МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

СОРОКА ОЛЕКСАНДР СЕРГІЙОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
доктор технічних наук, доцент
_____Т. В. Нескородєва
«__»___20__р.

РОЗРОБКА ВЕБ-ДОДАТКУ МОНІТОРІНГУ РЕЗУЛЬТАТІВ НАВЧАННЯ ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:

Бабаков Р. М., доцент кафедри
інформаційних технологій,

д.т.н., доцент

Оцінка:___/___/_____

(бали за шкалою СКТС/за національною шкалою)

Голова ЕК:_____

(підпис)

Вінниця – 2022

АНОТАЦІЯ

Сорока О. С. Розробка веб-додатку моніторінгу результатів навчання здобувачів вищої освіти. Спеціальність 122 «Комп'ютерні науки», спеціалізація «Інформаційні технології»

Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній (бакалаврській) роботі досліджені сучасні методи розробки веб-додатку з використанням побудови архітектури з незалежними модулями для системи моніторінгу успішності здобувачів вищої освіти. Пошук та вивчення веб-додатків для моніторінгу результатів навчання студентів, розгляд аналогів. Створення веб-сайту для зручного та безперебійного моніторінгу та керування успішністю навчання здобувачів вищої освіти.

Ключові слова: моніторінг, успішність, оцінки, адміністрування, веб-сайт, розробка, мікросервіси, REST, безпека, горизонтальне масштабування.

52 с., 29 рис., 20 джерел.

ABSTRACT

Soroka O. S. Development of a web application for monitoring the learning outcomes of higher education students. Specialty 122 “Computer Science”, specialization “Information Technology”. Vasyl Stus Donetsk National University, Vinnytsia, 2022.

In the qualification (bachelor's) work the modern methods of web application development with the use of architecture construction with independent modules for the system of monitoring the effectiveness of higher education students are investigated. Search and study of web applications for monitoring student learning outcomes, consideration of analogues. Creating a website for convenient and uninterrupted monitoring and management of the success of higher education.

Keywords: monitoring, performance, grades, administration, website, development, microservices, REST, security, horizontal scale.

52 p., 29 draws, 20 sources.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД – база даних

ПЗ – програмне забезпечення

API – Application Programming Interface (Програмний інтерфейс додатку)

JSON – JavaScript Object Notation (Формат об’єктів JavaScript)

XML - Extensible Markup Language (Формат об’єктів у вигляді тегів)

HTTP – Hypertext Transfer Protocol (Протокол передачі даних)

UI – User Interface (Інтерфейс користувача)

REST - Representational State Transfer (Передача представницького стану). Архітектурний стиль побудови API

JDK – Java Development Kit. Мінімальний набір для розробки і запуску програм на мові Java

SQL – Structured Query Language. Мова запитів у базу даних.

MVC – Model View Controller. Паттерн проектування архітектури.

DAO – Data Access Object. Об’єкт, що реалізує роботу з базою даних.

DTO – Data Transfer Object. Об’єкт, який вміщує в собі певну інформацію, зручну для представлення.

ORM – Object Relational Mapping. Система, яка дозволяє форматувати інформацію в об’єкт.

ВСТУП

Для будь якого студента важливий моніторінг за його успішністю в його навчальному закладі і бути в будь який час проінформованим за його поточну кількість оцінок. Але не в кожному вищому навчальному закладі є централізована система, яка дозволяє це перевіряти, і, деякі викладачі обирають зручний для себе метод: онлайн ресурси з загальним доступом для перегляду документів, Moodle, або розсилають вручну через старост груп. Такі підходи можуть бути не дуже зручними, оскільки студент має згадувати місце, де викладач виставляє оцінки та шукати їх. Для цього доцільно використовувати власну систему оцінювання, яка чітко орієнтована на вищий навчальний заклад і його викладачів зі студентами.

Для розробки такої системи потрібно орієнтуватись на наступні критерії: безперебійність, безпека, продуктивність та наявність зворотного зв’язку. Тільки коли вони будуть працювати злагоджено, тоді й буде досягнутий найкращий досвід користування додатком.

Тема бакалаврської роботи – «Розробка веб-додатку моніторінгу результатів навчання здобувачів вищої освіти». Тема є актуальною для більшості вищий навчальних закладів, оскільки:

1. Центр виставлення оцінок має бути централізованим.
2. Здобувачі вищої освіти повинні бути в курсі своєї успішності в будь коли.
3. Студенти можуть швидко реагувати на якісь неточності в оцінках.

Мета бакалаврської роботи – розробка веб-сайту для моніторінгу та керування оцінками студентів вищого навчального закладу. За допомогою цього

сайту, учні можуть в будь який час переглядати свою успішність, а викладачі – поновлювати цю інформацію.

Розділ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

1.1. Актуальність теми

Більшість вищий навчальних закладів не мають власних систем моніторингу та управління і навіть власних реалізацій існуючих. В часи диджіталізації, коли будь яку інформацію можна знайти в мережі Інтернет, така система є необхідною, оскільки, бувають випадки, коли викладачі зберігають інформацію про успішність на різних ресурсах. Деякі з них можуть працювати в реальному часі, а деякі – ні, і викладач повинен оприлюднювати оцінки раз в певний час. Такі нюанси призводять до того, що студент може втратити час на пошук актуальної інформації, або навіть не знайти.

Також актуальності такій системі додають зовнішні фактори, які впливають на роботу університету і студенти не можуть зустрітись очно з викладачем і дізнатись оцінки. Тим паче, бувають випадки, коли немає можливості відвідати навчальний заклад і сповістити про успішність. Тому, онлайн додаток є найбільш доречним у сучасному світі для виконання вище перерахованих задач.

1.2. Розгляд аналогів

На сьогодення, існує велика кількість додатків, які надають можливість створити систему виставлення та перегляду успішності студентів. Розглянемо найпопулярніші з них.

1.2.1 ThinkWave

Частково безкоштовний веб-сайт, який дозволяє викладачам керувати успішністю студентів та інформувати про неї їх батькам. Також є можливість створювати завдання для здобувачів освіти. Оскільки цей додаток є тільки частково безкоштовним, то деякий корисний функціонал є заблокованим.

В безкоштовній версії доступні такий функціонал:

1. Створення завдань для студентів.
2. Система спілкування між користувачами.
3. Автоматичне сповіщення про дії на поштову скриньку.

До переваг цього додатку можна віднести те, що має зручний інтерфейс, гнучкість в налаштуванні та простоту налаштування. Серед недоліків є обмеженість функціоналу додатку у безкоштовній версії, розташування інформації у віддалених баз даних і неможливість додавання функціоналу, який потрібен саме користувачу.

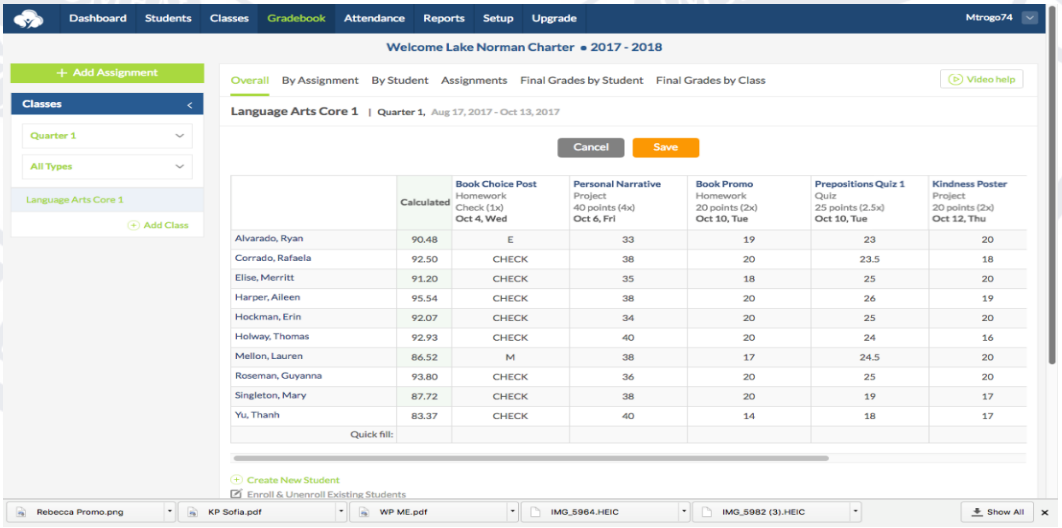

Рисунок 1.1 – Інтерфейс додатку ThinkWave

1.2.2 Moodle

Moodle – це open-source платформа навчання, яка надає інструменти для автоматизації навчальних процесів, серед який є і система оцінювання. Багато вищих навчальних закладів України використовують цю систему, оскільки в ній надається можливість власноруч налаштувати зовнішній вигляд додатку і системи загалом.

Такий широкий спектр інструментів надається не безкоштовно – безоплатно можна користуватись системою тільки певний період часу. Від ціни буде залежати максимальна кількість користувачів сайту та максимальний обсяг пам’яті, який можуть займати певні документи.

Переваги Moodle:

1. Великий набір інструментів для гнучкого налаштування системи.
2. Простота у створенні власної реалізації системи.
3. Висока продуктивність системи.

До недоліків можна віднести відносно велику ціну за утримання сайту, а також неможливість системи оброблювати велику кількість інформації, що призводить до його уповільнення.

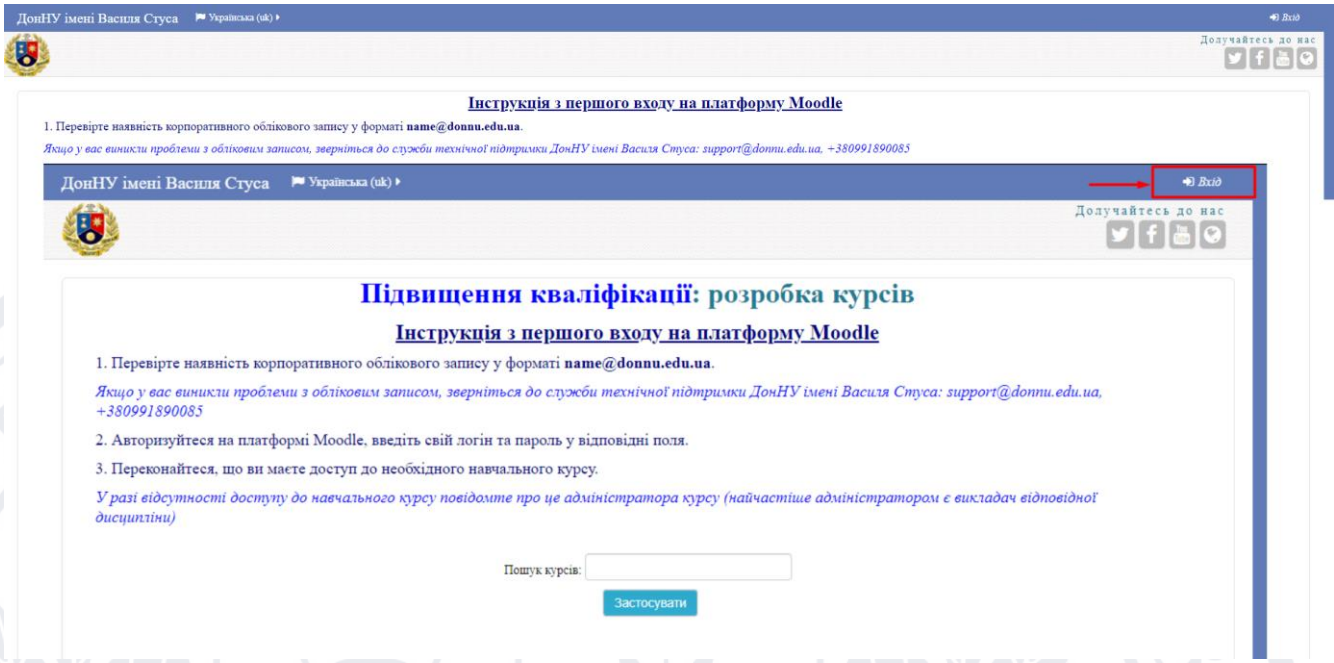

Рисунок 1.2 – Інтерфейс Moodle у реалізації ДонНУ ім. Василя Стуса

1.2.3 Google Classroom

Частково безкоштовна система, яка дозволяє створювати завдання, оцінювати їх, а також зберігати документи. Так як власник цієї системи є «гегемоном» у світі IT-розробки, то це дає сайту високої кросплатформеності та продуктивності у роботі. Система має простий для вивчення інтерфейс та можливість налаштування доступу користувачів для певного домену їх пошти.

