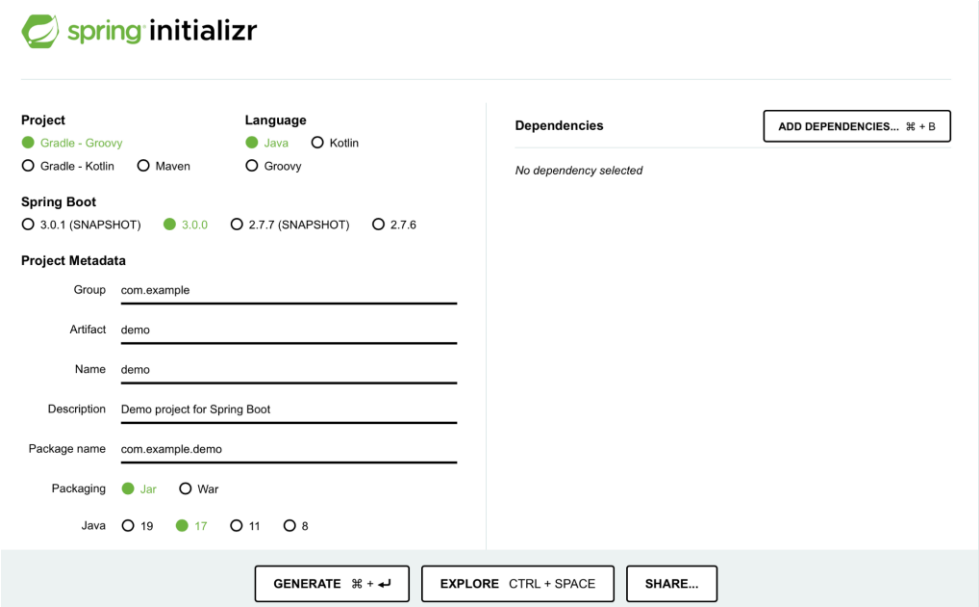

Рисунок 2.1 — Сторінка сайту для створення початкового проекту

Для початку потрібно вирішити який засіб складання проекту буде зручним для нашого проекту. Мій вибір випав на Maven, тому що він краще підходить для проектів розроблених мовою Java. Далі вибір мови програмування - буде, авжеж, Java. Версія Spring Boot - остання стабільна версія - 3.0.0, тому вибираємо її. Далі необхідно заповнити форму з назвою групи, іменем артефакту, та назвою проекту. Наступний крок - додавання набору залежностей, що я буду використовувати для створення системи, натиснувши кнопку “Add Dependencies” з’являється поп-ап з можливими залежностями (рис 2.2). Для нашого застосунку необхідні наступні залежності: JPA, Spring Security, Spring Web, PostgreSQL Driver.

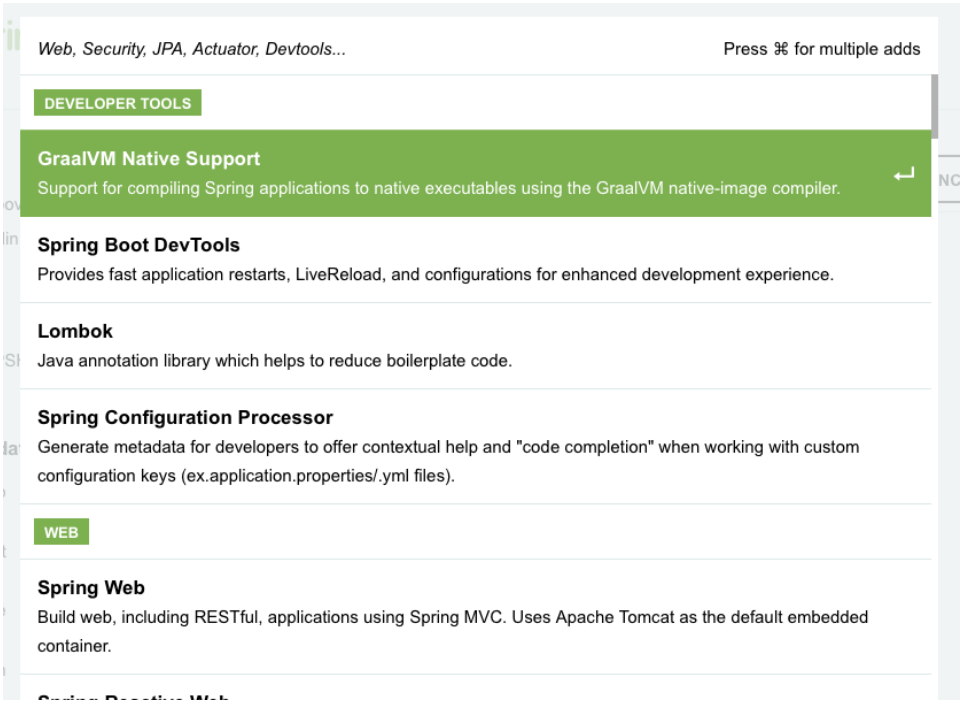

Рисунок 2.2 — Вікно вибору залежностей

Залишилось лиш згенерувати та завантажити початковий проект натиснувши кнопку “Generate”, що зображено на рисунку 2.3.

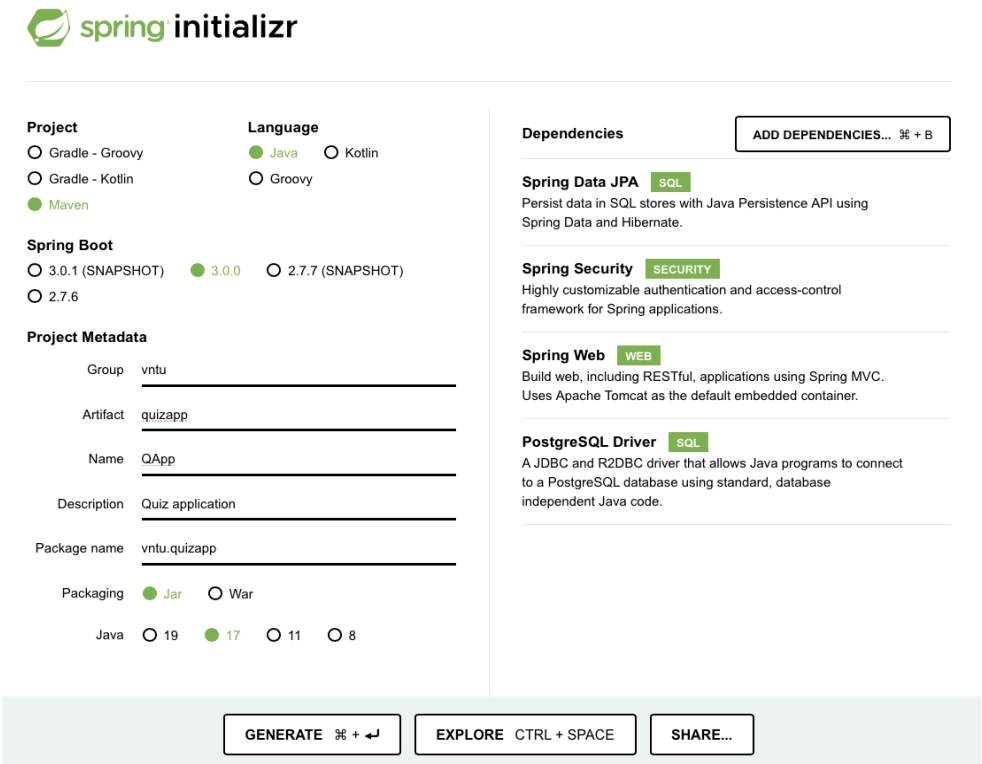

Рисунок 2.3 — Приклад готового до завантаження проекту

2.2. Розробка бази даних

Для розробки бази даних необхідно вирішити які сутності будуть необхідні нам для реалізації поставленої задачі.

Перша сутність – User. Ця сутність міститиме головну інформацію про користувача. Також необхідна сутність ConfirmationToken, яка містить в собі токен для підтвердження створеного аккаунту.

Далі потрібно вирішити як зберігати в базі наші тести, потрібна основна сутність тесту. Кожен тест складається з деякої кількості тест-кейсів. Кожен тест-кейс містить в собі рядки коду та визначення завдання задачі. Для цього створимо 3 сутності quiz, quizCase та quizRow відповідно. Створимо проміжні таблиці для можливості використання зв’язку, багато до багатьох.

