МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МАЗУРУК ОЛЕГ ВОЛОДИМИРОВИЧ

Допускається до захисту:
Завідувач кафедри інформаційних технологій,
кандидат технічних наук
_____ Т.В. Нескородева
«__» _____ 20__р.

РОЗРОБКА АВТОМАТИЗОВАНОЇ ПІДСИСТЕМИ ВИБОРУ НАУКОВОГО КЕРІВНИКА

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій,
к. фіз.-мат. н., доцент

Оцінка _/___/_____
(бали за шкалою ЄКТС/ за національною шкалою)
Голова ЕК:_____
(підпис)

Вінниця 2021

Мазурук О.В. Розробка автоматизованої підсистеми вибору наукового керівника. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У курсовій роботі проаналізовано та розроблено монолітну та мікросервісну архітектури для додатку, а також схеми баз даних.

Проведено аналіз і короткий опис інструментів які дозволяють вирішити поставлену задачу.

Розроблено Web-API та користувацький інтерфейс автоматизованої підсистеми обрання наукового керівника.

Ключові слова: автоматизована підсистема, WEB-API, мікросервісна архітектура, монолітна архітектура, RESTful, Java, Gradle, Spring Framework, Vaadin, Docker, Git, PostgreSQL.

Рис. 32, Табл. 14, Бібіограф.: 41 найм.

Mazuruk O.V. Development of an automated subsystem for the selection of a scientific director. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2021.

Monolithic and microservice architectures for the application, as well as database schemas, have been developed.

The analysis and the short description of tools that allow solving the set task are carried out.

Developed Web-API and user interface of the automated subsystem for the selection of the supervisor.

Keywords: automated subsystem, WEB-API, microservice architecture, monolithic architecture, RESTful, Java, Gradle, Spring Framework, Vaadin, Docker, Git, PostgreSQL.

Fig. 32,Table. 14, Bibliography.: 41 items.

ЗМІСТ

ВСТУП

Актуальність роботи: Щорічно у всіх закладах вищої освіти відбувається розподіл студентів до наукових керівників. При цьому існує ряд факторів, які не враховуються при розподілі, серед найважливіших можна виділити відсутність врахування пріоритетності студентів та викладачів, а також сфери майбутньої спеціалізації.

Процес розподілу є систематичним і не вимагає нововведень. Тобто цю сферу діяльності можна автоматизувати. Автоматизація дозволить налагодити процес та підвищити продуктивність.

Мета дослідження: Розробити автоматизовану підсистему вибору наукового керівника.

Завдання дослідження:

- аналіз предметної області;
- аналіз переваг та недоліків аналогів;
- розробити автоматизовану підсистему вибору наукового керівника на основі отриманих результатів.

Об’єкт дослідження: процес автоматизованого розподілу, інтеграція систем.

Предмет дослідження: застосування Web-технологій для створення автоматизованої підсистеми обрання наукового керівника.

Практичне значення одержаних результатів: розгортання підсистеми в якості незалежного додатку, інтеграція із наявними додатками, взаємодія із додатками для вирішення проблеми стабільних шлюбів по REST-API.

Зв’язок роботи з науковими програмами, планами, темами. Наведені у бакалаврські роботі дослідження пов’язані з фундаментальною науково-дослідною роботою «Дослідження та комп’ютерно-математичне моделювання складних систем та процесів у науці, освіті та інформаційно-комунікаційній діяльності підприємств» (№ держреєстрації 0116U002394, 2018-2022 рр.).

Апробація результатів дослідження: доповідь із публікацією тез на тему «Проєктування мікросервісної архітектури для підсистеми вибору наукового

керівника» у II Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Прикладні інформаційні технології»(ПІТ-2021).

Структура кваліфікаційної роботи: Бакалаврська робота складається із вступу, трьох розділів та висновків до них, списку використаних джерел та одного додатку.

У першому розділі бакалаврської роботи проведено огляд предметної області, та інструментів що дозволяють вирішити поставлену задачу.

У другому розділі наведено модель автоматизованої підсистеми обрання наукового керівника.

У третьому розділі наведено опис користувацького інтерфейсу, а також особливості налаштування технологій.

Бакалаврська робота включає в себе 61 сторінку, 32 рисунки і список літератури із 41 джерела.

РОЗДІЛ 1

АНАЛІЗ АНАЛОГІВ ПІДСИСТЕМИ ВИБОРУ НАУКОВОГО КЕРІВНИКА ТА АКТУАЛЬНИХ ЗАСОБІВ РОЗРОБКИ

1.1 Вибір наукового керівника

Кожний університет щорічно надає студентові наукового керівника, для підвищення якості написання наукових праць. Це дуже зручно, оскільки при виникненні запитань, можна проконсультуватись і все коректно описати.

Але проблема виникає при самому розподілі, тому що він відбувається у випадковому порядку і не враховуються пріоритети викладачів і студентів. Тому в даний час для університетів є актуальною розробка підсистеми, одним із головних завдань якої є розподіл студентів і наукових керівників із урахуванням їх пріоритетів.

Ідеальними критеріями для розподілу студентів і наукових керівників є врахування пріоритетів кожного учасника навчального процесу, прозорість та ефективність. Також необхідно врахувати сферу спеціалізації наукового керівника та студента.

Враховуючи те, що процес розподілу відбувається щорічно – доцільно реалізувати автоматизовану підсистему, яка буде задовольняти усім вищеперерахованих критеріям.

1.2 Автоматизована підсистема та приклад її використання

Для автоматизації рутинних задач по розподілу студентів та наукових керівників необхідно виділити основні кроки цього процесу:

1. Формування списків студентів, для яких буде здійснюватися пошук наукового керівника.
2. Формування списків викладачів, для яких буде здійснюватися підбір студентів.
3. Здійснення розподілу;
4. Оприлюднення результатів розподілу.