Серед переваг можна знайти швидкість роботи, можливість зберігати необмежену за планом підписки кількість документів, а також можливість комунікації викладача зі студентом. Недоліки в ній також існують – неможливість налаштувати доступ для різних доменів, неможливість інтегрувати систему з іншими додатками (Google Calendar) і відносну дорожнечу планів підписок.

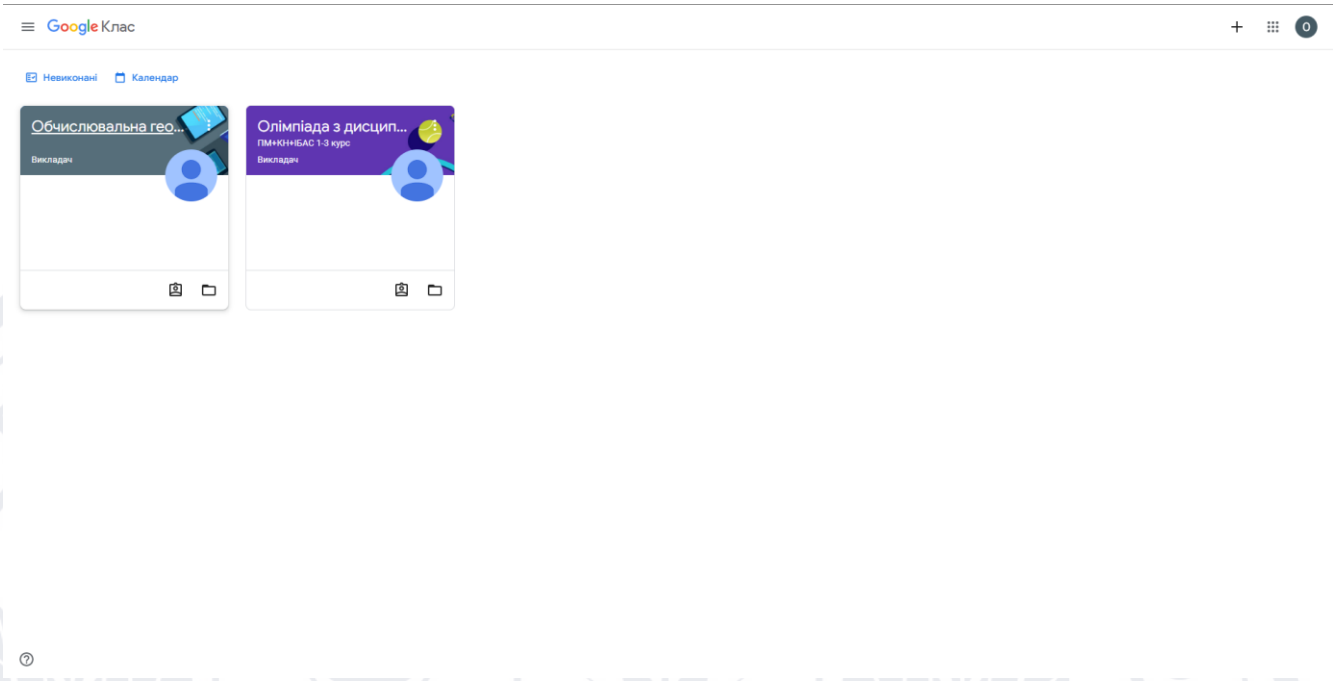

Рисунок 1.3 – Інтерфейс веб-сайту Google Classroom

1.3. Постановка задачі

Мета бакалаврської роботи полягає у створенні системи, що дозволить організовувати, редагувати та переглядати оцінки здобувачів вищої освіти, а також досягти можливості її горизонтального масштабування і безперебійності роботи при високих навантаженнях.

Для досягнення поставленої мети потрібно:

- проаналізувати існуючи системи;
- спроектувати архітектуру системи;
- спроектувати бази даних;
- визначити перелік інструментів, що будуть використовуватись;
- обґрунтувати вибір архітектури та інструментів;

Розділ 2.
АНАЛІЗ ТА ВИБІР АКТУАЛЬНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Вибір архітектури веб-додатку

Для високонавантажених систем і для тих, що повинні мати можливість до простого масштабування, прийнято використовувати мікросервісну архітектуру. Її суть полягає у тому, щоб розбити одну велику систему на декілька за певною ознакою. Це дає можливість паралельно від інших модернізувати окремий модуль, а також запускати безліч їх екземплярів на серверах, що дозволяє балансувати навантаження між ними. На противагу до мікросервісної архітектури, існує монолітна. В ній все знаходиться в одному місці і додавання нового функціоналу може впливати на інший. Оскільки ми плануємо створити систему, яку буде просто масштабувати горизонтально, то виберемо мікросервісну архітектуру.

Горизонтальна масштабованість – це можливість масштабувати обсяги системи за допомогою за допомоги створення нових сервісів до інфраструктури проекту. Такий підхід дозволяє більш заощадливо відноситись до ресурсів серверу, але таку архітектуру важко підтримувати і вона дає досить велике навантаження на мережеві пристрої серверу.

Вертикальна масштабованість – це додавання більше ресурсів на сервер, який не справляється з поточною завантаженістю. Такий підхід може дорого коштувати, оскільки зростає вартість оренди серверів і потрібно витрачати кошти на нові комплектуючі. Але таку систему, на відмінну від минулої, простіше підтримувати і спостерігати за її активністю, оскільки все знаходиться в одному місці.

Також, мікросервісна архітектура є безпечнішою для роботи ПЗ, а ніж монолітна. Наприклад, маємо фунціонал адміністрування сайту та роботи з оцінками, який працює на одному веб-сервері. Якщо один з них перестане працювати з деяких причин (кількість одночасних користувачів збільшилась, непередбачувані помилки в системі тощо), то інший також перестане працювати. З цього випливає, що їх потрібно розділяти і зробити їх роботою максимально незалежною.

Хорошою практикою вважається створювати окрему базу даних для кожного мікросервісу, оскільки найповільнішою частину будь якого масштабного проекту є саме БД. Тому, важливо декомпозувати велику систему на дрібні частини, виділяючи їх за спільною ознакою, і розташовувати їх окремо.

Але, якщо сильно розділяти систему і не давати можливість комунікації модулям, то отримаємо просто декілька монолітів. Спілкування може мати різний характер, але найголовнішими виділяють синхронні та асинхронні. В першому випадку, модулі передають та отримають інформацію за допомогою REST-клієнтів, а в другому – системи брокерів (Apache Kafka, RabbitMQ, JMS тощо).

2.2. Вибір інструментів для створення сайту.

Розглянемо інструменти, які будемо використовувати при створенні додатку, оскільки їх вибір напряму впливає на такі критерії, як продуктивність, здатність до підтримки та швидкість створення функціоналу.

Продуктивність проявляє себе у здатності оброблювати потоки даних в певну одиницю часу. Тому, інструменти мають підтримувати роботу в багатьох потоках і мати потужну систему.

Здатність до підтримки – можливість легко передавати код для інших розробників. Якщо інструмент буде занадто екзотичний або буде настільки новий, що документації по ній майже немає, то це призведе до неможливості нормальної підтримки коду розробникам. Отже, інструмент має бути добре відомий і популярний в IT-секторі, щоб можна було просто найти розробника, який зможе швидко приступити до підтримки сайту.

Інструмент також має бути настільки простим у використанні, наскільки це можливо. Адже вирішення проблем у системі має бути оперативним і не займати багато часу.

Тому, для створення системи було обрано наступні програмні засоби, які повністю задовольняють вищеперераховані критерії: Java 11 (LTS), Spring Framework, Maven, Angular, HTML, CSS, Bootstrap, PostgreSQL, Docker. Основне середовище розробки – IntelliJ IDEA (Ultimate Edition).

2.2.1. Java 11 (LTS)

Java – компільована об’єктно орієнтована мова програмування з підтримкою функціональної парадигми. Є найпотужнішою в плані обробки великих потоків інформації за рахунок Java Virtual Machine. JVM – це компонент Java Runtime Environment, який виконує відтворення коду, оперування змінними та пам’яттю. Оскільки вона існує досить довгий час, то модернізована вона до високого рівня, що дозволяє справлятись з високим навантаженням. Однак, цей компонент є дуже важким для освоєння, тому, для його тонкого конфігурування потрібно мати широке пізнання в його аспектах роботи.

Можливості в багатопоточності цієї мови дають змогу без проблем відтворювати певний код паралельно, що дуже сприяє створенню потужної

backend-системи сайту. Але її використання потребує більш потужного процесору (хоча б чотири ядра для малонавантаженного проекту) і більше оперативної пам’яті (мінімум вісім гігабайт) для того, щоб повністю розкрити потенціал використання цього механізму.

Саме ця версія була обрана з наступних причин.

По-перше, версія 11 має Long-Term-Support (Довготривала підтримка). Це означає, що таке маркування, за вказівками Oracle, є рекомендованим для використання у системі, для якої важлива стабільність. І, на відмінну версій без такого маркування, ця має швидку підтримку від розробників у разі виникнення певних проблем.

По-друге, ця версія має вдосконалені особливості мови, які були презентовані ще з восьмої версії, а саме підтримка функціональної парадигми програмування, реалізовано потужний механізм роботи з колекціями під назвою SteamAPI, а також багато покращень та оптимізацій в роботі JVM.

По-третє, як переконують в компанії JetBrains, ця версія є найпопулярнішою у використанні після Java 8. Можна запропонувати те, що восьма версія є найпопулярнішою через те, що в багатьох Ентерпрайз системах доволі дорого переписувати кодову базу на нову мову, тому, вони залишаються на старій. Але у нашому проекті немає такого обмеження, тому, будемо використовувати саме її.

2.2.3. Spring Framework

Для розробки backend частини сайту використовувати тільки одну мову програмування досить незручно, оскільки потрібно реалізовувати функціонал на

досить низькому рівні, тобто, створювати власну реалізацію обробки HTTP-запитів.

Для Java існують два основних способи розробки серверної частини застосунку – Java Enterprise Edition та Spring Framework. Перший є найстарішим і використовується тільки в підтримці Legacy проектів, де перехід на інший фреймворк або мову дорого обійдеться (банківські системи, сайти документообігу та ін.). На відміну від JEE, Spring Framework є більш сучасним, де створення серверної частини є на високому рівні абстракції.

Перша продуктивна версія Spring Framework вийшла ще у 2004 році і головним покликанням його створення було полегшити створення сайтів і відійти від розробки на низькому рівні, як в JEE. Перший час він був поза конкуренцією, оскільки мав багато недоліків та малу популярність, але з часом, коли все більше розробників повірило у цей проект, він почав містити в собі багато інших фреймворків, які доповнювали його роботу у різних аспектах. Наприклад, для формування веб-сторінок на стороні серверу, або модуль для роботи з базою даних. Наразі, використання у нових проектах JEE є моветоном, а перевага надається Spring Framework.