Також потрібно зберегти результати проходження тесту, для цього необхідна сутність, що буде містити результат, необхідну інформацію про користувача та обставини проходження тесту.

Визначивши всі необхідні поля сутностей отримуємо таку DB діаграму як зазначено на рисунку 2.4. Гарно спроектована база даних допоможе легко і швидко добавляти функціонал у майбутньому.

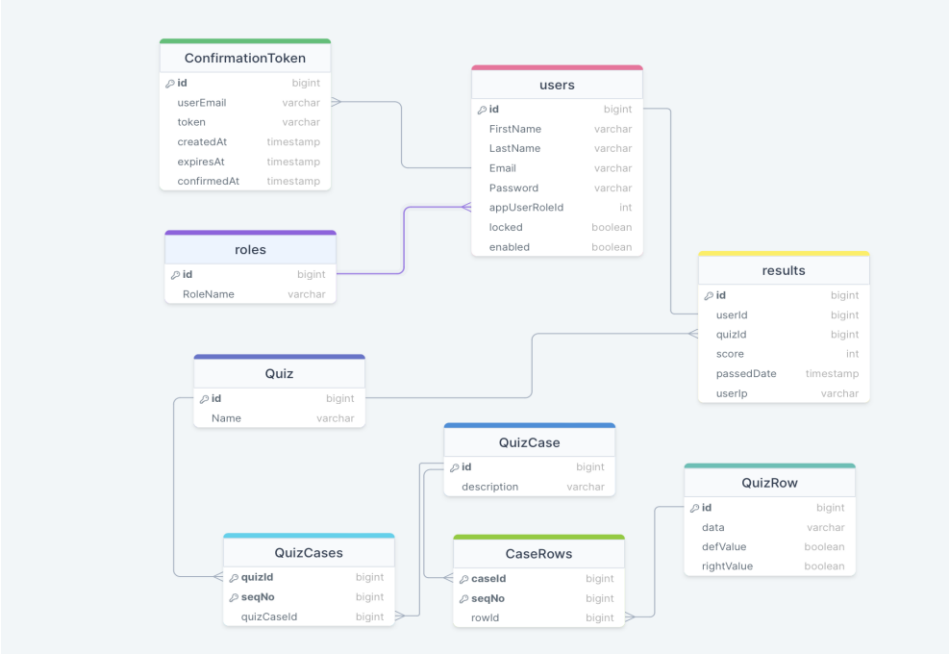

Рисунок 2.4 — Приклад спроектованої DB діаграми на початку проекту

2.3. Встановлення локального серверу бази даних PostgreSQL

Для створення та масштабування необхідно завантажити та запустити базу даних на локальній машині. Для цього потрібно перейти за посиланням postgresql.org/download/ та вибрати інсталятор відповідно до операційної системи. У моєму випадку це Windows. Далі запускаємо інсталяцію, в процесі необхідно придумати та запам'ятати пароль для супер користувача, він буде необхідний потім для підключення до нашого серверу бази даних.

Після завершення інсталяції можемо створити базу даних нашого застосунку. Щоб це зробити є два шляхи:

— використовуючи командну стрічку;

— за допомогою застосунку PGAdmin.

Більш швидко та зручно буде використати застосунок з зручним інтерфейсом PGAdmin. Для початку потрібно підключитись до нашого серверу бази даних, для цього натискаємо кнопку “Add new server”. У вікні, що з’явиться, вводимо назву нашої бази даних та у другій вкладці необхідно ввести адресу серверу, так як він знаходиться на локальній машині, її адреса це localhost. Стандартний порт серверу postgreSQL - це 5432. Також необхідно ввести пароль який ми вводили при інсталяції (рис 2.5).

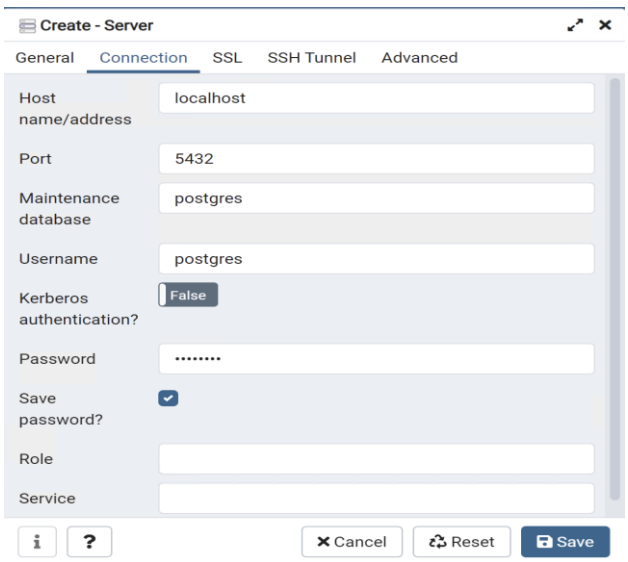

Рисунок 2.5 — Підключення до серверу бази даних

Якщо все введено правильно, ми отримаємо доступ до бази даних. Тут ми можемо побачити, що на сервері уже є головна стандартна база даних postgres, але нам необхідно створити свою, для цього потрібно клікнути правою кнопкою миші на вкладку “Databases” та вибрати пункт “Create Database”. Тут з’явиться вікно створення бази даних, як на рисунку 2.6. Необхідно тільки ввести назву та натиснути кнопку Save. Базу даних створено.

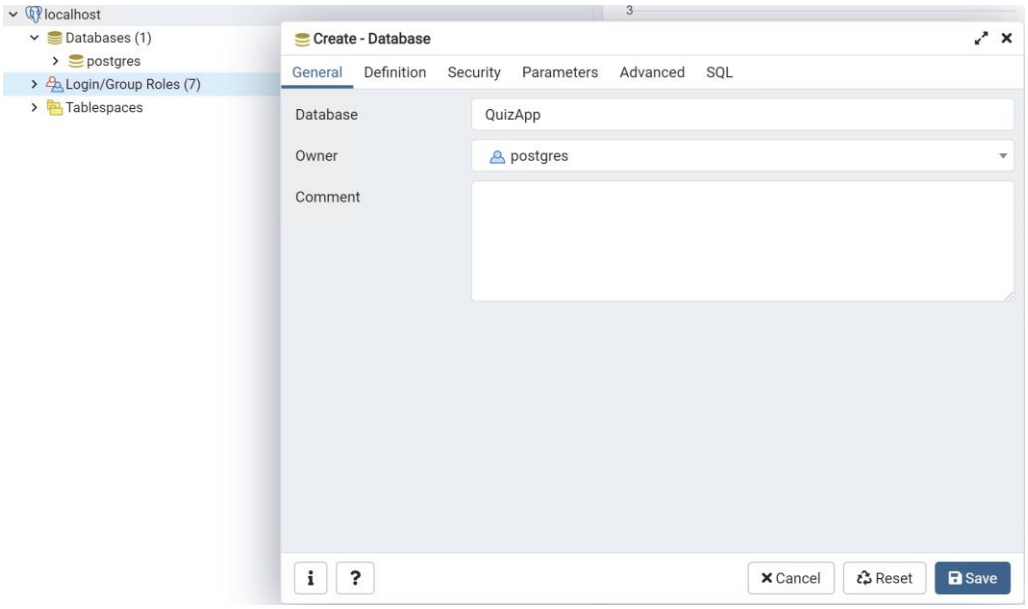

Рисунок 2.6 — Створення бази даних

2.4. Налаштування авторизації за допомогою Spring Security

Для того щоб додати в проект автентифікацію та авторизацію, у проект необхідно додати фреймворк Spring Security. Для цього необхідно додати залежності в pom.xml.

<dependency>

<groupId> org.springframework.security </groupId>

<artifactId> spring-security-core </artifactId>

<version> 5.7.3 </version>

</dependency>

<dependency>

<groupId> org.springframework.security </groupId>

<artifactId> spring-security-web </artifactId>

<version> 5.7.3 </version>

</dependency>

Також потрібно додати залежності для його конфігурування:

<dependency>

<groupId> org.springframework.security </groupId>

<artifactId> spring-security-config </artifactId>

<version> 5.7.3 </version>

</dependency>

<dependency>

<groupId> org.springframework.boot </groupId>

<artifactId> spring-boot-starter-security </artifactId>

<version> 2.5.2 </version>

</dependency>

Далі для створення конфігу безпеки для нашого застосунку необхідно створити клас SequrityConfig та унаслідувати від WebSecurityConfigurerAdapter.