Аналізуючи приведений вище алгоритм розподілу, можна помітити що не враховується велика кількість важливих факторів, таких як:

- Навчальне навантаження викладача;
- Спеціалізація викладача(веб-додатки, desktop-додатки, мікроконтролери і т.д.);
- Бажана спеціалізація студента;
- Пріоритети розподілу викладачів та студентів.

Створення автоматизованої підсистеми направлено на вирішення поточних недоліків.

1.3 Сервіси-аналоги

Головним критерієм при пошуку аналогів вважається автоматична система розподілу із врахуванням пріоритетів користувачів.

Знайдені системи не є загальнодоступними. В більшості це закриті системи, до яких доступ отримати неможливо. Проте, можна проаналізувати можливості цих систем на основі їх комерційних пропозицій.

Наприклад, платформа пацієнт-лікар [1] – основна ідея проєкту полягає у вирішенні проблеми взаємодії лікаря та пацієнта. Через відсутність простого у користування медичного програмного забезпечення, комунікація стає проблематичною ланкою взаємодії. Для цього було відкрито портал, який здійснює автоматизацію повсякденної роботи лікарів.

Групою дослідників було встановлено, що на вирішення адміністративних питань медичні працівники витрачають до 50% робочого часу [2]. Тому було створено автоматизовану систему для автоматизації поточного процесу:

В першу чергу користувачам необхідно вказати що саме їх турбує, для цього вони заповнюють анкету та за допомогою інтуїтивно-зрозумілого інтерфейсу(рис. 1.1) та вказують причину дискомфорту.

При створенні заявки також вказуються відгуки про попередні відвідини лікарів, якщо такі були. Після цього відбудеться автоматичний розподіл(рис.

1.2), враховуючи зайнятість лікарів, спеціалізацію та причину дискомфорту користувача.

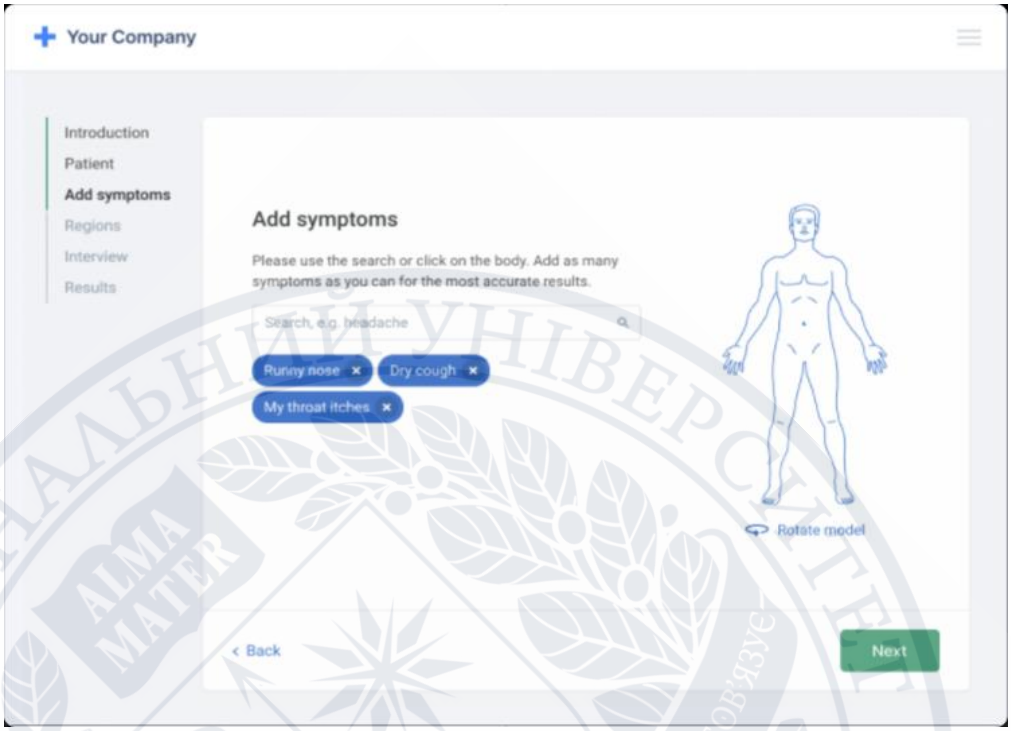

Рисунок 1.1 – користувацький інтерфейс для визначення симптомів пацієнта [1]