Основою роботи цього фреймворку є Dependency Injection. Це паттерн програмування, що описує можливість додавати певні елементи класу, але з мінімальною залежністю. Spring має багатий інструментарій для виконання подібних операцій.

Як зазначали вище, у Spring Framework є велика кількість допоміжних модулів. У нашій системі ми використаємо наступні: Spring Web, Spring Data, Spring Cloud, Spring Security та OpenFeign. Розглянемо їх призначення окремо.

Spring Web – використовується для автоматичного деплою проекту на веб-сервер (за замовченням – Tomcat). В JEE, для того, щоб додати проект на сервер,

потрібно зібрати веб-архів (зібрана програма в один архів в розширенням WAR), власноруч перенести його в спеціальну директорію веб-серверу та дочекатись поки він запустить його. Тепер, це робить самостійно Spring Web і цей процес спрощено в рази.

Також, цей модуль надає інструменти для створення API, такі як контролери, сервіси автоматичного конфігурування системи, логування повідомлень системи, веб-сервер тощо. За замовченням, надається веб-сервер Tomcat, але можна вибрати інший. Наприклад, Jetty або Undertow.

Tomcat – веб-сервер, що є контейнером для програми і протягом багатьох років є стандартом при створенні надійного програмного забезпечення. Розроблений компанією Apache ще у 1999 році, але досі має стабільну підтримку продукту. В плані швидкості роботи програє своїм конкурентам, але має більшу продуктивність.

Jetty – в аналогії з попереднім, також є сервлет-контейнером. Був створений у 1995 році компанією Eclipse, але досі має підтримку від ком'юніті та розробників. Але, на відміну від Tomcat, не має такого поширення у використанні в продуктивному середовищі. Також він є швидким в обробці інформації та комунікації з клієнтом. Різниця також в тому, що Apache Tomcat орієнтується на підтримку актуальності останніх специфікацій, а Eclipse Jetty – на потреби спільноти і надає перевагу покращення продуктивності веб-серверу.

Undertow – наймолодший з перелічених веб-серверів, який вперше отримав релізну версію у 2013 році. Головною ціллю розробників при створенні цього контейнеру була простота, максимальна швидкість та мінімальна вага і вони успішно цього досягли. Його швидкість запуску, обробки та комунікації переважають над Jetty та Tomcat, але має проблеми в продуктивності. Наприклад, він не може приймати дуже великі об'єми даних.

Розглянемо порівняльні діаграми цих контейнерів. Перший параметр, за яким порівняємо, це середнє використання оперативної пам’яті пристрою при запуску того самого ПЗ.

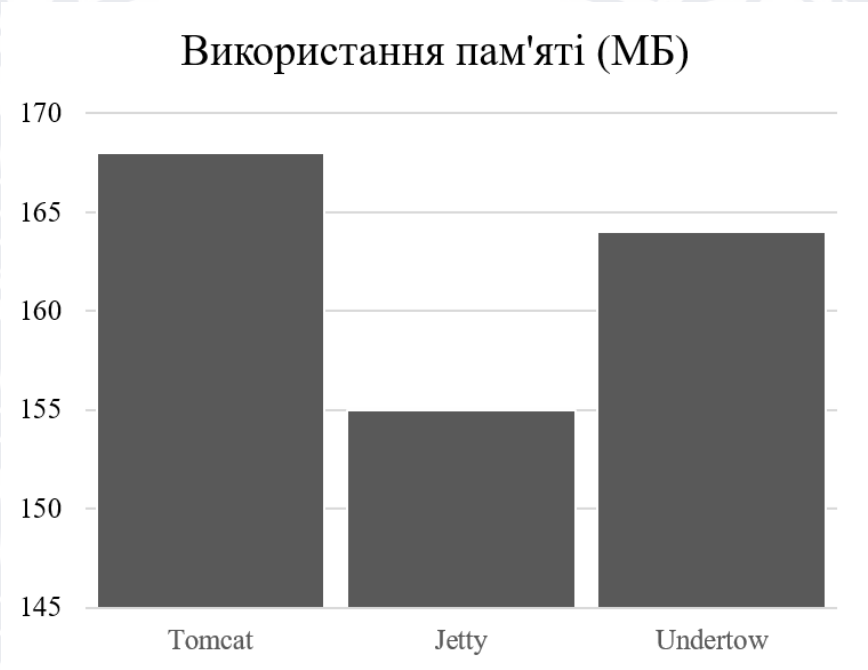

Рисунок 2.1 – Діаграма використання оперативної пам’яті веб-серверами

Наступним параметром розглянемо середню швидкість обробки запитів в секунду, оскільки є важливим параметром при створення швидкого API.

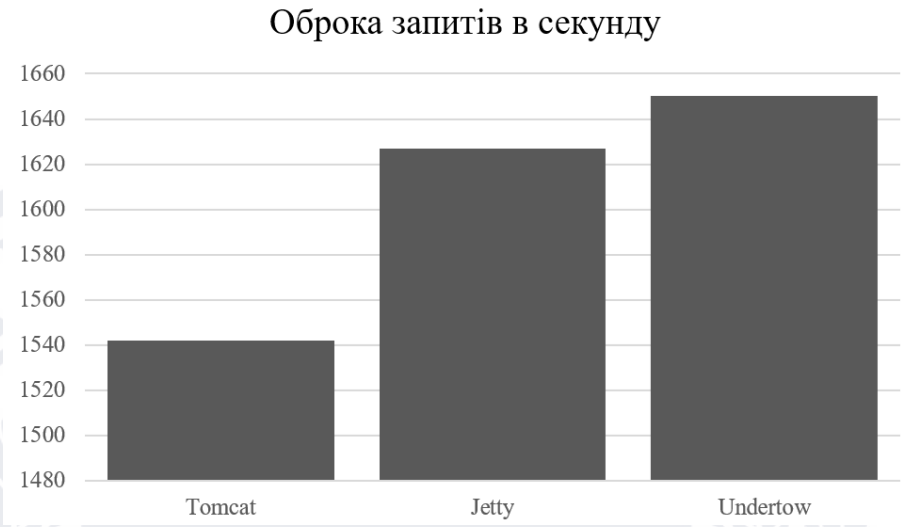

Рисунок 2.2 – Діаграма кількості обробки запитів в секунду

Як бачимо, в деяких аспектах один веб-сервер краще за інших, а в інших – гірше. Можна звідси винести таке правило – якщо потрібна надійність, то це Tomcat; якщо швидкість, то Jetty, а якщо простота, то Undertow. В нашому випадку, використаємо веб-сервер за замовченням (Tomcat), через його надійність роботи та високу потужність.

Spring Data. Надає інструменти для роботи з базою даних. З його допомогою, розробнику не потрібно переключатись між абстракціями БД і програмного коду, оскільки отримання інформації відбувається на рівні об’єктів. Наприклад, ми можемо в коді оголосити клас, в якому описуємо потрібні поля, а фреймворк на момент запуску може самостійно створити таблиці в БД і надати інтерфейс для роботи з ними.

Сам модуль дає змогу працювати з Hibernate. Це Object Relational Mapping фреймворк Java, який дозволяє створювати запити до бази даних на рівні об’єктів. ORM, в свою чергу, дозволяє переводити SQL відповідь в певний об’єкт на основі назви полів. Використовує він JDBC (Java Database Connectivity), який знаходиться на найнижчому рівні абстракції. Наприклад, в JDBC потрібно

власноруч налаштовувати підключення до БД, створювати сесії, транзакції та самостійно реалізовувати функціонал запису інформації в об’єкт з відповіді на SQL запит.

Однак, Hibernate надає сумнівні технології, одна з яких можливість ліниво завантажувати колекції моделі. Вона дає можливість додавати колекцію об’єктів в той, що отримується з бази даних, проте, доступна вона тільки коли відкрита сесія. Після закриття, звернення до неї викликатиме помилки. Такий підхід реалізовано для того, щоб зменшити навантаження на сервер, але бувають випадки, коли потрібно зменшити кількість запитів до БД і об’єкт довгий час зберігається в пам’яті і вирішення цієї проблеми може призвести до проблем з продуктивністю системи.

Spring Security – допомагає додати інтерфейси налаштування безпеки для API. Наприклад, додати користувачів і ролі для них, і на основі цих ролей розмежовувати доступ до кінцевих точок backend. Слід зазначити, що в нашій системі цей фреймворк буде використовуватись мінімально, оскільки буде реалізована власна система безпеки на основі токенізації запитів.

Spring Cloud та OpenFeign. Spring Cloud надає інструменти, які допомагають швидко реалізувати мікросервісну архітектуру системи. Серед них є такі:

1. Інструмент реєстрації та пошуку модулів.
2. Система балансування навантаження.
3. Обмін повідомленнями між сервісами.
4. Автоматичні вимикачі функціоналу.

OpenFeign – це декларативний REST-клієнт, який лаконічно влаштовується в Spring Framework екосистему (реалізує його функціонал на свій лад) та може виконувати HTTP-запити на будь який API. Декларативним він є через те, що

розробник не створює самостійно підключення, реалізацію та обробку відповідей, а тільки вказує місце розташування інформації та тип інформації що має повернутись, а клієнт вже сам зробить відповідні дії для отримання результату та обробить відповідь належним чином. Він має аналогічну функціональність з Spring Data, що дозволяє переводити отриману інформацію будь якого формату в формат POJO (Plain Old Java Object – звичайний об’єкт Java). Наприклад, він може форматувати JSON або XML відповідь за допомогою вбудованої бібліотеки Jackson.

2.2.4. Maven

Окремої уваги заслуговує така система збірки проекту, як Maven. З її допомогою можна підключати зовнішні залежності до проекту, налаштувати процес формування архіву проекту, а також запускати його. Існують також інші такі системи, наприклад, Ant і Gradle. Ant використовується в доволі старих проектах і на сьогодні рекомендується використовувати Maven або Gradle. Gradle відрізняється від Maven тим, що всі команди в ньому пишуться у форматі JSON, а в Maven – XML.

Використовувати системи збірки є необхідним, оскільки це допомагає без проблем додавати зовнішні фреймворки і бібліотеки. Наприклад, той самий Spring Framework можна додати за допомогою них. Після оголошення того, що проект повинен мати на нього залежність, Maven перевірить, чи він завантажений на пристрої локально і, якщо ні, завантажить його з репозиторію.

Також, ця система може допомогти автоматично збирати проект у потрібний формат і запускати його в Docker-контейнері (розглянемо його згодом) та запускати.

2.2.5. Angular. HTML/CSS. Bootstrap

Після створення API сайту, потрібно створити спосіб для зручного отримання інформації з нього користувачеві. Для цього необхідно створити інтерфейс користувача (UI). Оскільки ми в API будемо використовувати REST підхід, то потрібно використати фреймворк для побудови frontend сайту. Для цього виберемо Angular з наступних причин.

По-перше, в ньому можна створювати компоненти, які можна повторно використовувати в різних елементах сайту. Наприклад, «шапку» сайту можна створити один раз і потім вказати, що вона має бути на кожній сторінці. І Angular це буде робити автоматично.

По-друге, проект легко переносити на різні платформи. Це є перевагою, оскільки, як ми зазначали вище, маємо мікросервісну архітектуру.

HTML та CSS є невід'ємним стандартом побудови UI сайту і їх використання є апріорі. Звісно, можна за допомогою Java власноруч створювати сторінки у форматі JSP або інших, що підтримують шаблонізацію, і створювати їх відображення на стороні backend, але такий підхід створить високу залежність між frontend та backend сайту, що ускладнить розробку і додатково навантажить сервер.