Наступним кроком буде перевизначення методу protected void configure (HttpSecurity http), де буде визначена поведінка під час авторизації.

```
http.cors()
        .and()
        .csrf().disable()
        .httpBasic()
        .and()
        .authorizeRequests()
           .antMatchers("/api/registration/*").permitAll()
        .anyRequest().authenticated()
        .and()
        .formLogin().disable();
```

Щоб не зберігати в базі даних паролі користувачів в первісному виді, що може бути досить небезпечно, потрібно їх закодувати. Для цього засновуємо клас

PasswordConfig, що створюватиме для нашого застосунку BCryptPasswordEncoder.

```
@Bean
public BCryptPasswordEncoder passwordEncoder(){
return new BCryptPasswordEncoder(12);
}
```

Далі необхідно створити провайдер, який буде проводити аутентифікацію користувачів DaoAuthenticationProvider, для його налаштування потрібен шифрувач паролів, який ми створили раніше, та UserDetailsService за допомогою якого він буде отримувати дані користувача з бази даних. При автентифікації він бере введений користувачем пароль, шифрує його та порівнює з збереженим в базі даних зашифрованим паролем.

```
@Bean
   public DaoAuthenticationProvider daoAuthenticationProvider() {
      DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();
      provider.setPasswordEncoder(passwordEncoder);
      provider.setUserDetailsService(applicationUserService);
      return provider;
   }
```

2.5. Налаштування Mail Server та верифікації email

Для ускладнення створення мультиакаунтів, необхідно додати підтвердження email під час реєстрації, не підтверджені аккаунти не зможуть проходити тести. Для цього нам необхідно створити Mail Server. Для цієї задачі додаємо до нашого проекту Spring Mail, що є частиною Spring Boot. Додаємо в залежності в pom.xml.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-mail</artifactId>
```

</dependency>

Бібліотека Spring Mail має невелику кількість класів та інтерфейсів, основні з них це MailSender, який і відповідає за відправку повідомлень, MimeMessage, що містить в собі повідомлення та інформацію, куди потрібно надіслати, та MimeMessageHelper, який є інтерфейсом для зручного заповнення даними сутності MimeMessage. Діаграму класів та інтерфейсів можна подивитись на рисунку 2.7.

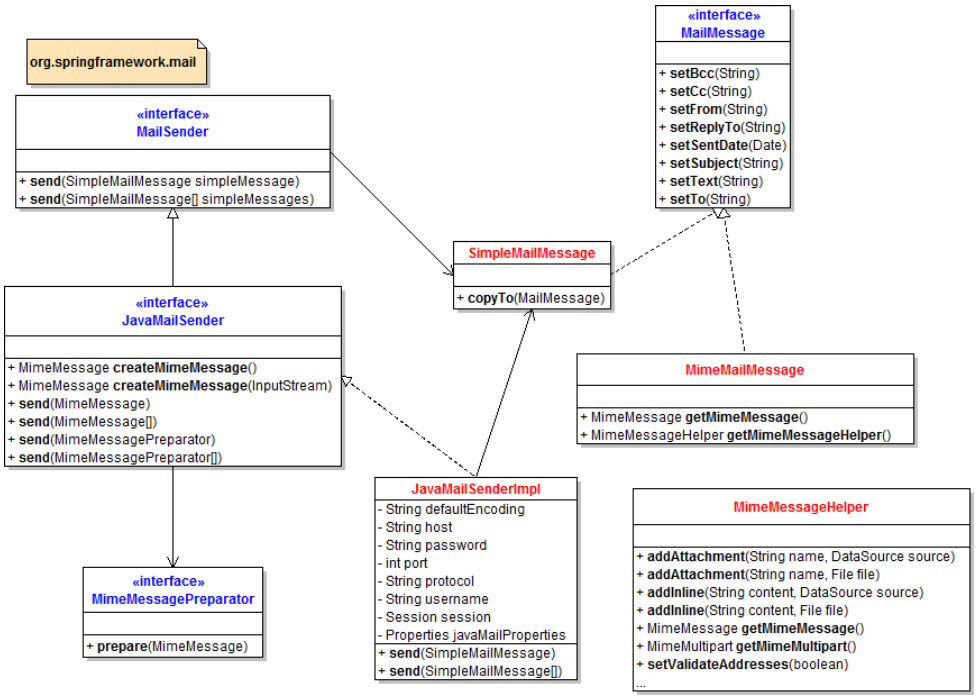

Рисунок 2.7 — Діаграма класів та інтерфейсів Spring Mail

Для відправки повідомлень нам потрібна поштова скринька на одному з поширених Mail-Server'ів. Зараз один з найкращих це Gmail. Створивши обліковий аккаунт на gmail.com, необхідно ще дати дозвіл на управління ним з нашого додатку.

Варто створити Spring-Bean для MailSender. Також в файлі application.properties вказуємо властивості для нашого SMTP-серверу gmail. Налаштування виглядають таким чином:

spring.mail.host=smtp.gmail.com

spring.mail.port=587

```
spring.mail.username=sa1zelage@gmail.com
spring.mail.password=qmskefitfacknmrs
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
```

Тепер можемо створити наш MailService, унаслідувавшись від класу EmailSender та перевизначивши метод send (String to, String email).