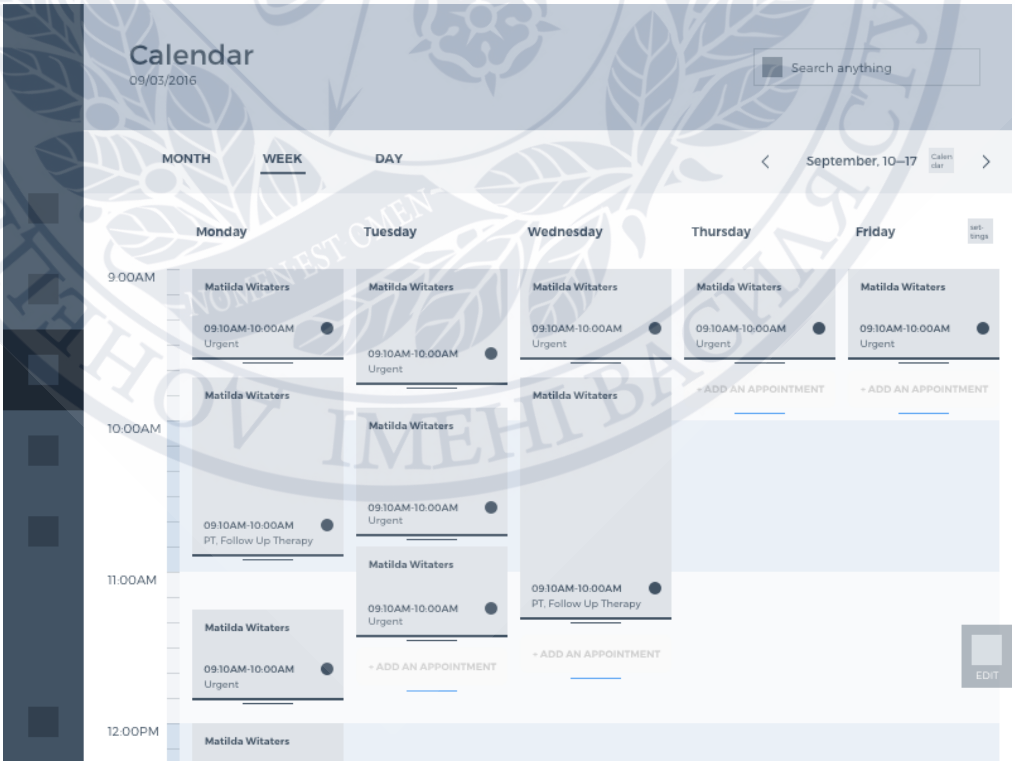

Рисунок 1.2 – результат автоматичного розподілу пацієнтів до лікарів [1]

Наступним аналогом можна вважати сервіс «Автоматизованого зарахування студентів до закладів вищої освіти України» [3]. Принцип роботи полягає у автоматичному розподіленні студентів на основі ЗНО до закладів вищої освіти.

На першому етапі абітурієнти складають здійснюють реєстрацію у електронному кабінеті. Після цього відбувається складання ЗНО та подача заяв. Заявку на вступ формують абітурієнти, вказуючи бажаний заклад вищої освіти та його пріоритетність.

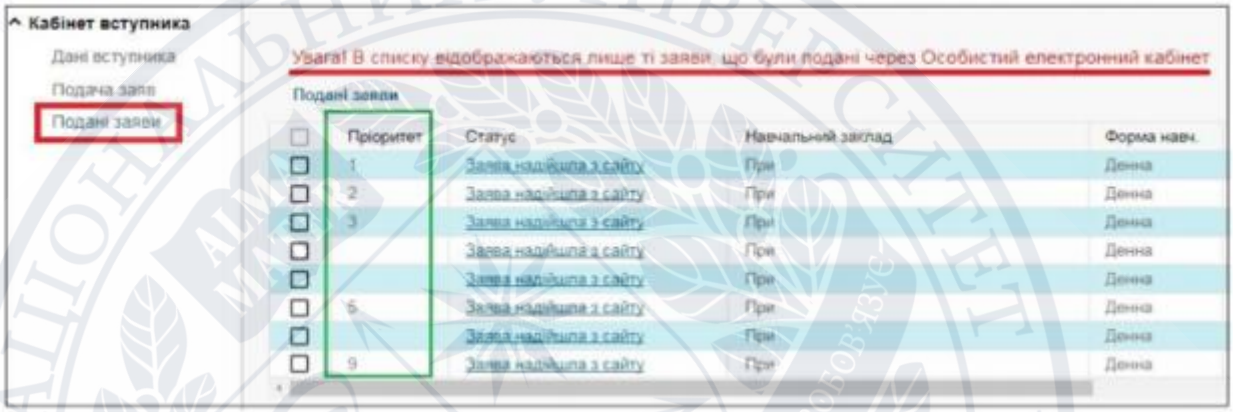

Рисунок 1.1 – ілюстрація вказаних пріоритетів у особистому кабінеті абітурієнта [3]

Після завершення строку подачі заяв, система автоматично створює пріоритети вищих навчальних закладів, для цього обираються абітурієнти із найвищими балами, кількість яких становить максимально можливому набору. Далі здійснюється автоматичний розподіл в результаті якого буде отримано списки абітурієнтів рекомендованих до зарахування:

#	ПІБ	П	Σ	С	ДЕТАЛІЗАЦІЯ	К	Д
1	246-504536 В. Ю.	1	143.310	До наказу (б)	• Українська мова та література: **140** • Математика: **143** • Фізика: **135** • Середній бал документа про освіту: **6** • РК	**Квота №2**	+
2	Харченко Е. К.	1	125.970	Рекомендовано (б)	• Українська мова та література (ЗНО): **113** • Математика: **122** • Іноземна мова (ЗНО): **129** • Середній бал документа про освіту: **6.1** • РК	**Квота №1**	—
3	Капля Г. О.	2	191.352	Рекомендовано (б)	• Українська мова та література (ЗНО): **181** • Математика (ЗНО): **195** • Фізика (ЗНО): **180** • Середній бал документа про освіту: **9.9** • РК	—	—
4	Федоренко Є. О.	1	189.822	Рекомендовано (б)	• Українська мова та література (ЗНО): **174** • Математика (ЗНО): **194** • Фізика (ЗНО): **186** • Середній бал документа про освіту: **9.1** • РК	—	—

Рисунок 1.2 - результат автоматичного розподілу абітурієнтів до ЗВО [4]

1.4 Огляд мов та інструментів, що використовують у Web-розробці

Оскільки передбачається створення автоматизованої підсистеми на основі клієнт-серверної архітектури, тому буде розглянуто інструменти які спрощують створення web-додатків.

Найпопулярнішими мовами програмування для створення web-додатків є Java, Go, JavaScript, Php.

Java – це строго-типізована, об’єктно-орієнтована та компільована мова програмування, випущена 1995 року компанією «Sun Microsystems». Для програмування на Java необхідна віртуальна машина, яку можна завантажити на офіційному сайті компанії «Oracle» [5]. Java є кросплатформеною, тому підтримує безліч різних операційних систем і СКДБ, на відміну від ASP.NET, який найбільш повно взаємодіє лише з продуктами Microsoft. Створенні Web-API на Java, на відміну від платформи NodeJS, має ряд переваг, таких як: багатопоточність, краща продуктивність і менша ресурсоємність, а також дуже висока стабільність системи. Java допомагає вирішувати завдання розробникам серверних, клієнтських та вбудованих систем. На Java написано безліч готових бібліотек і корпоративних вирішень, які спрощують написання коду, його підтримку, а також удосконалення. Це означає, що при створенні свого Web-API можна користуватися вже існуючими напрацюваннями, а не писати все з нуля. До найпопулярніших інструментів розробки належать Spring Framework, JHipster, Struts, Vaadin та Thymeleaf.