Перевага у Angular над іншими frontend фреймворками у тому, що він простіший у вивчені та має більш лаконічний код. А те, що він використовує TypeScript, додає безпеки для коду, оскільки звичайний JS не строго типізований і це спонукає до виникнення маси проблем. TypeScript строго типізований і це додає лаконічність коду і безпеки його відтворення.

Bootstap – open-source фреймворк, який створений для полегшення створення UI сайту. Його доцільно використовувати тоді, коли потрібно з мінімальними навичками в frontend розробці створити на основі існуючих шаблонів власні. Наприклад, коли backend розробнику потрібно створити інтерфейс користувача без поглиблення у знання їх розробки. Оскільки він добре документований і має безліч прикладів створення різних компонентів сайту, то це лише полегшує користування ним.

Також він допомагає створювати сайти, які будуть правильно відображатись на різних платформах. Bootstrap має два основні поняття – row та col. Row – це деякий рядок, якому мають зберігатись колонки (col). Ширина цього рядку умовно ділиться на 12 частин і в колонках вказується скільки таких частин вона буде займати. Наприклад, якщо дві колонки в рядку матимуть ширину 6, то вони будуть порівну ділити цей рядок між собою.

2.2.6. PostgreSQL

У якості системи управління баз даних виберемо PostgreSQL, оскільки вона себе добре зарекомендувала у високо навантажених системах. Також, для її використання, не потрібно ніяких ліцензій і вона повністю безкоштовна для використання навіть для комерційних проектів. Так як ми створимо мікросервісну архітектуру, то будемо використовувати декілька баз даних, деякі з яких будуть більш завантажені через більшу кількість таблиць. Також, вона має велику підтримку зі сторони прихильників, що дозволить швидко вирішувати проблеми. Ці всі критерії запевняють, що PostgreSQL цілком підходить для її використання в нашій системі.

2.2.7. Docker

Docker – це платформа, що дозволяє контейнеризувати елементи системи і зробити їх незалежними. Такий підхід дозволяє просто переносити сервіси в мікросервісній архітектурі, а також полегшити роботу з ними. Наприклад, можна створити контейнер з базою даних і, після цього, її буде легко реплікувати і поширювати.

Контейнер – це незалежна одиниця платформи, яка містить певний функціонал, який можна запустити. Всередині він містить усю важливу інформацію та файли для правильного відтворення програми, що робить його ізольованим і він не займає багато пам’яті.

Образ – це деякий шаблон, який містить інструкції, на основі якого створюються контейнери.

Volume – деякий набір зовнішньої інформації, який потрібен для контейнера. Використовується для збереження стану контейнеру після завершення його роботи, або поширювати інформацію між контейнерами.

В нашій системі, усі елементи будуть знаходитись в контейнерах, що дозволить їх ізолювати і уникнути зовнішнього впливу на них. Ізольованість досягається тим, що в кожному контейнері знаходиться власна мінімальна операційна система. Порти, в свою чергу, в контейнера закриті і отримати доступ до нього через localhost не вийде.

2.2.8 IntelliJ IDEA

IntelliJ IDEA – це середовище розробки, розроблене компанією JetBrains та орієнтоване для розробки програмного продукту на мові програмування Java, але має підтримку і для інших. Надає можливість швидко налаштовувати проект і містить внутрішньо розміщені інструменти розробки, такі як JDK, Maven тощо. Однак, безкоштовна версія є дуже обмеженою і, наприклад, не дає змоги належним чином працювати зі Spring Framework, Docker та формуванням звітів. Для того, щоб з цими елементами можна було працювати в IDE, потрібно оформити план на Ultimate Edition, що дозволяє працювати з вище перерахованими технологіями, а також додавати інструменти для роботи з багатьма іншими, наприклад, Angular, Docker Environment та багато інших.

Розділ 3.
РОЗРОБКА ТА ПРОЕКТУВАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

3.1. Архітектура системи

Розглянемо спрощену діаграму додатку.

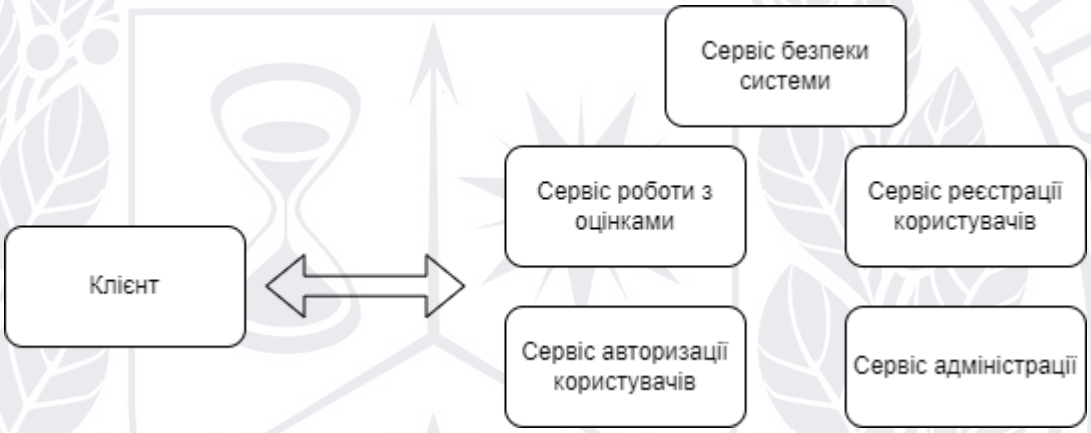

Рисунок 3.1 – Діаграма архітектури системи

У спрощеному варіанті, архітектура додатку має такий вигляд. Розберемо всі елементи окремо.

Клієнт. У якості клієнта виступає браузер, з якого йдуть запити на API і приймаються та оброблюються відповіді від серверу. Кількість клієнтів необмежена, що досягається використанням Spring REST.

Сервіс безпеки системи. Сервіс, що інкапсулює роботу з безпеки використання сайту. Його основна робота – це надавати певним користувачам токени, якими клієнт буде підтверджувати своє право користуватись іншими сервісами.

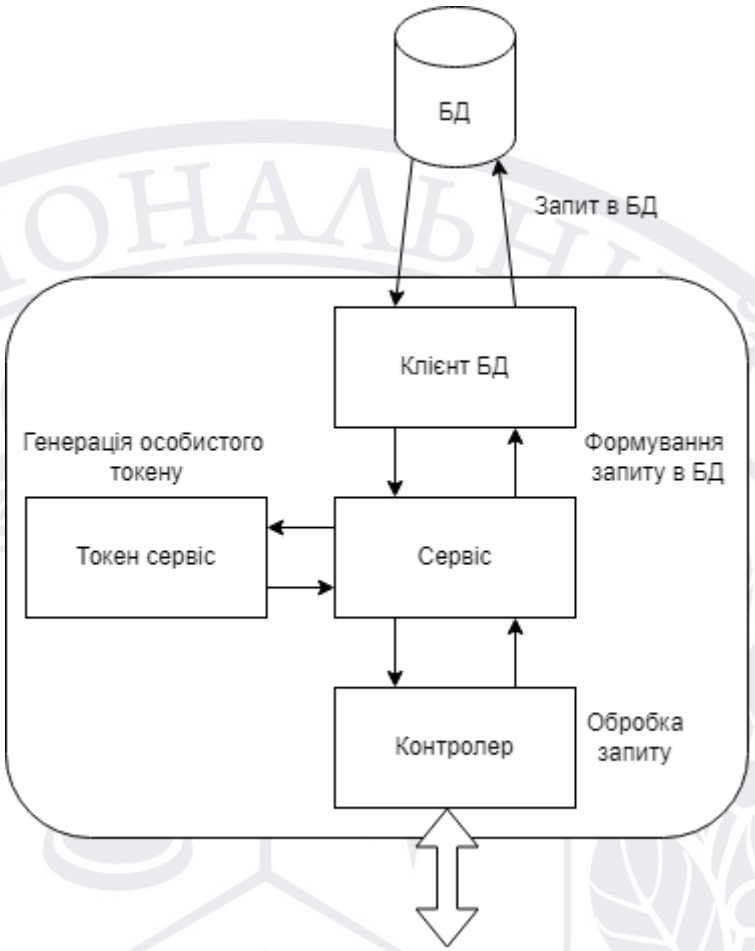

Рисунок 3.2 – Діаграма сервісу безпеки системи

Сервіс має контролер, який приймає запити з інших сервісів. Отримати доступ до нього може тільки той, хто матиме токен, що підтверджує його належність до системи. Інакше, сервіс не оброблюватиме такі запити і буде сповіщавати про небажану спробу доступу.

Коли сервіс авторизується, він поверне певний запит на контролер. На основі нього, вирішиться, як його обробити і яку сформувати на це відповідь. Сам сервіс вміє додавати користувачів до своєї БД і формувати для них токени. Алгоритм його роботи представимо у вигляді діаграми станів.

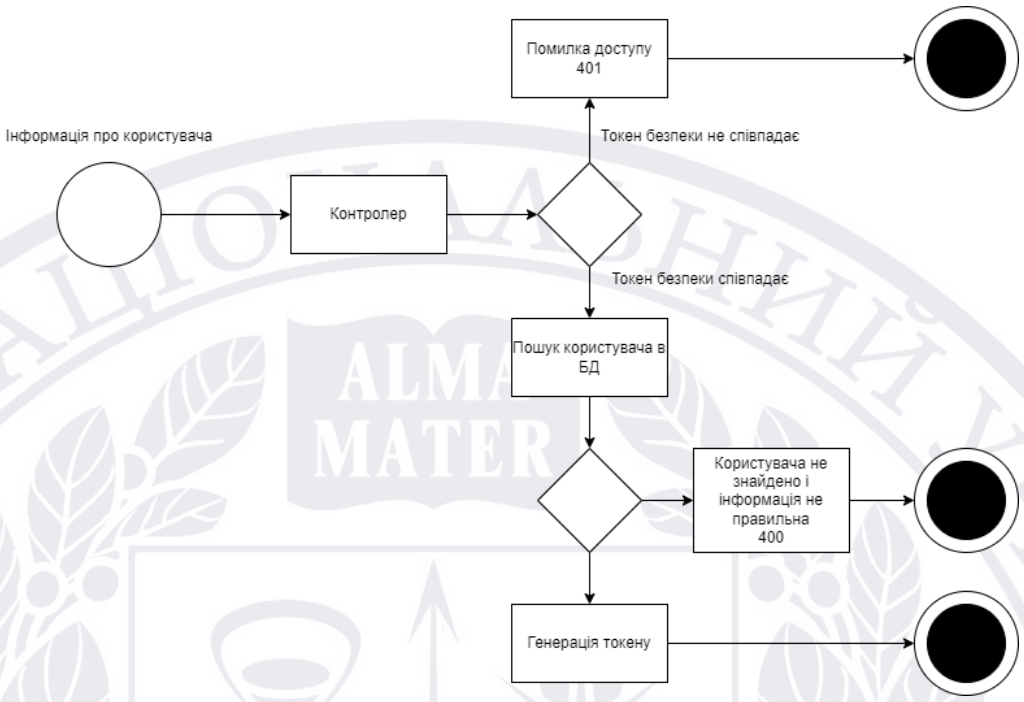

Рисунок 3.3 – Алгоритм формування токену

Сервіс роботи з оцінками. Головний сервіс, в якому знаходиться вся основна робота з оцінками. Він, в свою чергу, спілкується з сервісом безпеки для того, щоб переконатись, що користувач з вказаним токеном дійсно є авторизованим користувачем і знаходиться в базі даних. Тільки після цього він дозволяє користуватись його функціоналом. Алгоритм цієї перевірки представлено на діаграмі:

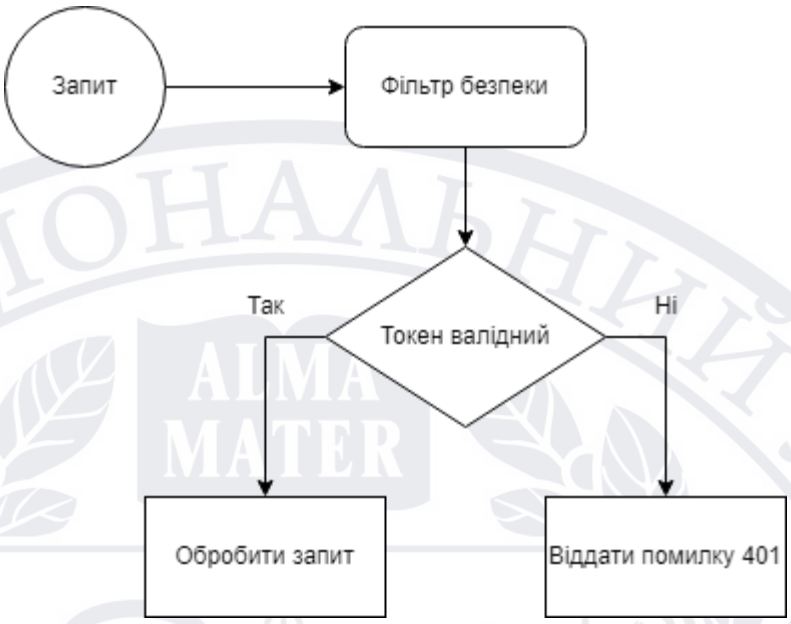

Рисунок 3.4 – Алгоритм роботи фільтру безпеки

Тільки того, як токен буде підтверджено, система почне перенаправляти обробку на певну кінцеву точку.

Сервіс реєстрації та сервіс авторизації. Перший сервіс відповідає за валідацію введених даних для реєстрації користувача та, у випадку правильності введеної інформації, передасть цю інформацю у сервіс безпеки та сервіс роботи оцінками. Для останнього сервісу ця інформація потрібна для того, щоб додати користувача з вказанною роллю до бази даних і мати можливість прив’язувати його до певних таблиць. Наприклад, студентів до груп, а викладачів – до дисциплін.

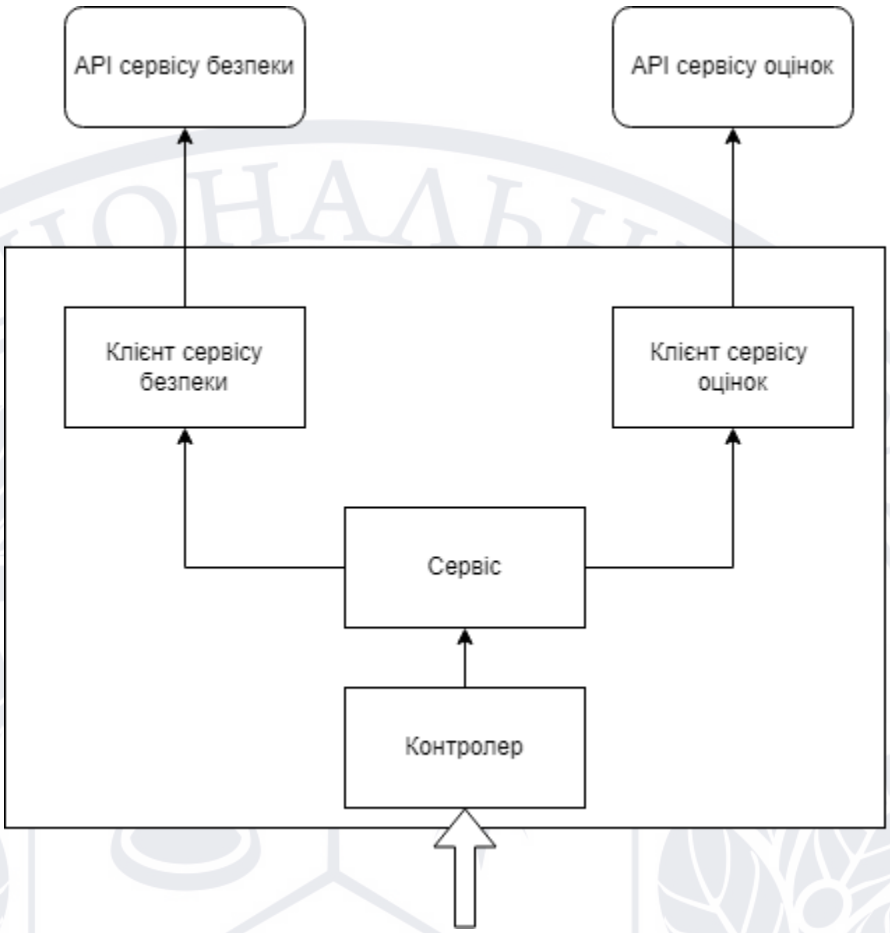

Рисунок 3.5 – Діаграма роботи сервіса реєстрації

Сервіс авторизації, в свою чергу, є посередником, який приймає запит на авторизацію від клієнта, віддає ці дані в сервіс безпеки, а він перевіряє правильність введених даних і віддає токен.

Сервіс адміністрації. Потрібен для роботи адміністраторів сайту і виконує роль зберігання та відтворення інформації про запити та звіти про помилки. Ним мають право користуватись тільки користувачі з роллю адміністратора. Сам сервіс ізольований від інших і не впливає на їх роботу.

Більше подробиць у архітектурі системи та способи їх комунікацій можна переглянути в наступній діаграмі.

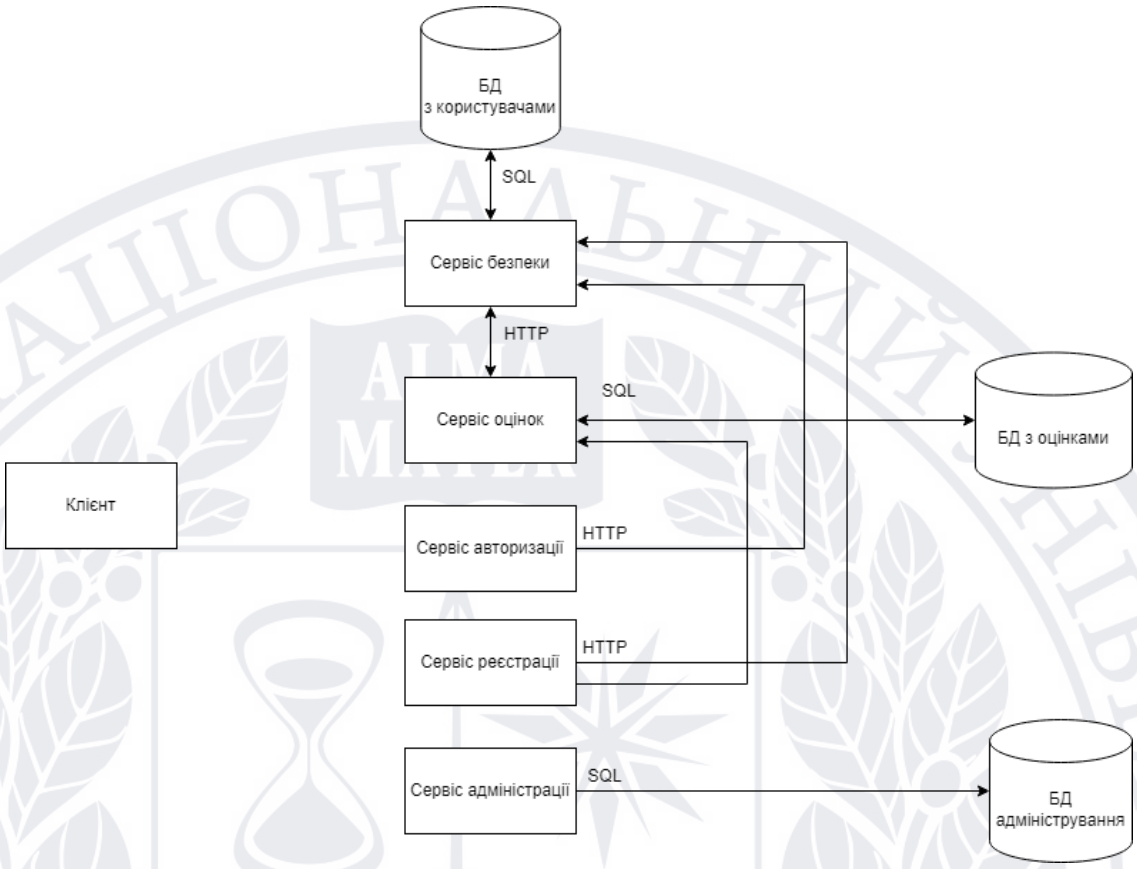

Рисунок 3.6 – Детальна діаграма архітектури системи

Сервіси комунікують між собою за допомогою REST-клієнтів, що додає безпечність та надійність переданим повідомленням, оскільки ці клієнти спостерігають за процесом виконання операції і тільки тоді, коли вона буде успішно виконана, перейде до обробки інших операцій. Оскільки на кожного користувача створюється нова сесія користування, то це означає, що клієнт працює в кожній з таких сесій незалежно і робота їх паралельна.

Кожен сервіс теж має внутрішню архітектуру побудови під назвою Three Tier, де сервіс поділений на три прошарки – Presentation Layer, Bussiness Layer та Data Access Layer. Важливо не плутати з MVC, який в контексті Spring має інакший принцип побудови ПЗ. Spring MVC призначений для побудови монолітної системи, де UI та API знаходяться в одному місці та сервер займається створенням

веб-сторінок. Наприклад, при запиті, Spring MVC віддасть сторінку, а REST – відповідь у вказаному форматі (JSON, XML тощо).

Presentation Layer (прошарок представлення) – відповідає за представлення інформації для клієнта, який її запросив. В проекті, цю роль виконують контролери, які приймають запити, передають їх на рівень нижче та повертають результат.

Bussiness Layer (прошарок логіки) – оброблює вхідну інформацію, віддає запит на виконання операціїї на нижчий рівень та віддає результат на верхній.

Data Access Layer (прошарок даних) – приймає інформацію про необхідну операцію з верхнього рівня, робить запит в певне джерело інформації та віддає її на назад на верхній рівень.

Такий підхід є доцільним для проектування системи тому, що він розділяє зону відповідальності елементів і дозволяє притримуватись принципу SOLID, де кожен відповідає за певну річ. Також такий проект матиме кращу читабельність, оскільки розробник знатиме, за що компонент відповідатиме. Наприклад, GradeRepository відповідає за отримання або додавання даних в БД, а GradeService – обробка оцінок.

3.2. Схема бази даних

Як було показано на діаграмах раніше, існує три бази даних – БД з користувачами системи, з оцінками та БД адміністрування. Розглянемо їх окремо.

БД з користувачами. Зберігає в собі інформацію про користувачів системи, а також про токени цих користувачів. Схема цієї бази матиме наступний вигляд.