```
@Override
   @Async
   public void send(String to, String email) {
      try {
         MimeMessage mimeMessage = mailSender.createMimeMessage();
         MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, "utf-8");
         helper.setText(email, true);
         helper.setTo(to);
         helper.setSubject("Email confirmation");
         helper.setFrom("sa1zelage@gmail.com");
         mailSender.send(mimeMessage);
      } catch (MessagingException e) {
         LOGGER.error("failed to send email", e);
         throw new IllegalStateException("failed to send email");
      }
   }
```

Для верифікації нам треба перевіряти валідність введених email. Це можна реалізувати за допомогою класу InternetAddress, який містить метод validate(). Далі надсилати на поштову скриньки зареєстрованих користувачів токен верифікації, що буде замаскований під посилання підтвердження. Цей токен буде дійсним визначений час, що додасть більшої безпеки нашому застосунку.

Створюємо сутність ConfirmationToken, що міститиме сам токен, дату його створення та email.

2.6. Розробка логіки проходження тестування

Як було описано в розділі, де розроблялась база даних, нам потрібно 3 сутності:

— сутність самого тесту Quiz, що міститиме деяку кількість тест-кейсів, назву тесту та іншу додаткову інформацію;
— сутність тест-кейсу QuizCase, що містить в собі рядки коду та опис задачі;
— сутність рядку коду QuizRow, що містить в собі сам рядок, інформацію про його початковий стан та стан у правильному рішенні.

Тест-кейси можна буде додавати до будь-яких тестів, що додасть більшої гнучкості та ефективності під час створення комплексних тестів. Проте рядки відповідатимуть тільки до одного тест-кейсу.

Під час тестування для користувачеві буде показано один тест-кейс. Після коментування та розкоментування рядків він надсилається одразу на перевірку. Якщо всі рядки у стані правильного рішення, тест-кейс буде рахуватись як пройдений правильно та додавати бали до резутату. І надсилати наступний тест-кейс для вирішення. Тест завершується коли всі тест-кейси пройдено. В цей час зберігається кінцевий результат до бази даних. Результати може переглядати викладач та виставляти відповідні бали.

2.7. Визначення кінцевих точок серверу

Наш проект має серверну і клієнтські частини, для їх взаємодії необхідно пов’язати їх між собою. Це можна реалізувати шляхом створення кінцевих точок серверу. Зв’язок детально показаний на рисунку 2.8.

Кінцеві точки розташовані у контролерах. Для цього їх потрібно створити класи контролерів та додати анотацію @RestController, що визначить клас як контейнер кінцевих точок. Також потрібно додати анотацію @RequestMapping("/api/") з аргументом який містить шлях.

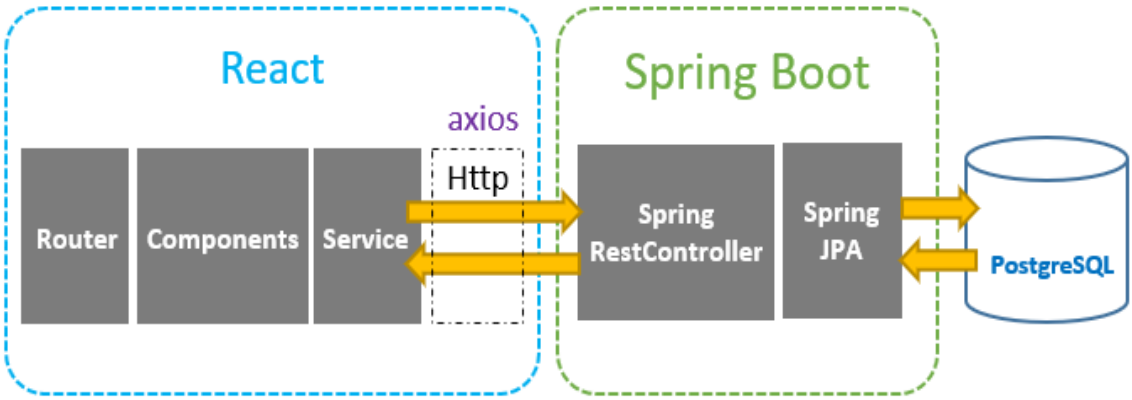

Рисунок 2.8 — Зв’язок між серверною частиною та клієнтом

Кінцеві точки можна поділити на дві частини, одна частина відповідає за автентифікацію та реєстрацію, а інша роботу з тестати та їх результатами.

Кінцеві точки для автентифікації та реєстрації:

— /login (post) - приймає для аутентифікації дані користувача на сервер;

— /register(post) - приймає дані для реєстрації користувача на сервер;

— /confirm (get) - приймає токен верифікації та верифікує його акаунт.

— Кінцеві точки для роботи з тестами:

— /tests(get) - повертає список всіх тестів;

— /tests/count(get) - повертає загальну кількість тестів;

— /test/admin/{id}(get) - повертає тест для редагування користувачу тільки з ролью адміністратора;

— /test/admin/{id}(get) - повертає тест для проходження;

— /test/save(post) - збереження редагованого або створеного тесту;

— /test/delete(delete) - видалення тесту;

— /results(get) - повертає всі доступні результати;

— /results/test/{id}(get) - повертає резельтати відповідно до тесту.

2.8. Створення клієнту за допомогою фреймворку React

Створення клієнту ми будемо за допомогою мови програмування JavaScript та фреймворку React. Ця мова програмування та фреймворк дає велику

зручність для швидкого та зручного створення та налагодження клієнту для нашої системи тестування.

Для початку розробки нам потрібно завантажити Node.js та менеджер пакетів npm. Після встановлення потрібно завантажити початковий темплейт клієнту. Виконуємо наступні команди:

npm install -g create-react-app

create-react-app quizapp

Після їх виконання, у нас в проекті з’явиться файли початкового проекту у дерикторії під назвою webapp, що міститимуть всі файли нашого клієнту. Виконавши команду npm start, буде відкрито браузер на початковій сторінці localhost:3000, де буде зображено початковий інтерфейс застосунку який зображено на рисунку 2.9.

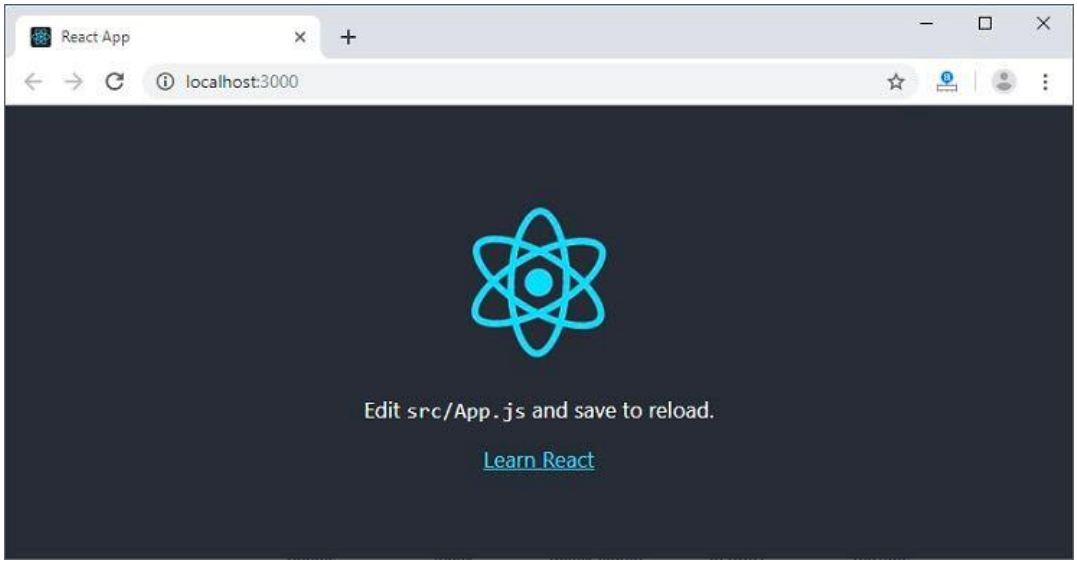

Рисунок 2.9 — Початкова сторінка create-react-app

Потрібно розробити всі сутності, які будуть відповідати сутностям, що були створені на стороні серверу, для їх візуалізації.

Необхідно створити компоненти реєстрації та авторизації. Компонент реєстрації матиме 4 поля: ім’я, прізвище, email та пароль. Кнопка “Register” буде заблокована допоки не буде введено всі дані. Компонент авторизації міститиме

тільки 2 поля: email та пароль. Ці компоненти будуть виглядати як на рисунку 2.10.

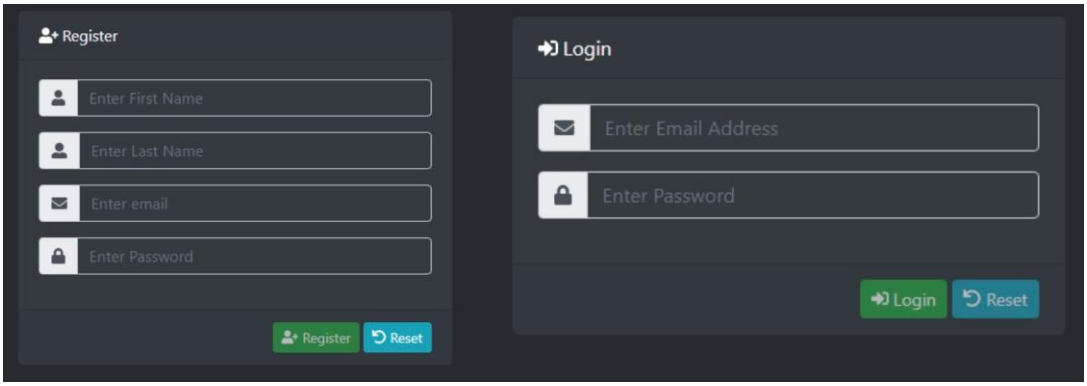

Рисунок 2.10 — Компоненти реєстрації та авторизації

Наступним кроком створюємо сторінки для користувача: перша з переліком доступних для тестів та друга з проходженням тесту.

Також потрібні сторінки для адміністрування, які будуть доступні тільки адміністратору. А саме сторінка з переліком тестів, де буде можливість їх редагувати та сторінка з результатами користувачів, де будуть дані про користувача та його ip.

Створивши всі сторінки та компоненти нашого клієнту, нам необхідно створити маршрутизацію, яка дає можливість пересуватись між сторінками клієнту. React сам по собі не підтримує маршрутизацію, тому необхідно завантажити додатковий модуль під назвою react-router-dom, що містить бібліотеку react-router.

Маршрутизація нашого клієнту описана в файлі App.js(рис). Усі компоненти маршрутизації заключені між тегами <Router>. У роутері визначається набір маршрутів. Коли приходить запит, роутер виконує зіставлення запиту з маршрутами. І якщо один з маршрутів збігається з посиланням запиту, то він вибирається для його обробки. Також визначено об’єкт Switch, що дозволяє обрати відповідний маршрут. Без цього об’єкту, теоретично, можливо вибрати кілька маршрутів для обробки одного запиту.

У кожному маршруті містяться два атрибути:

— шаблон адреси path;

— назва компоненту що обробляє запит.

Також для маршрутизації у нашому додатку визначений хедер що описаний в компоненті NavigationBar.js.

Останнє що потрібно розробити для функціонування клієнту, це налаштування зв’язку з серверною частиною. А саме описати запити до серверу. Для цієї задачі ми будемо використовувати ttp клієнт Axios.

Вся взаємодія з сервером системи, знаходиться в папці services. А саме запити для реєстрації, авторизації та роботи з тестами.

3. ТЕСТУВАННЯ РОБОТИ ДОДАТКУ

Сьогодні все більшої популярності набуває тестова розробка (TDD) — техніка розробки програмного забезпечення, при якій спочатку пишеться тест для певної функціональності, а потім пишеться реалізація цієї функціональності. На практиці, звичайно, не все так ідеально, але в результаті код не просто написаний і протестований, але тести неявно задають вимоги до функціональності, а також показують приклад використання цього функціональність.

Backend Testing — це метод тестування, який перевіряє базу даних або серверну частину веб-програми. Основна мета бекенд-тестування — перевірити прикладний рівень і рівень бази даних. Він знайде помилку в базі даних або на сервері.

Щоб реалізувати бекенд-тестування, інженер-бекенд-тестувальник також повинен мати певні знання про цю конкретну серверну мову або мову бази даних. Він також відомий як тестування бази даних.

Важливість бекенд-тестування: бекенд-тестування є обов’язковим, оскільки будь-яка помилка чи помилка на стороні сервера не призведе до подальшого виконання цього завдання або вихідні дані відрізнятимуться, або іноді це також спричинить такі проблеми, як втрата даних, взаємоблокування, тощо.

3.1. JUnit тестування

JUnit — це платформа модульного тестування для мови програмування Java. JUnit відіграв важливу роль у розвитку розробки, керованої тестуванням, і є одним із сімейства фреймворків модульного тестування, спільно відомих як xUnit, які походять від JUnit.

Модульне тестування — це тестування невеликого фрагмента логіки або коду, щоб переконатися, що результат коду відповідає очікуванням, враховуючи введення певних даних і/або виконання певних умов. Як правило, модульні тести повинні бути незалежними від інших тестів.

Модульні тести неможливі для тестування складних інтерфейсів за допомогою іншої програми або сторонніх/зовнішніх служб. Модульний тест націлений лише на невелику одиницю коду, яка може бути лише методом або класом.

За допомогою модульного тестування в нашому проекті можна покрити тестами бізнес логіку сервісного модуля проекту.

Для цього потрібно створити для кожного сервісу свій клас тесту, та визначити в ньому тести для наших методів. Тести повинні охоплювати крайні ситуації які можливі під час подальшої експлуатації застосунку.

Створюємо клас що буде містити юніт тести нашого QuizModelService, називаємо його відповідно QuizModelServiceTest.

Необхідно створити об’єкт класу QuizModelService та в його конструктор додати заглушки класів QuizModelRepository та QuizResultService. Заглушки потрібні для того щоб не створювати об’єкти цих класів, та відслідковувати методи та дані які їм буде передавати QuizModelService. Для цього ми будемо використовувати фреймворк для тестування Mockito який дає можливість створення заглушок. Заглушки створюємо як глобальні змінні класу та додаємо для них анотації @Mock, а для об’єкту клас якого будемо тестувати @InjectMocks, що автоматично підбирає необхідні заглушки для створення цього об’єкту.