Spring – це Java-фреймворк який реалізує інверсію управління(IoC) та ін’єкцію залежності(DI) [6]. Spring забезпечує легке створення корпоративного програмного забезпечення на Java. Також реалізована підтримка Groovy та Kotlin як альтернативні мови на JVM. Використовуючи Spring, можлива реалізація багатьох видів архітектур залежно від потреб програми. Починаючи з Spring Framework 5.1, Spring вимагає JDK 8+ (Java SE 8+) і надає повну підтримку для JDK 11 LTS.

Spring Framework розділений на модулі. В основі знаходяться модулі основного контейнера, які також включають конфігураційну модель та механізм

введення залежностей. Також, Spring Framework надає повну підтримку різним типам архітектур програмного забезпечення, включаючи можливість обміну повідомленнями, транзакційні дані, persistence та веб. Він також включає веб-фреймворк Spring MVC на основі технології Servlet і паралельно реактивний веб-фреймворк WebFlux.

Spring з'явився у 2003 році як противага складним специфікаціям J2EE. Інколи вважають Java EE та Spring конкурентами, проте Spring доповнює можливості Java EE. Модель програмування Spring не охоплює специфікації платформи Java EE; а здійснює інтеграцію з наступними технічними характеристиками EE: Сервлети (JSR 340); веб-сокети (JSR 356); багатопоточність (JSR 236); валідація JSON (JSR 367); Bean валідація (JSR 303); Persistence (JSR 338); Message Service (JSR 914); Transaction / Cryptography.

Spring Framework здійснює підтримку специфікації ін'єкції залежностей (JSR 330) та підтримку анотацій (JSR 250). Тобто, розробникам надається можливість замінити стандартні механізми Spring на власні.

Spring Framework починаючи з версії 5.0 має наступні мінімальні вимоги: Java EE 7; Сервлети версії 3.1+; Persistence 2.1+. При цьому здійснюється збереження підтримки нових можливостей Java EE. Завдяки цьому Spring забезпечує повну сумісність із контейнерами сервлетів(Tomcat, WebSphere та JBoss).

JHipster – це технологія для швидкої розробки та розгортання веб-додатків. Підтримує створення на основі стеку технологій: Spring Framework + React(Angular,Vue), а також безлічі інших технологій [7]. JHipster забезпечує спрощене розгортанням у Docker та Kubernetes. Також надає налаштування для таких технологій як Hibernate, Thymeleaf, Maven, Gradle, Micronaut, OOS Netflix, MySQL, PostgreSQL, ELK та інших.

Тобто, JHipster не є окремим фреймворком, а лише агрегує в собі всі самі популярні варіанти поєднання інших фреймворків. Ця взаємодія забезпечується завдяки готовим модулям. До головних переваг варто віднести автоматичну

генерацію та розгортання, готові варіанти архітектур додатку, використовуються тільки сучасні фреймворки що активно розвиваються.

Серед недоліків варто зазначити генерацію великої кількості зайвого коду, файлів, що спричиняє великий розмір додатку. Також, фреймворки що підтримуються JHipster не постачаються «із коробки», для їх використання необхідно здійснити на машину де здійснюється розробка додатку.

Struts – це фреймворк, що створений на основі Servlet технології для розробки веб-додатків на основі MVC-архітектури [8]. Був створений Крейгом МакКланаханом у 2000 році. Фреймворк є частиною Apache Software Foundation. На сьогодні підхід і методологія розробки на Struts є застарілими.

Vaadin – це платформа для розробки користувацького односторінкового додатку(SPA), використовуючи лише Java та xml [9]. Функціонування додатку відбувається за server-side підходом, тобто вся бізнес-логіка знаходить на сервері, а front-end лише відображає дані для користувача. Vaadin забезпечує взаємодію між клієнтом та сервером та рендеринг веб-сторінок. Цей підхід має ряд переваг та недоліків. До основних переваг відносять те, що непотрібно вивчати HTML/CSS/JavaScript, забезпечення стабільного рівня безпеки додатку, писати та тестувати код простіше. До недоліків відносять надмірну кількість пам’яті, що використовує додаток, важкість при масштабуванні, а також складність розробки при необхідності персональних компонентів.

JavaScript(JS) – це інтерпретована об’єктно-орієнтована мова програмування [10]. На JS можна створити як серверну і користувацьку частину додатку. Програми які були написані на JavaScript мають назву – скрипт. Також для виконання JavaScript програм необхідно мати спеціальний «JS рушій», оскільки мова не є компільованою. JavaScript дозволяє без перезавантаження сторінки змінювати її стиль та вміст, реагувати на дії користувачів(натискання на кнопку чи переміщення вказівника), здійснювати запити на сервери для отримання даних, здійснювати маніпуляції із cookie. Для створення серверної частини використовують Node.js, а для користувацької – React, Angular або ж Vue.

Node.js – це платформа, що дозволяє створювати мережеві застосунки, транслюючи JavaScript код у машинний [11]. Головне призначення Node.js полягає у створенні серверних додатків на мові JavaScript. Node.js використовує однопоточну, неблокуючу модель вводу/виводу. Це забезпечується за рахунок асинхронних функцій. Використовуючи лише один потік, додаток створений на Node.js споживає менше системних ресурсів. Проте, відсутність багатопоточності є недостатком, оскільки це корисно із довгими або циклічними операціями. Також до мінусів варто віднести відсутність перевірки на строгу типізацію даних, оскільки це може призводити до виникнення помилок під час виконання додатку, а також те, що платформа не підходить для складних та розподілених обчислень.