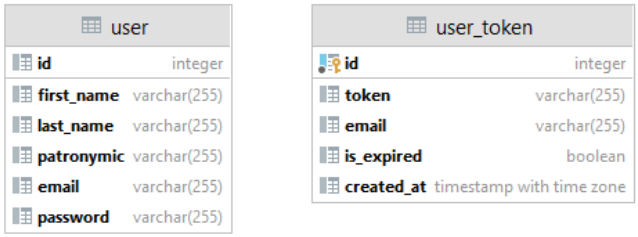

Рисунок 3.7 – Схема бази даних користувачів

БД з оцінками. В цій базі розташована інформація про оцінки студентів, а також додаткову інформацію про студентів, викладачів, групи та дисципліни. Схема матиме наступний вигляд.

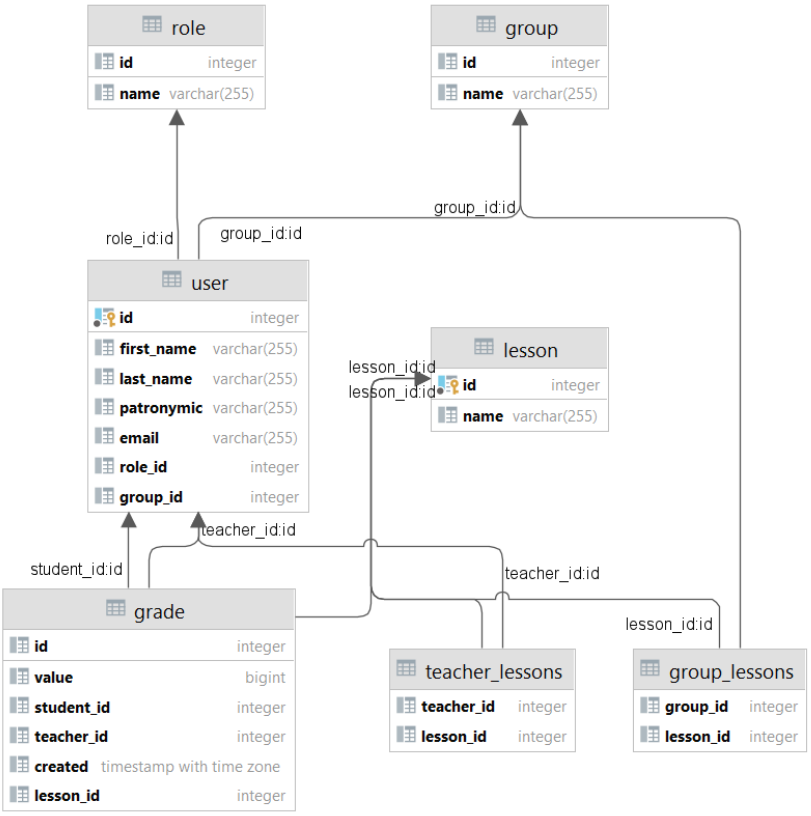

Рисунок 3.8 – Схема бази даних з оцінками

БД адміністрації. Зберігає інформацію про запити користувачів та повідомлення про помилки в роботі сайту. Схема матиме наступний вигляд.

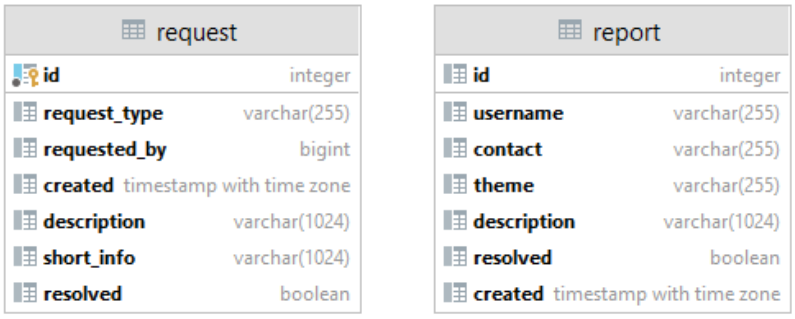

Рис. 3.9 – Схема бази даних адміністрації

3.3. Розробка API сайту

Тепер, коли ми визначились з архітектурою, приступимо до створення API. Спершу потрібно створити модель, які реплікують дані з БД. Наприклад, модель для таблиці оцінок матиме наступний вигляд:

```java
@Data
@Entity
@Builder
@Table(schema = "grades", name = "grade")
@AllArgsConstructor
@NoArgsConstructor
public class Grade implements Comparable<Grade> {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private Long value;

    @ManyToOne
    private User student;

    @ManyToOne
    private User teacher;

    @ManyToOne
    private Lesson lesson;

    private LocalDateTime created;

    @Override
    public int compareTo(@NotNull Grade that) {
        if (created == null || that.created == null) {
            return 0;
        }
        return created.compareTo(that.created);
    }
}
```

Рисунок 3.10 – Приклад моделі для таблиці оцінок

В ній ми вказали потрібні поля, які мають братись з колонок таблиці, а також зв’язки між іншими таблицями. Тепер, коли є модель, потрібно створити інтерфейс, який буде виконувати операції з базою даних для цієї моделі і

переводити результат з БД в потрібну модель системи. Проста реалізація такого інтерфейсу матиме наступний вигляд:

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

}
```

Рисунок 3.11 – Інтерфейс доступу до бази даних

Тепер Spring Data при запуску, створить базові запити до БД, наприклад, отримати сутність по ідентифікатору, зберегти її і видалити. Оскільки це елемент Spring Framework, то його можна додавати в прошарок сервісу за допомоги Dependency Injection.

Сервісний прошарок буде містити логіку обробки та вихідної інформації. Для вхідної – це валідація і перенаправлення запиту до БД, а вихідної – це отримання інформації з бази даних та перетворення її у зручний формат для клієнта.

В якості прошарку представлення виступатиме контролер, в якому оголошується кінцева точка, який шлях і операцію, що вказані в HTTP-запиті, має обробляти. Вона, в свою чергу, прийматиме запит, забиратиме, за необхідності, певну інформацію з запиту і віддавати її на прошарок логіки. Після цього віддаватиме інформацію клієнтові.

Створивши за таким принципом сервіс, нам потрібно реалізувати їх зв’язки з іншими. Для цього використаємо REST-клієнт під назвою Feign. Він виконує HTTP-запити на будь яке посилання та отримує інформацію з нього у вказаному

вигляді. Такий клієнт, який буде отримувати інформацію з сервісу безпеки, матиме наступний вигляд.

```
@FeignClient(name = "auth-api-client", url = "${HOST_IP}:8380/api/user")
public interface AuthClient {

    @GetMapping("valid-token")
    Boolean isValidToken(@RequestParam("token") String token, @RequestHeader("TOKEN") String trustToken);

    @GetMapping("user-email")
    String userEmailByToken(@RequestParam("token") String token, @RequestHeader("TOKEN") String trustToken);

}
```

Рисунок 3.12 – Приклад REST-клієнту для сервісу безпеки

Його перевага в тому, що ми декларативно вказуємо джерело інформації та очікуваний результат, а Feign вже зробить це підключення та отримання самостійно. Але головне пам’ятати те, що дані, які прийдуть в результаті, мають співпадати з вказаним, інакше, можлива помилка або втрата даних. Для уникнення цього, потрібно обережно вказувати очікуваний результат та більш детально дослідити API.

3.4. Розробка UI сайту

Тепер, коли API готове, розробимо UI нашого сайту. Для цього створимо Angular-проект і ньому створимо компоненти, що будуть отримувати інформацію з нашого API. Наприклад, сервіс отримання інформації про оцінки буде мати наступний вигляд.

```
@Injectable()
export class GradesApiService {

    url = 'http://localhost:8180/api/student/grade-sum?'
    gradeUrl = 'http://localhost:8180/api/grade/'

    constructor(private http: HttpClient,
                private cookieService: CookieService) {
    }

    get(lessonId, groupId): Observable<any> {
        const token = this.cookieService.get('token');
        if (!token) {
            throw new Error("Token is empty")
        }
        return this.http.get( url: this.url + 'lessonId=' + lessonId + '&groupId=' + groupId);
    }

    studentGrades(userId: bigint) {
        return this.http.get( url: this.gradeUrl + userId);
    }