```
@Mock
QuizModelRepository modelRepository;
@Mock
QuizResultService resultService;
@InjectMocks
QuizModelService modelService;
```

Наступним кроком потрібно створити методи що будуть виконуватись перед всіма тестами та після, додавши анотації @Before та @After відповідно. Перед тестами потрібно ініціалізувати всі заглушки, створюємо метод setup та анотуємо його як @Before.

```
@Before
public void setup() {
MockitoAnnotations.openMocks(this);
}
```

Далі пишемо тести, для кожного публічного методу потрібно визначити визначити крайні випадки, коли метод видасть результат, який буде неправильним. Всі тести анотуються за допомогою анотації @Test та назва методу описує те що він перевіряє.

Наприклад створимо тест що буде перевіряти результат, коли всі відповіді правильні.

```
@Test
void getResultAllCorrect() {
   List<QuizAnswers> testAnswers = createTestAnswers();
   Map<ObjectId, QuizCase> quizCases = createQuizCases();
   double result = 100;
   final double actualResult = modelService.getResult(testAnswers, quizCases);
   assertEquals(result, actualResult);
}
```

Також наприклад створимо тест, що буде зберігати тест використовуючи заглушку репозиторію. Ми знаємо що він після збереження повинен отримати id. Для обробки запитів до репозиторію використовуємо метод when(). Як аргумент описуємо виклик методу заглушки quizModelRepository.save(quizModel).

```
   @Test
   void save() {
      final QuizModel quizModel = new QuizModel();
      when(quizModelRepository.save(quizModel)).thenReturn(new
QuizModel(new ObjectId()));
      final QuizModel savedModel = modelService.save(quizModel);
      assertNotEquals(savedModel.getId(), quizModel.getId());
      assertNotNull(savedModel.getId());
```

```
}
```

Такими ж тестами потрібно покрити всю бізнес логіку проекту.

3.2. Тестування контролерів за допомогою Mock MVC

Платформа Spring MVC Test, також відома як MockMvc, надає підтримку для тестування додатків Spring MVC. Він виконує повну обробку запитів Spring MVC, але через імітаційні об’єкти запиту та відповіді замість запущенного сервера.

MockMvc можна використовувати окремо для виконання запитів і перевірки відповідей. Його також можна використовувати через WebTestClient, де MockMvc підключено як сервер для обробки запитів. Перевагою WebTestClient є можливість працювати з об’єктами вищого рівня замість необроблених даних, а також можливість перемикатися на повні наскрізні HTTP-тести на живому сервері та використовувати той самий тестовий API.

Тестування контролерів майже не відрізняється від юніт тестування, проте необхідні додаткові залежності для створення запитів в середині тесту.

```
 <dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<version>2.5.0</version>
<scope>test</scope>
<dependency>
```

Так як у нас в проекті є spring security необхідно щоб наші запити в тестах були авторизовані. В методі setup, що буде виконано перед усіма тестами, необхідно створити об’єкт mockMvc.

```
    @Before
    public void setup() {
        mockMvc = MockMvcBuilders
                .webAppContextSetup(context)
                .apply(springSecurity())
```

```
            .build();
    }
```

Також для всіх ендпойнтів, в яких потрібно щоб запит був авторизований, додаємо анотацію @WithMockUser() та додаємо параметр roles. Наприклад якщо це адміністратор то roles="admin".

Переходимо до створення тестів. Для перевірки ендпоінту, нам необхідно надіслати запит на контролер. Це можна зробити за допомогою об’єкту mockMvc що ми створювали раніше.

```
@Test
    @WithMockUser(roles="ADMIN")
    void getAllQuiz() throws Exception {
        when(service.findAll()).thenReturn(new ArrayList<>());

        mockMvc.perform(
                get("/api/tests/all"))
                .andExpect(status().is2xxSuccessful());