React – це бібліотека, що дозволяє створювати користувацький інтерфейс використовуючи JavaScript [12]. React було створено спеціально для створення великих веб-додатків, де дані можуть змінюватись дуже часто. До переваг варто віднести відносну простоту вивчення, високий рівень гнучкості, створення віртуальної об’єктної моделі документу(Document object model, DOM), забезпечує високу відмовостійкість, стабільність розвитку бібліотека та простота при переході на нові версії. Серед недоліків виділяють інколи недостатню або застарілу документацію, для більш глибокого розуміння роботи – необхідно витратити багато часу.

VueJS – це JavaScript фреймворк, який був розроблений для створення адаптованих користувацьких інтерфейсів і складних SPA [13]. До переваг VueJS відносять оптимізовану обробку HTML-блоків за рахунок компонентів, відносна простота інтеграції у інші системи, детальна та актуальна документація, має схожий дизайн із React, легко здійснити масштабування додатку, висока швидкість роботи. Серед недоліків виділяють відсутність однозначного підходу до розробки додатку, невелику поширеність фреймворку, що спричиняє меншу затребуваність спеціалістів і як наслідок розвиток цієї технології відбувається не так інтенсивно.

Go – це процедурна мова програмування, яка поширюється із відкритим вихідним кодом [14]. Go має велику кількість переваг у порівнянні із іншими мовами. До переваг можна віднести: лаконічність і простота програмного коду, ефективна підтримка багатопоточності, висока швидкість роботи та компіляція, ефективна реалізація збирання сміття(garbage collection), а найважливішим аспектом є те, що для написання повноцінного додатку все доступно «із коробки». До недоліків відносять відсутність підтримки generics-типів, відсутність повної підтримки об’єктно-орієнтовної підтримки. Також, Go більше націлений на реалізацію серверних додатків, тому існує мала кількість інструментів, які дозволяють спростити створення користувацького інтерфейсу.

Php – це скриптова мова програмування, основною задачею якої була генерація статичного контенту на стороні серверу [15]. Як і Go, Php постачається із відкритим вихідним кодом. До головних переваг Php варто віднести низький поріг входження при вивченні мови, швидке вирішення простих задач. До негативних сторін відносять слабку систему безпеки, надійність, передбачуваність та цілісність даних якими оперує додаток. Серед найпопулярніших фреймворків можна виділити Laravel, CodeIgniter.

1.5 Висновки до розділу 1

У даному розділі було розглянуто постановку задачі бакалаврської роботи, проведений огляд існуючих аналогів та інструментів, за допомогою яких можливо створити підсистему. Наступний розділ буде присвячений формуванню загальних вимог до підсистеми та бази даних, а також проєктуванню архітектури додатку.

РОЗДІЛ 2
РОЗРОБКА КОМП'ЮТЕРНО МАТЕМАТИЧНОЇ МОДЕЛІ АВТОМАТИЗОВАНОЇ ПІДСИСТЕМИ ВИБОРУ НАУКОВОГО КЕРІВНИКА

2.1 Загальні вимоги до підсистеми

Необхідно розробити Web-API та UI для автоматизованої підсистеми вибору наукового керівника.

Загальні вимоги до підсистеми є наступними:

- наявність авторизації та автентифікації;
- можливість створення, проходження та видалення користувачів;
- можливість створення, проходження та видалення опитувань;
- наявність анкети для користувачів;
- зручний користувацький інтерфейс;
- відносна простота інтеграції у інші системи.

2.1.1 Вимоги до бази даних

Необхідно створити базу даних, яка буде відповідати наступним вимогам:

- забезпечити зберігання опитувань. Сутність «Опитування» містить наступні поля: дата створення та завершення, група для якої створено опитування, викладачі що можуть бути науковими керівниками та кількість вільних місць, ким було створене опитування, tenantId;
- забезпечити зберігання користувачів. Сутність «Користувач» містить наступні поля: прізвище, ім’я, по-батькові, e-mail, пароль, чи є акаунт активним, анкетні дані, перелік груп доступу, перелік ролей, tenantId;
- забезпечити зберігання ролей користувачів. Сутність «Роль користувача» містить наступні поля: назва ролі, короткий опис, перелік можливостей, tenantId;
- забезпечити зберігання груп доступу. Сутність «Група доступу» містить наступні поля: назва групи доступу, перелік можливостей, tenantId;

- забезпечити зберігання інформації про ЗВО. Сутність «ЗВО» містить наступні поля: повна назва ЗВО, коротка назва ЗВО, адреса, tenantId;
- забезпечити зберігання інформації про факультети. Сутність «Факультет» містить наступні поля: повна назва факультету, коротка назва факультету, ЗВО до якого належить, перелік студентів, перелік викладачів, tenantId;
- забезпечити зберігання інформації про групи. Сутність «Група» містить наступні поля: повна назва групи, скорочена назва групи, дата початку навчання, дата випуску, перелік студентів, tenantId;
- забезпечити підтримки мультитенантності. Сутність «Tenant» містить наступні поля: tenantId, назва орендаря.
-

2.1.2 Вимоги до веб додатку

Система повинна задовольняти наступним вимогам:

- можливість інтеграції у системи закладів вищої освіти;
- можливість працювати як незалежний сервіс;
- можливість імпорту користувачів використовуючи excel файл;
- можливість експорту даних у excel файл;
- авторизація і автентифікація на основі протоколу OAuth2.0;
- розмежування користувачів на основі груп доступу та ролей;
- підтримку трьох типів користувачів із ролями admin, moderator, teacher, student;
- ведення журналу метрик серверу;
- здійснювати автоматичну розсилку e-mail повідомлень про початок, завершення та результат опитування;
- можливість підтримки мультитенантності;
- можливість визначити поставника авторизації;
- можливість налаштувати SMTP сервер без перезапуску додатку.

Також має бути включений розподіл прав доступу згідно ролей користувачів, який наведено у таблиці 2.1.