}
```

Рисунок 3.13 – Приклад сервісу отримання інформації з API

Інтерфейс сайту матиме наступний вигляд.

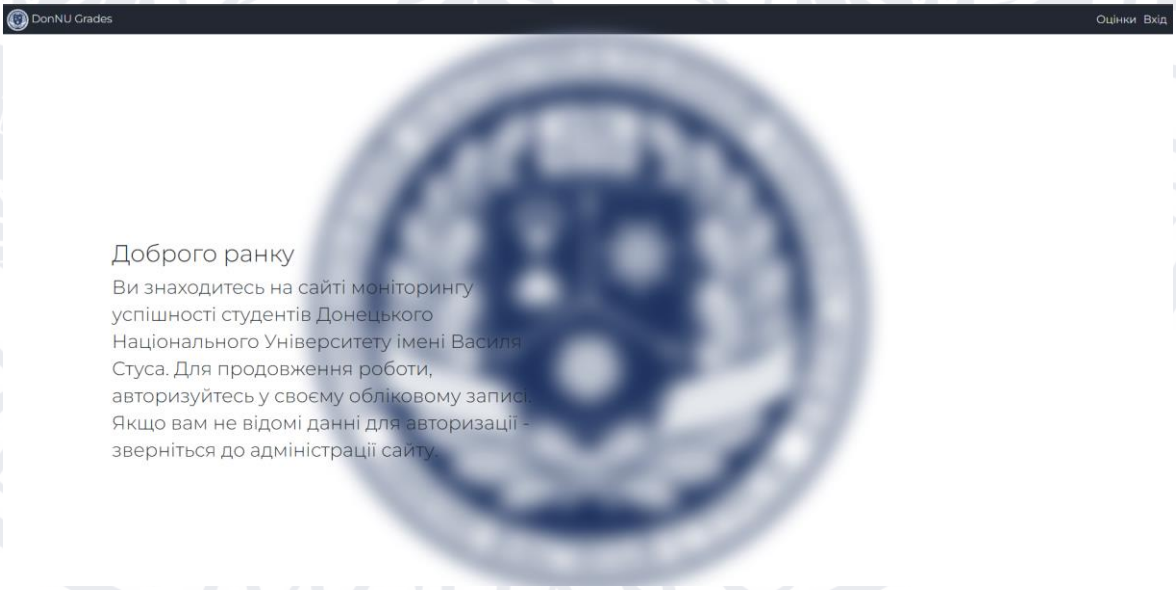

Рисунок 3.14 – Головна сторінка сайту

Тут ми бачимо головну сторінку, на якій знаходиться привітання в залежності від теперішнього часу. Якщо користувач не авторизований на сайті, то доступ до оцінок, особистого кабінету буде заблокований. Сторінка авторизації має наступний вигляд.

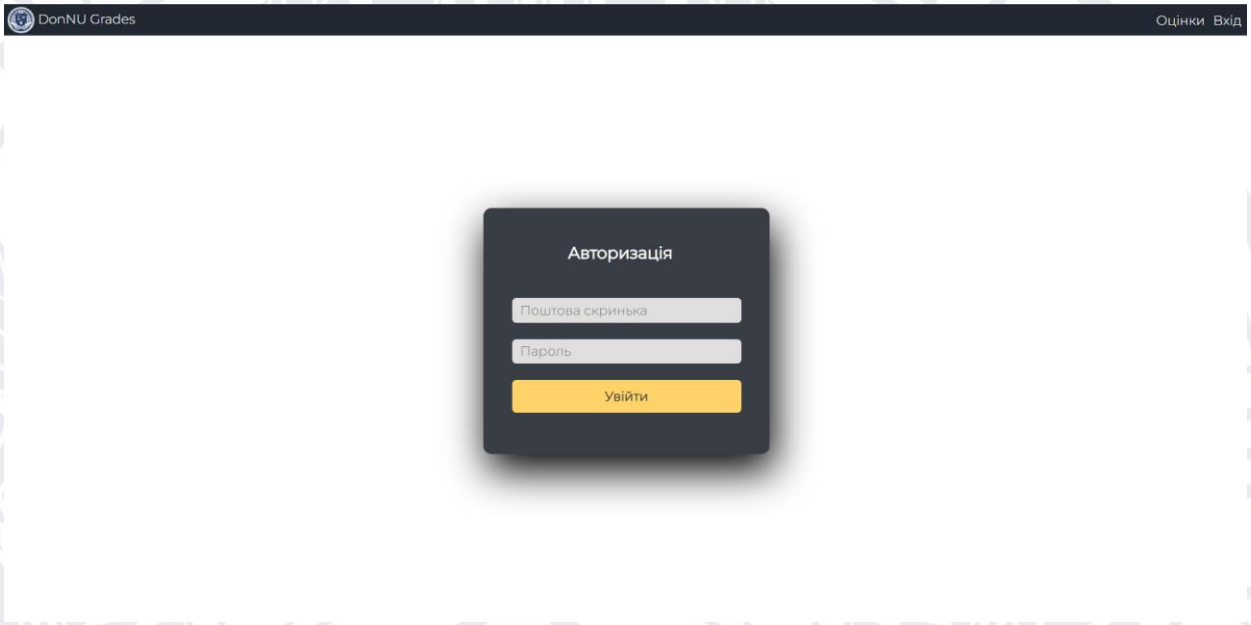

Рисунок 3.15 – Сторінка авторизації

На цій сторінці вводимо поштову скриньку та свій пароль, після чого отримаємо унікальний токен, який дозволить користуватись системою. Зайдемо на сайт під роллю студента і перейдемо в особистий кабінет.

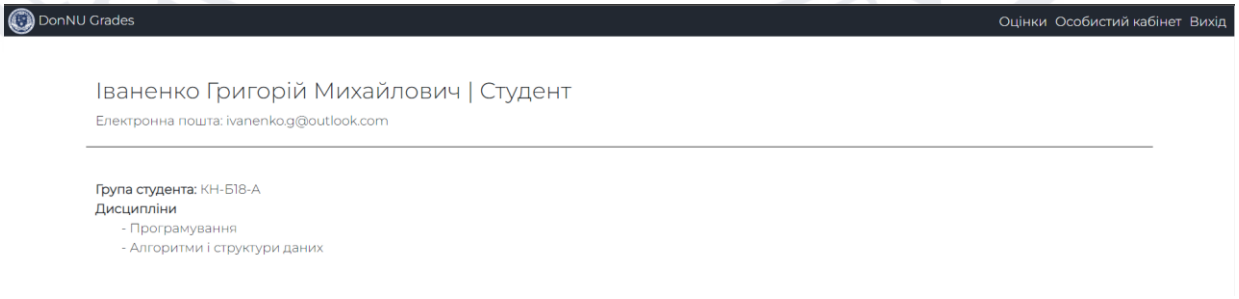

Рисунок 3.16 – Особистий кабінет користувача

В особистому кабінеті знаходиться інформація про користувача, і, в залежності від його ролі, вона відрізняється. Для студента будуть дані про групу, в якій він навчається і список дисциплін. Для викладача в цьому блоці буде інформація про дисципліни, які він викладає та в яких групах. Перейдемо в список оцінок.

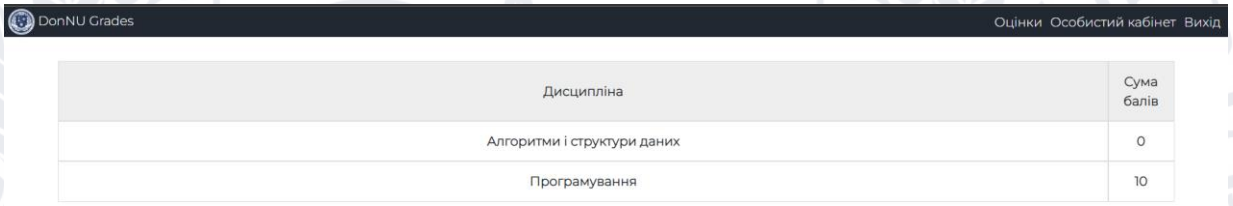

Рисунок 3.17 – Список оцінок поточного студента

Тут студент може переглянути список дисциплін, які йому викладаються і суму балів, яку він набрав на теперішній момент. Якщо натиснути на назву дисципліни, то можна побачити список цих оцінок.

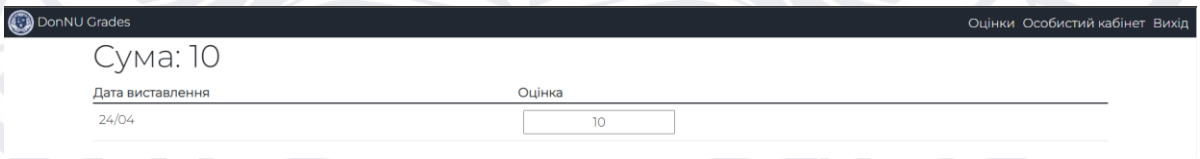

Рисунок 3.19 – Інформація про оцінки з дисципліни

Тут студент може побачити всі оцінки з дисципліни окремо. Тепер авторизуємось на сайті під роллю викладача та перейдемо у список оцінок.

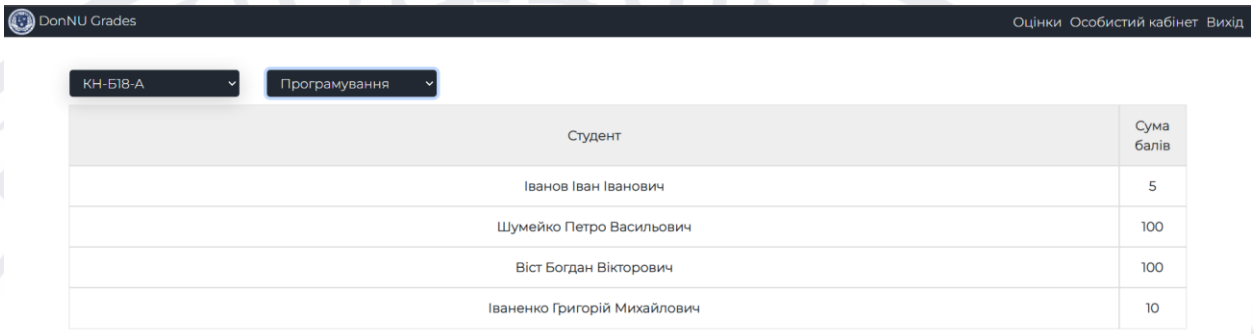

Рисунок 3.18 – Список оцінок для викладача

Для викладача існують фільтри по групам і дисциплінах, щоб зменшити обсяг інформації, що одночасно знаходиться на екрані. Ввівши потрібні параметри, користувач побачить список студентів та суму балів, яку він набрав з вказаної дисципліни. В аналогії зі студентом, викладач при переході на певного студента також побачить більш детальну інформацію про оцінки, але вже зможе їх редагувати.

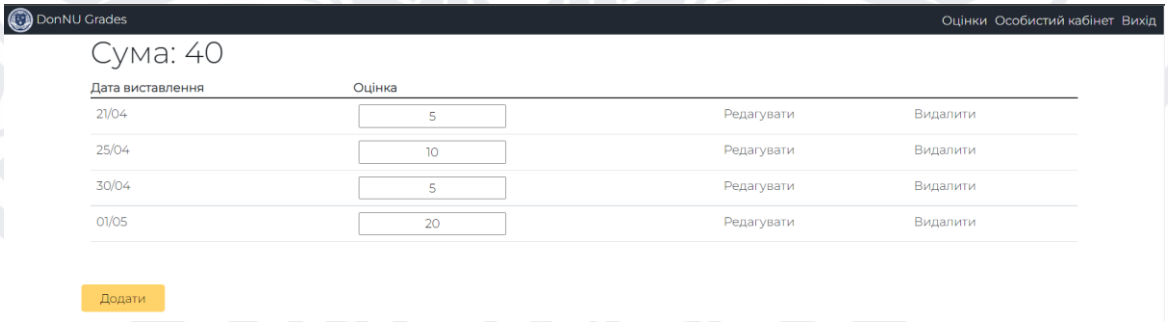

Рисунок 3.20 – Інформація про оцінки для викладача

Як бачимо, є операції з додавання, редагування та видалення оцінок. Ці операції недоступні для студента зі зрозумілих причин.

Також сайт підтримує зворотній зв’язок з адміністрацією. Його можна використати для двох цілей – створити запит на певну операцію і створити звіт про знайдену помилку. Спробуємо створити якийсь запит. Для цього потрібно спуститись в кінець сайту та натиснути на перше посилання.

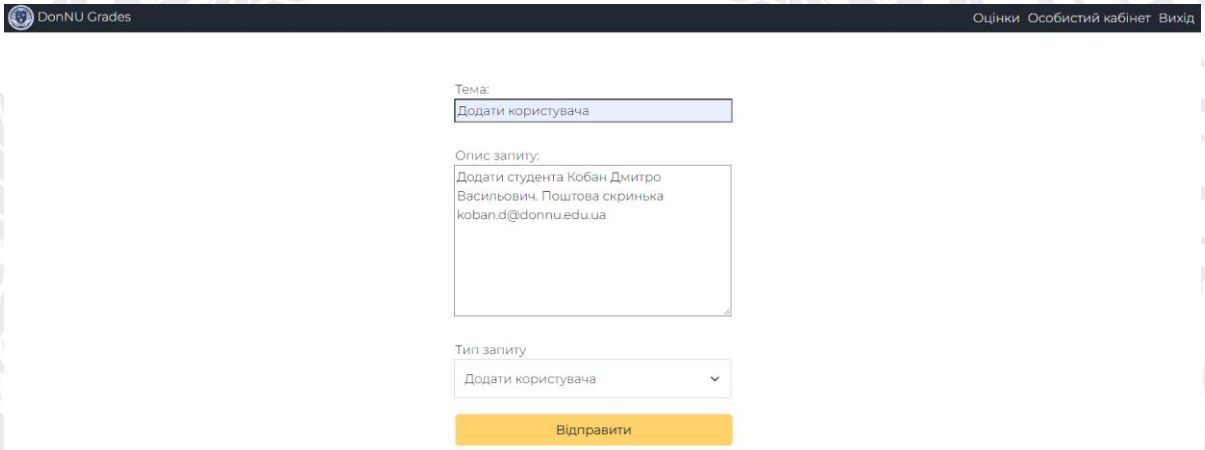

Рис. 3.21 – Запит на додавання користувача в систему

В цій формі потрібно вказати тему, опис запиту і тип запиту. Тип запиту потрібно вказувати для того, щоб полегшити їх сортування адміністраторами сайту. Відправимо і пізніше розглянемо панель адміністратора. А зараз створимо звіт про помилку. Для цього внизу відкриваємо друге посилання і бачимо наступну форму.

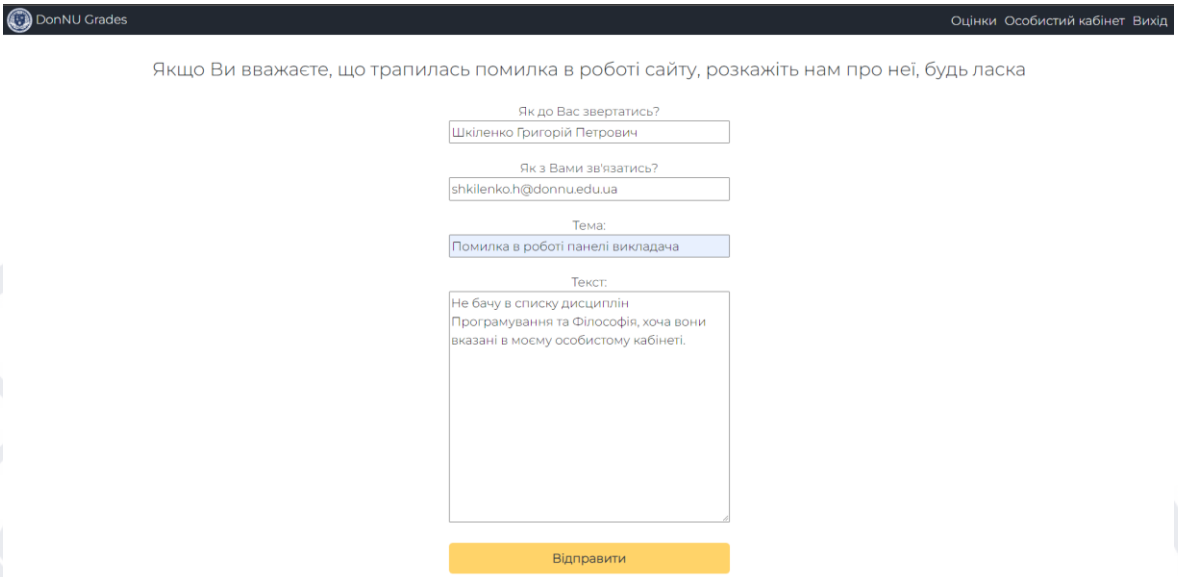

Рисунок 3.22 – Форма звіту про помилку

Відправимо звіт, авторизуємось під адміністратором і перейдемо на його панель.

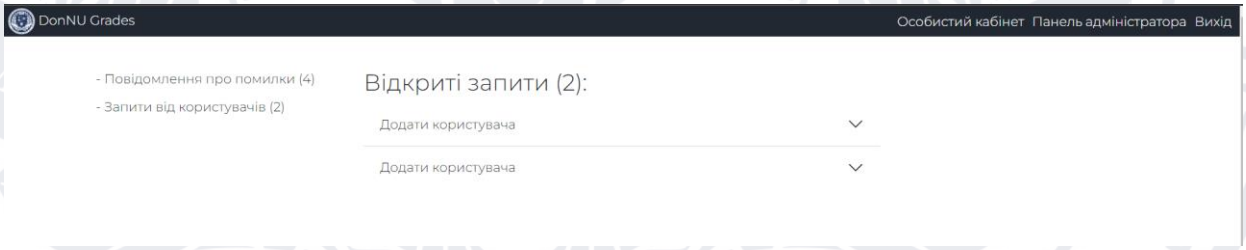

Рисунок 3.23 – Панель адміністратора

В цій панелі ми бачимо чергу невирішених запитів та звітів. Розкриємо запит і побачимо недавно створений нами запит.

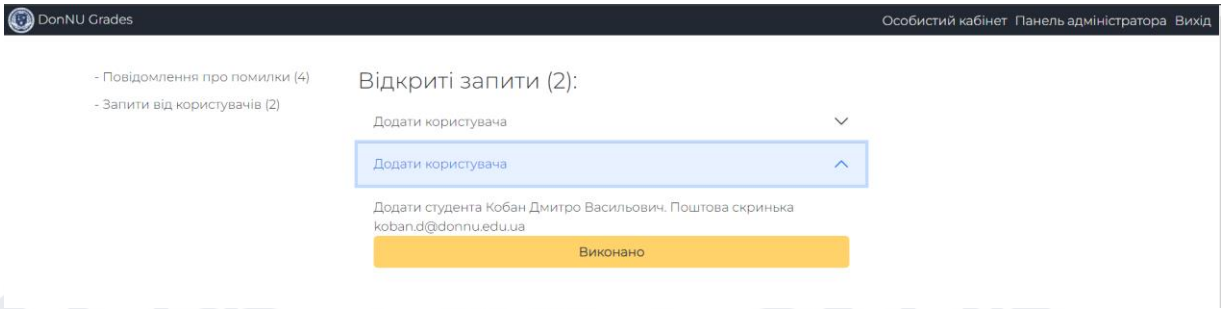

Рисунок 3.24 – Запит на додавання користувача

Адміністратор має виконати цю операцію (зв’язатись з технічною підтримкою або зробити це власноруч) і відмітити її виконаною. Аналогічна ситуація зі звітами про помилки. В звітах не обов’язково вказувати свої дані, тоді в панелі це буде відмічатись як анонімний звіт.

Сам адміністратор відповідальний за обробку запитів, оскільки, це підвищує безпеку сайту. Заборона для користувачів виконувати адміністративні дії (створення груп, додавання студентів тощо) була зумовлена тим, що це підвищує ризики недобросовісного використання цих функцій. Наприклад, студент зможе використати DDOS-атаку при створені груп, або змінити собі роль і виставити оцінки. Тому, важливо підтримувати систему посередника, який буде або самостійно виконувати запити, або делегувати це технічній підтримці.

Створення облікових записів також відбувається адміністратором сайту для того, щоб користувачі, що не числяться в навчальному закладі, не могли отримати доступ до системи. Адміністратор повинен сам їх створювати і вказувати роль користувача, а також вказувати зону його відповідальності на основі дисциплін та груп. Відбувається це наступним чином – адміністратор на своїй панелі натискає кнопку «Зареєструвати користувача» і він перейде на наступну форму:

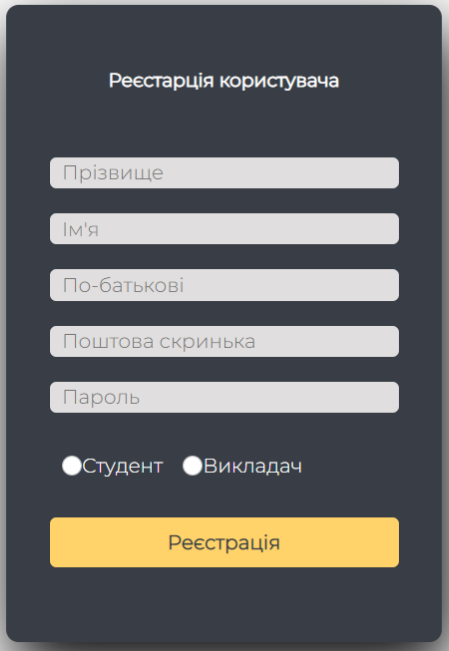

Рисунок 3.25 – Форма реєстрації користувача

В цій формі, адміністратор вказує ПІБ, поштову скриньку, пароль від облікового запису та роль користувача. Після реєстрації, сервіс реєстрації сповістить сервіс безпеки та сервіс для роботи з оцінками про те, що в системі є новий користувач. Але, він не матиме ні груп, ні дисциплін, і їх додавання потрібно виконати вручну (див. Запити користувачів).

Узагальнюючи, адміністратор виконує всю роботу по обліковим записам користувачів для того, щоб уникнути до них небажаного доступу до важливих елементів системи. Такий підхід дозволяє підвищити безпеку ПЗ і унеможливити вказування невірної інформації, яка може спричинити помилки в системі.