        verify(service, times(1)).findAll();
    }
```

Таким чином можна перевірити всі кінцеві точки проекту.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного продукту система онлайн тестування з програмування на основі редагування програмного коду. Даний продукт допоможе викладачам ефективніше проводити тестування знань студентів з мов програмування.

Особливістю програми полягає в тому, що редагування програмного коду буде виконуватись методом коментування та розкоментування рядків коду.

Аналогом може бути HackerEarth, вартість місячної підписки – 5000 грн. (125$), відповідно річної – 60000 грн.

Усі дані по кожному параметру занесено в таблиці 4.1

Таблиця 4.1 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	3	4
Наявність аналогів на ринку	3	3	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	4	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	3	3	4
Сума	43	42	44
Середньоарифметична сума балів	(43+42+44) / 3 = 43		

За даними таблиці 4.1 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.2.

Таблиця 4.2 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт відрізняється від існуючих тим, що дана технологія є системою онлайн тестування з програмування на основі редагування програмного коду і редагування програмного коду буде виконуватись методом коментування та розкоментування рядків коду. Даний продукт допоможе викладачам ефективніше проводити тестування знань студентів з мов програмування.

4.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

4.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \,, \tag{4.1}$$

де M — місячний посадовий оклад конкретного розробника , грн.;

$T_р$ — число робочих днів в місяці, 21 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 4.4.

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Таблиця 4.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	35000	1666,67	35	58333,333
Програміст	32000	1523,81	35	53333,333
Всього				111666,67

4.2.2 Додаткова заробітна плата розробників, які приймали участь в розробці обладнання.

Додаткова заробітна плата прийнято розраховувати як 11 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 11\ \% / 100\ \% \qquad (4.2)$$

$$З_д = (111666{,}67 \cdot 11\ \% / 100\ \%) = 12283{,}33 \text{ (грн.)}$$

4.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (4.3)$$

$$Н_з = (111666{,}67 + 12283{,}33) \cdot 22\ \% / 100\ \% = 27269{,}00 \text{ (грн.)}$$

4.2.4 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизації обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \qquad (4.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 56000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,67 міс.

$$A_{обл} = \frac{56000}{2} \times \frac{1{,}67}{12} = 3888{,}89\, грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.5.

Таблиця 4.5 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Комп'ютер та комп'ютерна периферія (Apple MacBook Air m1)	56000	2	1,67	3888,889
Офісне обладнання (меблі)	20000	4	1,67	694,444
Приміщення	900000	20	1,67	6250,000
Всього				10833,33

Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів є безкоштовною, то амортизації відповідно не має, $B_{нем.ак.}$ = 0 грн.

4.2.5 Тарифи на електроенергію для непобутових споживачів відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників, будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільчих компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \qquad (4.6)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,5 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$В_е = 0,9 \cdot 0,5 \cdot 8 \cdot 35 \cdot 6,2 = 781,2$ (грн.)

4.2.6 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (4.7)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$I_в$ = 111666,67 * 60% / 100% = 67000 (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \qquad (4.8)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$H_{нзв}$ = 111666,67 * 130 % / 100 % = 145167 (грн.)

5.2.7 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$B_{заг}$ = 111666,67+12283,33+27269,00+10833,33+0+781,20+67000+
+145167 = 375000,20 грн.

5.2.8 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.9)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 375000{,}20 / 0{,}5 = 750000 \text{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до

отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

4.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.10)$$

де $\pm\Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 35000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *2000* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 350 шт., протягом другого року – на 150 шт., протягом третього року на 100 шт. Всього в Україні на 01.01.2022 року налічувалось близько 1500 ЗВО, технікумів, училищ, коледжів і ліцеїв (не шкільного типу). До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1$ = (0*2000 + (35000 + 2000)*350)* 0,8333* 0,45) * (1 - 0,18) = 3766874,849 грн.

$\Delta П_2$ = (0*2000 + (35000 + 2000)*(350+150)* 0,8333* 0,45) * (1 - 0,18) = 5688749,772 грн.

$\Delta П_3$ = (0*2000 + (35000 + 2000)*(350+150+100)* 0,8333* 0,45) * (1 - 0,18) = 6826499,727 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 16282124,35 грн.

4.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

$$ПП = (3766874,849/(1+0,1)^1)+(5688749,772/(1+0,1)^2)+(6826499,727/(1+0,1)^3)$$
$$= 3424431,68 + 4701446,093 + 5128850,283 = 13254728,06 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \qquad (4.12)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

PV = 2 * 750000 = 1500000,80 грн.

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \qquad (4.13)$$

$$E_{абс} = 13254728{,}06 - 1500000{,}80 = 11754727{,}26 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (4.14)$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$E_в = \sqrt[3]{(1 + 11754727{,}26/1500000{,}80} - 1 = 1{,}067$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \qquad (4.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \qquad (4.16)$$

$$T_{ок} = 1 / 1,067 = 0,94 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,94 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 750000 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,94 роки.

ВИСНОВКИ

Під час виконання завдання дипломної роботи було створено програмне забезпечення для тестування з мов програмування.

У першому розділі було розглянуто переваги та доцільність використання мов програмування Java та JavaScript. Також було розглянуто фреймворк Spring з величезною кількістю можливостей для створення онлайн систем. Розглянувши всі можливості цього фреймворку було вибрано низку необхідних бібліотек з його переліку для реалізації поставленої задачі. Також було розглянуто фреймворк React та його можливості для створення клієнтської частини застосунку.

У другому розділі було розглянуто процес створення серверу на базі Spring та налаштування модулів Spring Security, Mail Client та інших. Розроблено структуру бази даних. Показано процес встановлення PostgreSQL та створення бази даних на локальному комп’ютері. Також було показано процес створення клієнтської частини проекту.

У третьому розділі було показано процес написання unit тестів для бізнес логіки. Також протестовані кінцеві точки серверної частини за допомогою Mock MVC, що дає можливість перевірити не тільки правильність логіки, а і безпечність систем авторизації та аутентифікації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Кадомський К.К., Ніколюк П.К. JAVA. ТЕОРІЯ І ПРАКТИКА. – Вінниця, 2018.

2. Java 8 in Action [Електронний ресурс] – Режим доступу: https://www.manning.com/books/java-8-in-action?query=java

3. Java-програмування: комп’ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 122 «Комп’ютерні науки», освітньо-професійної програми «Комп'ютерний моніторинг та геометричне моделювання процесів і систем» / КПІ ім. Ігоря Сікорського ; уклад.: Ю. А. Тарнавський. – Електронні текстові дані. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 95 с.

4. Шилдт Г. Java. Полное руководство. 10-е издание / Герберт Шилдт., 2019 – 1488с.

5. Офіційна документація Spring [Електронний ресурс] – Режим доступу: https://spring.io/docs

6. Spring in Action [Електронний ресурс] – Режим доступу: https://www.manning.com/books/spring-in-action-fifth-edition

7. Spring MVC [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html.

8. Spring Architecture [Електронний ресурс] – Режим доступу: https://www.educba.com/spring-architecture/

9. P. Mularien, M. Knutson, R. Winch, Spring Security (Third Edition), p. 542.

10. Spring Security [Електронний ресурс] – Режим доступу:: https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

11. JSON Web Token (JWT) [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc7519.

12. Кеннеди Б., Муссиано Ч. HTML и XHTML. Полное руководство / Б. Кеннеди, Ч. Муссиано. - М.: «Символ-Плюс», 2012. - 752 с. - ISBN: 5-93286-104-5

13. HTML & CSS – W3C [Електронний ресурс] – Режим доступу: http://www.w3.org/standards/webdesign/htmlcss

14. About JavaScript [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

15. Використання React JS. [Електронний ресурс] – Режим доступу: https://uk.education-wiki.com/6785493-usesof-react-js

16. Container Components in React. 2021 [Електронний ресурс] – Режим доступу: https://www.section.io/engineering-education/container-components-in-react/.

17. “Hibernate ORM 5.4.31.Final User Guide” [Електронний ресурс] – Режим доступу: https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Gui de.html

ДОДАТОК А

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н..
_____Азаров О.Д.

"____" _____2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи
" Система онлайн тестування з програмування на основі редагування програмного коду"
08-23. МКР.010.00.000.ТЗ

Науковий керівник: проф.
_____ Черняк О.І.

Студент групи 1КІ-21м
_____ Нагорний С.М.

ВНТУ 2022

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

Підставою для виконання МКР є підвищення ефективності перевірки знань з програмування, що в свою чергу підвищить рівень знань студентів. Актуальність роботи полягає в розробці системи онлайн тестування з програмування на основі редагування програмного коду. Що дозволить перевірити навички з програмування на реальних прикладах програмного коду.

2 Мета МКР і призначення розробки

2.1 Мета проекту — огляд сучасного стану систем перевірки знань з програмування. Аналіз популярних сайтів, що дають змогу перевірити знання з програмування на основі редагування програмного коду. Аналіз способів покращення та їх практичне використання.

2.2 Призначення розробки — створення системи онлайн тестування на основі редагування програмного коду;

3 Вихідні дані для виконання МКР

3.1 Розкриття актуальності роботи над перевіркою знань з програмування;

3.2 Аналіз усіх систем перевірки знань з програмування та огляд методів покращення їх можливостей;

3.3 Розробити систему з використанням розглянутих методів;

3.4 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору;

3.5 Протестувати додаток.

4 Вимоги до виконання МКР

Головна вимога — використати, методи покращення доступності сайтів

для досягнення максимального результату за параметром доступності

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Огляд технологій для розробки серверної частини онлайн системи.	24.09.2022	27.09.2022	Розділ 1
2	Огляд методів та технологій для створення клієнту онлайн системи.	28.09.2022	29.09.2022	Розділ 1
3	Розробка бази даних та серверної частини онлайн системи	15.10.2022	18.10.2022	Розділ 2
4	Розробка клієнту для доступу до онлайн системи	23.10.2022	05.11.2022	Розділ 2
5	Тестування додатку	06.11.2022	08.11.2022	Розділ 3
6	Підготовка економічної частини	10.11.2022	15.11.2022	Розділ 4
7	Оформлення пояснювальної записки, графічного матеріалу і презентації	29.11.2022	15.12.2022	ПЗ, графіч. матеріал і презентація
8	Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат	17.12.2022	19.12.2022	Оформлені документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

Додаток Б

Лістинг клієнтської частини додатку

Файл App.js:

```
export default function App() {

  window.onbeforeunload = (event) => {
    const e = event || window.event;
    e.preventDefault();
    if (e) {
      e.returnValue = '';
    }
    return '';
  };

 const heading = "Welcome to Testing Service";
 const quote = "Testing leads to failure, and failure leads to understanding.";
 const footer = "Burt Rutan";

 return (
  <Router>
    <NavigationBar/>
    <Container>
      <Row>
        <Col lg={12} className={"margin-top"}>
          <Switch>
            <Route path="/" exact component={() => <Welcome
heading={heading} quote={quote} footer={footer}/>}/>
            <Route path="/confirm" exact component={() => <Welcome
heading="Successful registration" quote="To finish
            registration, confirm your email." footer="Administration"/>}/>
            <Route path="/add" exact component={Quiz}/>
            <Route path="/edit/:id" exact component={Quiz}/>
            <Route path="/tests" exact component={QuizList}/>
            <Route path="/results" exact component={ResultList}/>
```

```
                    <Route path="/register" exact component={Register}/>
                    <Route path="/login" exact component={Login}/>
                    <Route path="/logout" exact component={Login}/>
                </Switch>
            </Col>
        </Row>
    </Container>
  </Router>
 );
}
```

Файл NavigationBar:

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import {Navbar, Nav} from 'react-bootstrap';
import {Link} from 'react-router-dom';
import {FontAwesomeIcon} from '@fortawesome/react-fontawesome';
import {faUserPlus, faSignInAlt, faSignOutAlt} from '@fortawesome/free-solid-svg-
icons';
import {logoutUser} from '../services/index';

class NavigationBar extends Component {
  logout = () => {
    this.props.logoutUser();
  };

  render() {
    const guestLinks = (
      <>
        <div className="mr-auto"></div>
        <Nav className="navbar-right">
          <Link to={"register"} className="nav-link"><FontAwesomeIcon
icon={faUserPlus} /> Register</Link>
          <Link to={"login"} className="nav-link"><FontAwesomeIcon
icon={faSignInAlt} /> Login</Link>
```

```
            </Nav>
        </>
    );

    const userLinks = (
        <>
            <Nav className="mr-auto">
                <Link to={"tests"} className="nav-link">Testing</Link>
                <Link to={"results"} className="nav-link">Results</Link>
            </Nav>
            <Nav className="navbar-right">
                <Link to={"logout"} className="nav-link"
onClick={this.logout}><FontAwesomeIcon icon={faSignOutAlt} /> Logout</Link>
            </Nav>
        </>
    );


    const adminLinks = (
        <>
            <Nav className="mr-auto">
                <Link to={"add"} className="nav-link">Add Test</Link>
                <Link to={"tests"} className="nav-link">Tests</Link>
            </Nav>
            <Nav className="navbar-right">
                <Link to={"logout"} className="nav-link"
onClick={this.logout}><FontAwesomeIcon icon={faSignOutAlt} /> Logout</Link>
            </Nav>
        </>
    );

    let linksToDisplay = guestLinks;
    const { isLoggedIn, isAdmin } = this.props.auth;
    if(isLoggedIn) {
        linksToDisplay = isAdmin? adminLinks: userLinks;
    }
```

```
        return (
            <Navbar bg="dark" variant="dark">
                <Link to={""} className="navbar-brand">
                </Link>
                {linksToDisplay}
            </Navbar>
        );
    };
};

const mapStateToProps = state => {
    return {
        auth: state.auth
    };
};

const mapDispatchToProps = dispatch => {
    return {
        logoutUser: () => dispatch(logoutUser())
    };
};

export default connect(mapStateToProps, mapDispatchToProps)(NavigationBar);
```

Файл Quiz:

```
import React, {Component} from 'react';

import {connect} from 'react-redux';
import {saveQuiz, fetchQuiz, updateQuiz, fetchLanguages, fetchGenres} from
'../../services/index';

import {Card, Form, Button, Col} from 'react-bootstrap';
```

```
import {FontAwesomeIcon} from '@fortawesome/react-fontawesome';
import {faSave, faPlusSquare, faUndo, faList, faEdit} from '@fortawesome/free-solid-svg-icons';
import MyToast from '../MyToast';

class Quiz extends Component {

    constructor(props) {
        super(props);
        this.state = this.initialState;
        // this.quizChange = this.quizChange.bind(this);
        this.submitQuiz = this.submitQuiz.bind(this);
    }

    initialState = {
        id: '', name: '', questions: []
    };

    componentDidMount() {
        const quizId = +this.props.match.params.id;
        if (quizId) {
            this.findQuizById(quizId);
        }
    }

    findQuizById = (quizId) => {
        this.props.fetchQuiz(quizId);
        setTimeout(() => {
            let quiz = this.props.quizObject.quiz;
            if (quiz != null) {
                this.setState({
                    id: quiz.id,
                    title: quiz.title,
                    creator: quiz.creator
                });
```

```
            }
        }, 1000);
    };

    resetQuiz = () => {
        this.setState(() => this.initialState);
    };

    prepareData = () => {
        return {
            name: this.state.name,
            questionList: this.state.questions.map(question => {
                const rightAnswerQuestion = question.answers.find(answer => answer.id ===
question.correctAnswerId);
                return {
                    title: question.title,
                    answers: question.answers.map(answer => answer.title),
                    rightAnswer: rightAnswerQuestion.title
                }
            })
        }
    }

    submitQuiz = event => {
        event.preventDefault();

        this.props.saveQuiz(this.prepareData());



        //
        // const quiz = {
        //     title: this.state.title
        // };
        //
        // this.props.saveQuiz(quiz);
```

```
    // setTimeout(() => {
    //     if (this.props.quizObject.quiz != null) {
    //         this.setState({"show": true, "method": "post"});
    //         setTimeout(() => this.setState({"show": false}), 3000);
    //     } else {
    //         this.setState({"show": false});
    //     }
    // }, 2000);
    // this.setState(this.initialState);
};

updateQuiz = event => {
    event.preventDefault();

    const quiz = {
        id: this.state.id,
        title: this.state.title,
    };
    this.props.updateQuiz(quiz);
    setTimeout(() => {
        if (this.props.quizObject.quiz != null) {
            this.setState({"show": true, "method": "put"});
            setTimeout(() => this.setState({"show": false}), 3000);
        } else {
            this.setState({"show": false});
        }
    }, 2000);
    this.setState(this.initialState);
};


quizList = () => {
    return this.props.history.push("/list");
};
```

```
  render() {
    const {questions} = this.state;

    console.log(questions);

    return (
      <div>
        <div style={{"display": this.state.show ? "block" : "none"}}>
          <MyToast show={this.state.show}
                   message={this.state.method === "put" ? "Quiz Updated
Successfully." : "Quiz Saved Successfully."}
                   type={"success"}/>
        </div>
        <Card className={"border border-dark bg-dark text-white"}>
          <Card.Header>
            <FontAwesomeIcon
              icon={this.state.id ? faEdit : faPlusSquare}/> {this.state.id ? "Update
Quiz" : "Add New Quiz"}
          </Card.Header>
          <Form onReset={this.resetQuiz} onSubmit={this.state.id ? this.updateQuiz
: this.submitQuiz}
                id="quizFormId">
            <Card.Body>
              <Form.Row>
                <Form.Group as={Col} controlId="formGridTitle">
                  <Form.Label>Quizz Title</Form.Label>
                  <Form.Control required autoComplete="off"
                                type="test" name="title"
                                value={this.state.name}
                                onChange={(event => this.setState({name:
event.target.value}))}
                                className={"bg-dark text-white"}
                                placeholder="Enter Quiz Title"/>
                </Form.Group>
              </Form.Row>
              {questions.map(this.renderQuestion)}
```

```
                    <Fo rm.Row>
                      <Button size="sm" variant="success" type="button"
onClick={this.handleAddQuestion}>
                        <FontAwesomeIcon icon={faPlusSquare}/> Add Question
                      </Button>{' '}
                    </Form.Row>
                    <Form.Row>
                    </Form.Row>
                  </Card.Body>
                  <Card.Footer style={{"textAlign": "right"}}>
                    <Button size="sm" variant="success" type="submit">
                      <FontAwesomeIcon icon={faSave}/> {this.state.id ? "Update" :
"Save"}
                    </Button>{' '}
                    <Button size="sm" variant="info" type="reset">
                      <FontAwesomeIcon icon={faUndo}/> Reset
                    </Button>{' '}
                    <Button size="sm" variant="info" type="button"
onClick={this.quizList.bind()}>
                      <FontAwesomeIcon icon={faList}/> Quiz List
                    </Button>
                  </Card.Footer>
                </Form>
              </Card>
            </div>
        );
    }

    createAnswer = () => {
        return {
            id: new Date().getTime() + Math.random(),
            title: '',
        }
    }

    handleAddQuestion = () => {
```

```
    const {questions} = this.state;
    const answers = Array(4).fill(0).map(() => this.createAnswer())
    questions.push({
        id: new Date().getTime(),
        title: '',
        correctAnswerId: answers[0].id,
        answers,
    });
    this.setState({questions})
}

updateQuestionTitle = (question, title) => {
    question.title = title;
    this.setState({
        questions: [...this.state.questions]
    })
}

updateQuestionAnswerTitle = (answer, title) => {
    answer.title = title;
    this.setState({
        questions: [...this.state.questions]
    })
}

deleteQuestion = (question) => {
    const questions = this.state.questions.filter(q => q !== question);
    this.setState({questions: [...questions]})
}

updateCorrectAnswer = (question, answer) => {
    question.correctAnswerId = answer.id;
    this.setState({
        questions: [...this.state.questions]
    })
```

```
  }

  renderQuestion = (question) => {
    const {title, answers} = question;
    return (
      <Form.Row key={question.id}>
        <Form.Group as={Col} controlId="formGridTitle">
          <Form.Control required autoComplete="off"
                        type="test" name="title"
                        value={title}
                        onChange={(event => this.updateQuestionTitle(question,
event.target.value))}
                        className={"bg-dark text-white"}
                        placeholder="Enter Quiz Question Title"
          />

          <div>
            {answers.map((answer) => {
              return (
                <div style={{paddingLeft: 30}} key={answer.id}>
                  <div>
                    <span>Is correct answer?</span>
                    <input type="checkbox"
                           checked={answer.id === question.correctAnswerId}
                           onChange={() => this.updateCorrectAnswer(question,
answer)}
                    />
                  </div>
                   <Form.Control required autoComplete="off"
                                 type="test" name="title"
                                 value={answer.title}
                                 onChange={(event =>
this.updateQuestionAnswerTitle(answer, event.target.value))}
                                 className={"bg-dark text-white"}
                                 placeholder="Enter Quiz Question Title"
                   />
```

```
                        </div>
                    )
                })}
            </div>



                <Button type="reset" onClick={() =>
this.deleteQuestion(question)}>Delete</Button>
            </Form.Group>
        </Form.Row>
    )
  }

};


const mapStateToProps = state => {
  return {
    quizObject: state.quiz
  };
};

const mapDispatchToProps = dispatch => {
  return {
    saveQuiz: (quiz) => dispatch(saveQuiz(quiz)),
    fetchQuiz: (quizId) => dispatch(fetchQuiz(quizId)),
    updateQuiz: (quiz) => dispatch(updateQuiz(quiz))
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(Quiz);
```