Таблиця 2.1 – розподіл прав доступу користувачів згідно ролей.

Можливості	Guest	Student	Teacher	Moderator	Admin
Проходження опитувань	—	+	+	+	+
Заповнення анкети	—	+	+	+	+
Створення/редагування /видалення опитувань	—	—	—	+	+
Створення/редагування /видалення нових користувачів	—	—	—	+	+
Створення/редагування /видалення нових груп	—	—	—	+	+
Створення/редагування /видалення нових факультетів	—	—	—	+	+
Створення/редагування /видалення нових університетів	—	—	—	—	+
Доступ до панелі адміністрування додатком	—	—	—	—	+

2.1 Постановка задачі розподілу у вигляді математичної моделі

При аналізі задачі автоматизованого розподілу, можна виділити дві множини які взаємодіють між собою – це студенти та викладачі. Нехай $C = \{c_1, c_2, \ldots, c_n\}$ – множина студентів. Тоді $L = \{l_1, l_2, \ldots, l_p\}$ – множина викладачів. Оскільки кількість місць у викладача є обмеженою – необхідно це врахувати. Для кожного викладача l_i буде встановлено значення s_i ($i = 1, \ldots, p$), яке вказуватиме на максимальну кількість вільних місць у *i*-го викладача.

Кожний студент(c_j) встановлює пріоритети для викладача формуючи підмножину $Lj \subseteq L$. Також, кожний викладач(l_i) виставляє пріоритети для студентів. В результаті отримаємо підмножину студентів $C_i \subseteq C$. Елементи отриманих підмножин будемо вважати впорядкованими згідно пріоритетності, де перший елемент вважається найбільш-пріоритетним.

Після отриманих підмножин пріоритетів відбувається розподіл. В контексті цієї задачі вважається, що розподіл – це бінарне відношення $P \subseteq L \times C$, яке має задовольняти наступні обмеження:

$$P(s_j) \subseteq L_j \ (j = 1, \dots, n) \tag{2.1}$$

але $P(s_j) \leq 1$. Дана умова вказує на те, що кожний студент може мати наукового керівника і лише одного.

$$P(l_i) \subseteq C_i \ (i = 1, \dots, p) \tag{2.2}$$

але $P(l_i) \leq q_i$ Дана умова вказує, що кожний викладач може бути науковим керівником для обмеженої кількості студентів.

Сам процес розподілу не є стійким. Оскільки завжди є можливість перейти до більш пріоритетного викладача. Аналогічно і для викладача – завжди є можливість вибрати більш пріоритетного студента. Будемо вважати розподіл стійким та оптимальним, якщо $(l_i, c_j) \in P$ та кожний студент отримує найкращого із доступних для нього викладачів.

2.2 Алгоритм Гайла-Шеплі

Розподіл відбується на основі алгоритму Гайла-Шеплі або більш відомому як «Проблема стабільних шлюбів (англ. Stable Matching Problem або SMP)» [16]. Проблема стабільних шлюбів – це формування стабільних пар між двома множинами згідно пріоритетів кожного елемента множини. Парою вважається поєднання елементу першої множини із елементом другої множини.

Формулювання SMP відбувається наступним чином: існує рівна кількість(n) чоловіків та жінок. Кожен виставляє унікальні пріоритети(від 1 до n) стосовно осіб протилежної статі. Якщо сформована пара задовольняє наступній умові: немає людей протилежної статі, які б підходили краще один одному ніж поточні

партнери, враховуючи протилежні пари, то таке одруження називають стабільним.

Поточний алгоритм використовується в сервісі «Автоматизованого зарахування студентів до закладів вищої освіти України». Для наочності, розглянемо блок-схему алгоритму:

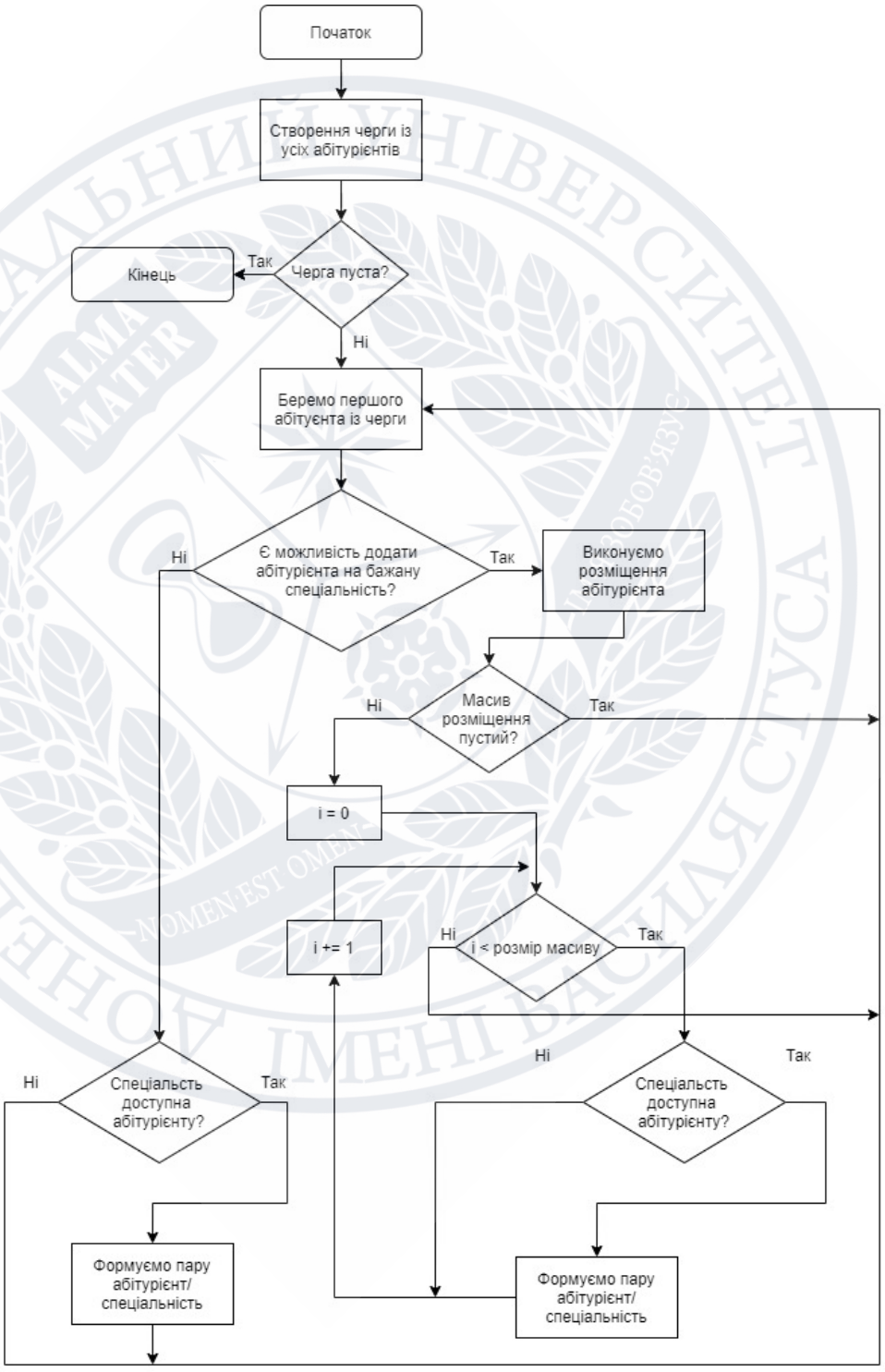

Рисунок 2.1 – блок-схема алгоритму автоматичного розподілу абітурієнтів до ЗВО [17]

2.4 Розробка моделі веб додатку

Архітектура програмного забезпечення – це розподілення системи на компоненти, а також взаємодією компонентів між собою та оточенням(згідно із стандартом «IEEE 42010:2011») [18].

Проєктування архітектури програмного забезпечення – один із найважливіших етапів розробки, оскільки він забезпечує майбутні характеристики системи, а саме:

- відмовостійкість – забезпечується безперервність функціонування, незалежно від кількості і коректності вхідних даних;
- масштабованість – система відкрита для вертикального та горизонтального масштабування;
- гнучкість – додання нових можливостей системи вимагає мінімум зусиль;
- тестування – забезпечується відносна ізольованість компонентів, яка спрощує написання автоматичних тестів для системи;
- повторне використання – архітектура забезпечує можливість повторного використання фрагментів [19].

Оскільки розробка архітектури програмного забезпечення є типовою задачею, над якою працювала велика кількість розробників, було виділено ефективні способи вирішення або ж шаблони [20]. До основних архітектурних шаблонів належать:

- Модель-вид-контролер(MVC) – передбачає наявність у системі три основних шари:
    1) Модель даних(Model) – забезпечує збереження, редагування, оновлення та видалення даних;
    2) Вигляд(View) – інтерфейс користувача;
    3) Контролер(Controller) – забезпечує керування та взаємодію між моделлю даних та вигядом [21].
- Клієнт-серверна архітектура – забезпечує створення розподіленого мережевого програмного забезпечення. Для коректного функціонування передбачається наявність наступних компонентів:

1) Сервери – забезпечують надання та обробку інформації;
2) Клієнти – використовують сервіси надані серверами;
3) Мережа – забезпечує взаємодію між серверами та клієнтами [22].

- Триярусна архітектура – забезпечує розбиття системи на три шари:
  1) Клієнт – компонент для взаємодії користувача із системою, зазвичай графічний;
  2) Сервер застосунків – отримує і обробляє дані, забезпечуючи функціонування усієї бізнес-логіки додатку;
  3) Сервер бази даних – забезпечує зберігання, отримання, видалення та оновлення інформації [23].
- Багатошарова архітектура – передбачається розбиття додатку на шари, які заточені під виконання певної бізнес-логіки [24].
- Сервісно-орієнтована архітектура – передбачає розбиття програмного забезпечення на компоненти [25]. Кожний компонент виконує певну задачу і слабко пов'язаний із іншими.
- Мікросервісна архітектура – передбачається розбиття програмного забезпечення на безліч невеликих та незалежних додатків, кожен з яких заточений під свою бізнес-задачу, а взаємодія між сервісами забезпечується за допомогою швидких протоколів передачі даних [26].
- Монолітна архітектура – передбачається, що програмне забезпечення буде працювати як одна система.
- REST – використовується для побудови веб-служб [27]. Вимагається клієнт-серверна архітектура додатку, сервер не повинен зберігає інформацію про поточну сесію клієнта, у відповіді сервер зазначає чи є необхідність кешування, визначається єдиний інтерфейс взаємодії між клієнтом і сервером.
- Головний-підлеглі – забезпечує підвищену відмовостійкість. Передбачається наявність двох сторін:
  1) Головний або master – основний компонент, що розподіляє задачі між однаковими підлеглими компонентами, а також обробляє результат;

2) Підлеглі або salves – додатки, що виконують задачу поставлену головним.

Оскільки підсистема створюється для подальшої інтеграції у інші система, тому обрано REST архітектуру додатку. Для внутрішньої організації взаємодії компонентів буде застосовано багатошарову архітектуру. Для моніторингу та розгортання системи, потрібно визначити переваги та недоліки мікросервісної та монолітної архітектури.

2.3.1 Мікросервісна архітектура

В першу чергу необхідно визначитись із переліком бізнес-задач, які необхідно реалізувати. До бізнес-задач можна віднести: авторизація, автентифікація, CRUD-операції над опитуваннями, а також формування пар.

Далі необхідно здійснити декомпозицію даних задач на декілька основних мікросервісів. Кожний мікросервіс буде відповідати за виконання певної бізнес-задачі(рис. 2.3).