ВИСНОВОК

У бакалаврській роботі виконано поставлені задачі, що відповідають меті роботи і спроектовано та створено систему моніторінгу успішності здобувачів вищої освіти.

Для вирішення задачі було розглянуто та проаналізовано наявні системи, що дозволяють виконувати задачі моніторінгу та коригування оцінок студентів, переглянули їх відмінності та переваги і недоліки, врахували їх і уникнули при створенні власної системи.

Було розглянуто перелік інструментів, що використовувались для вирішення задачі і обґрунтували доцільність їх використанні у цьому випадку.

Реалізовано функціонал як для перегляду оцінок зі сторони студента, так і їх додавання та коригування зі сторони викладача. Також додано можливість зворотного зв’язку для оперативного вирішення проблем. В майбутньому, є можливість покращувати продукт і додавати новий функціонал, оскільки обрана архітектура це дозволяє. Прикладом може бути експортування оцінок, а також їх імпортування, повідомлення про зміни в оцінках та ін.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. JDK 11 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/en/java/javase/11
2. Comparing Embedded Servlet Containers in Spring Boot [Електронний ресурс] - https://www.baeldung.com/spring-boot-servlet-containers
3. Which versions of Java do you regularly use? [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/lp/devecosystem-2019/java/#:~:text=Although%20Java%2010%20and%2011,still%20the%20most%20used%20version
4. PresentationDomainDataLayering [Електронний ресурс] – Рижим доступу до ресурсу: https://martinfowler.com/bliki/PresentationDomainDataLayering.html
5. Pattern: Microservice Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://microservices.io/patterns/microservices.html
6. Building REST services with Spring [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/guides/tutorials/rest/
7. Spring Data JPA - Reference Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/
8. Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-security/reference/index.html
9. Spring Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-cloud
10. Spring Cloud OpenFeign [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-cloud-openfeign/docs/current/reference/html/

11. Java EE: Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=applications-java-ee-overview
12. Maven Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/guides/index.html
13. Introduction to the Angular Docs [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs
14. Angular Developer Guides [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/developer-guide-overview
15. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HTML
16. CSS [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/CSS
17. Bootstrap Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/4.1/getting-started/introduction/
18. PostgreSQL vs. MySQL: What You Need to Know [Електронний ресурс] – Режим доступу до ресурсу: https://www.fivetran.com/blog/postgresql-vs-mysql#:~:text=PostgreSQL%20is%20an%20object%2Drelational,%2C%20ACID%2Dcompliant%20storage%20engine.
19. PostgreSQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/
20. Docker Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/

ВІДГУК

на кваліфікаційну (бакалаврську) роботу Сороки Олександра Сергійовича з теми: «Розробка веб-додатку моніторінгу результатів навчання здобувачів вищої освіти»

Оцінка якості виконання кожного розділу та кваліфікаційної (бакалаврської) роботи

_____

Оцінка праці здобувача та результатів кваліфікаційної (бакалаврської) роботи

_____

Висновок про наявність плагіату_____

Висновок щодо можливості допуску кваліфікаційної (бакалаврської) роботи до захисту

_____

Кваліфікаційна (бакалаврська) робота Сороки Олександра Сергійовича «Розробка веб-додатку моніторінгу результатів навчання здобувачів вищої освіти» відповідає / не відповідає вимогам, які висуваються до кваліфікаційних (бакалаврських) робіт, і може бути рекомендована / не рекомендована до захисту.

Керівник: _____

РЕЦЕНЗІЯ

на кваліфікаційну (бакалаврську) роботу здобувача вищої освіти

Сороки Олександра Сергійовича з теми:

«Розробка веб-додатку моніторінгу результатів навчання здобувачів вищої освіти»

Кваліфікаційна (бакалаврська) робота Сороки О.С. «Розробка веб-додатку моніторінгу результатів навчання здобувачів вищої освіти» відповідає вимогам, які висуваються до кваліфікаційних робіт та може бути рекомендована до захисту. Кваліфікаційна (бакалаврська) робота заслуговує оцінки «_____», а її автор – присудження ступеня вищої освіти «Бакалавр».

Рецензент: _____

(підпис)

_____

Прізвище, ім’я, по-батькові

_____

Факультет

_____

Шифр і назва спеціальності

_____

Освітня програма

ДЕКЛАРАЦІЯ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (бакалаврська) робота на тему: «_____
_____»
є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;

- не порушує авторських та суміжних вправ, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;

- не отримувались іншими особами, а також дані та інформація не отримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна (бакалаврська) робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____

дата підпис здобувача