Додаток В

Лістинг серверної частини додатку

Файл Application.java:

```
package com.studenttesting;

import com.studenttesting.model.Question;
import com.studenttesting.model.QuizModel;
import com.studenttesting.model.QuizResult;
import com.studenttesting.repository.ResultRepository;
import com.studenttesting.service.QuizModelService;
import com.studenttesting.service.QuizResultService;
import com.studenttesting.users.ApplicationUser;
import lombok.AllArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

@AllArgsConstructor
@SpringBootApplication
public class Application {

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

Файл QuizController.java:

```
@AllArgsConstructor
@RestController
@RequestMapping("/api/tests")
@CrossOrigin(origins="http://localhost:3000")
public class QuizController {

   private final static Logger LOGGER =
LoggerFactory.getLogger(QuizController.class);
   private final QuizModelService quizModelService;



   @PreAuthorize("hasAuthority('ADMIN')")
   @PostMapping("test")
   @ResponseStatus(HttpStatus.CREATED)
   public QuizModel postQuizModel(@RequestBody QuizModel testModel,
Authentication authentication) {
      ApplicationUser user = (ApplicationUser) authentication.getPrincipal();
      return quizModelService.save(testModel, user);
   }




   @GetMapping
   public ResponseEntity<Page<QuizModel>> getQuizModels(int pageNumber, int
pageSize, String sortBy, String sortDir) {
      return new ResponseEntity<>(quizModelService.findAll(
            PageRequest.of(
                  pageNumber, pageSize,
                  sortDir.equalsIgnoreCase("asc") ? Sort.by(sortBy).ascending() :
Sort.by(sortBy).descending()
            )
      ), HttpStatus.OK);
   }

   @GetMapping("all")
```

```
    public List<QuizModel> getAllQuizz() {
        return quizModelService.findAll();
    }

    @JsonView(Views.AdminUI.class)
    @GetMapping("test/admin/{id}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<QuizModel> getAdminQuizModel(@PathVariable String id)
{
        Optional<QuizModel> testModel = quizModelService.findOne(id);
        if (testModel.isEmpty())
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        return ResponseEntity.ok(testModel.get());
    }

    @PreAuthorize("hasAuthority('ADMIN') or hasAuthority('USER')")
    @JsonView(Views.UserUI.class)
    @GetMapping("test/user/{id}")
    public ResponseEntity<QuizModel> getUserQuizModel(@PathVariable String id) {
        Optional<QuizModel> testModel = quizModelService.findOne(id);
        if (testModel.isEmpty())
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        return ResponseEntity.ok(testModel.get());
    }

    @GetMapping("tests/count")
    public Long getCount() {
        return quizModelService.count();
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @DeleteMapping("test/{id}")
    public String deleteQuizModel(@PathVariable String id) {
        return quizModelService.delete(id);
    }
```

```
    @PreAuthorize("hasAuthority('ADMIN')")
    @DeleteMapping("tests")
    public void deleteQuizModels() {
        quizModelService.deleteAll();
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @PutMapping("test")
    public QuizModel putQuizModel(@RequestBody QuizModel testModel) {
        return quizModelService.update(testModel);
    }

    @ExceptionHandler(RuntimeException.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public final Exception handleAllExceptions(RuntimeException e) {
        LOGGER.error("Internal server error.", e);
        return e;
    }
}
```

Файл RegistrationService.java:

```
@Service
@AllArgsConstructor
public class RegistrationService {

    private final ApplicationUserService applicationUserService;
    private final EmailValidator emailValidator;
    private final ConfirmationTokenService confirmationTokenService;
    private final EmailSender emailSender;

    public String register(RegistrationRequest request) {
        boolean isValidEmail = emailValidator.test(request.getEmail());
```

```
    if (!isValidEmail) {
        throw new IllegalStateException("Email not valid");
    }

    String token = applicationUserService.signUpUser(
            new ApplicationUser(
                    request.getFirstName(),
                    request.getLastName(),
                    request.getEmail(),
                    request.getPassword(),
                    ApplicationUserRole.USER
            )
    );

    sendEmail(request.getEmail(), request.getFirstName(), token);

    return token;
}

public void sendEmail(String email, String firstName, String token) {
    String link = "http://localhost:8080/api/registration/confirm?token=" + token;
    emailSender.send(
            email,
            buildEmail(firstName, link));
}

public String confirmToken(String token) {
    ConfirmationToken confirmationToken = confirmationTokenService
            .getToken(token)
            .orElseThrow(() ->
                    new IllegalStateException("token not found"));

    if (confirmationToken.getConfirmedAt() != null) {
        throw new IllegalStateException("email already confirmed");
    }
```

```
        LocalDateTime expiredAt = confirmationToken.getExpiresAt();

        if (expiredAt.isBefore(LocalDateTime.now())) {
            throw new IllegalStateException("token expired");
        }

        confirmationTokenService.setConfirmedAt(token);
        applicationUserService.enableAppUser(
                confirmationToken.getUserEmail());
        return "confirmed";
    }
}
```

ДОДАТОК Г
ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Система онлайн тестування з програмування на основі редагування програмного коду

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 89.7% Схожість 10.3%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Нагорний С.М.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Черняк О.І.
(підпис) (прізвище, ініціали)