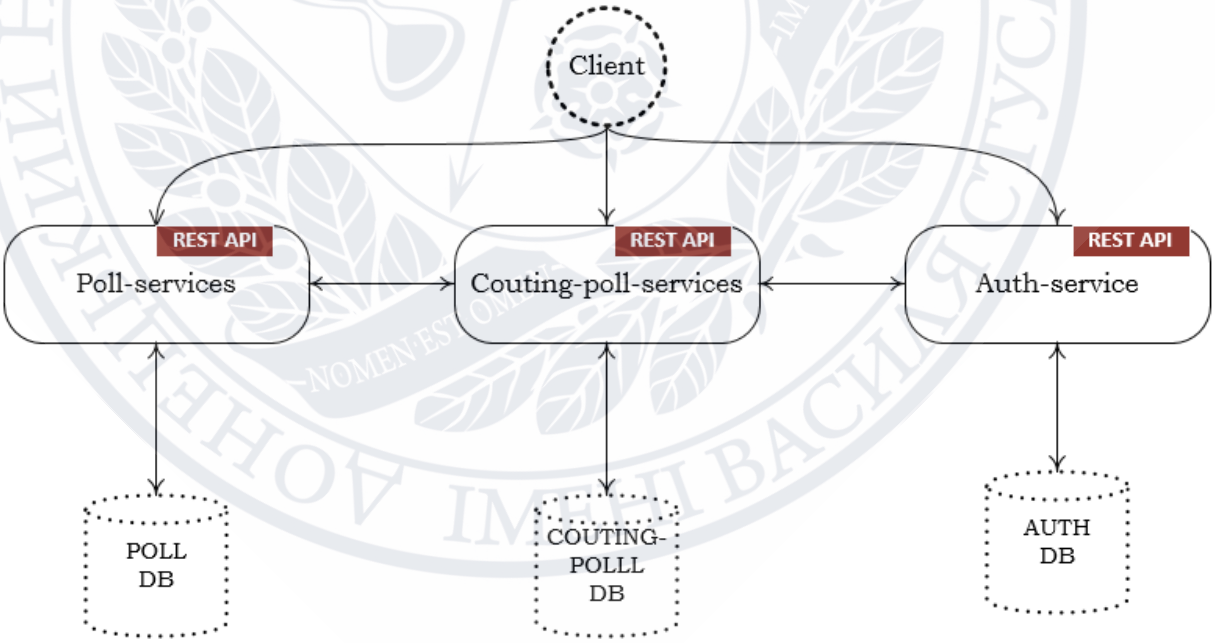

Рисунок 2.3- розподілення бізнес-задач серед сервісів

Було виділено три основних мікросервіса(рис. 2.3):

а) Обов'язки по авторизації повністю винесені в окремий сервіс. Auth-services використовується як для авторизації користувачів, так і для захищеного спілкування між сервісами.

b) Обов’язки по виконанню CRUD операцій над опитуваннями було винесено у Poll-services. Щоб отримати доступ до даного сервісу, необхідно отримати токен доступу у Auth-service.

c) Обов’язки по формуванню пар(науковий керівник - студент), було винесено у сервіс Couting-Poll-Services. При настанні завершення опитування, дані будуть надсилатись у поточний сервіс, де буде сформовано результат. Однією із переваг є те, що Web-API даного сервісу можуть використовувати інші розробники [28].

Для забезпечення спільної роботи описаних вище сервісів, буде використано набір основних патернів і практик мікросервісної архітектури(рис. 2.4). Розглянемо детальніше кожен компонент:

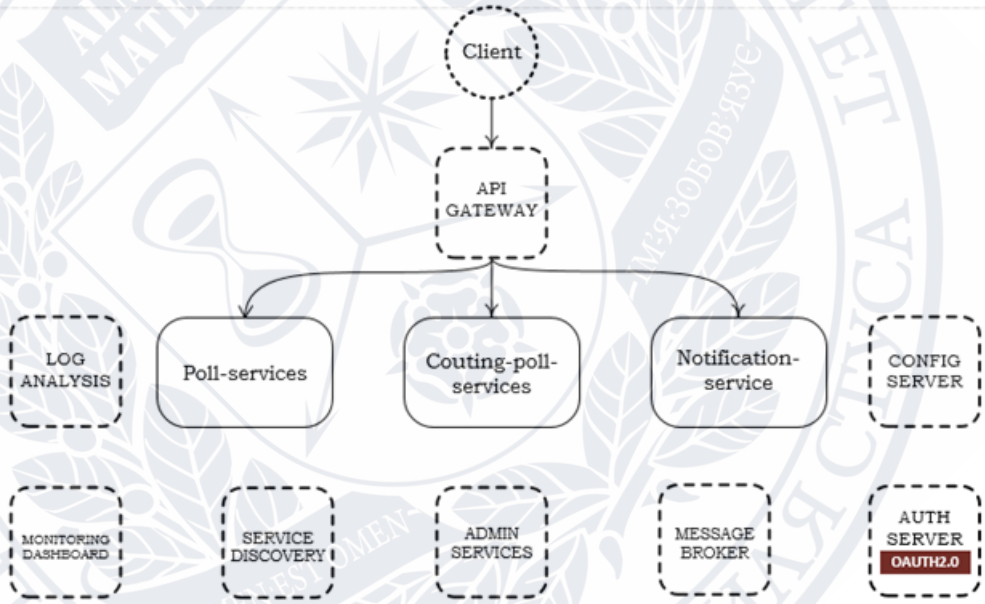

Рисунок 2.4- інфраструктурний розподіл монолітної архітектури

- Config server - це сховище конфігурацій для розподіленої системи, яке підтримує горизонтальне масштабування. Як джерело даних конфігураційних файлів можуть виступати системи Git, Subversion або ж прості файли, що зберігаються локально. На практиці найбільший інтерес представляє завантаження конфігурацій із систем котроля версій, але для простоти можна використовувати локальні файли.

- API Getawey – це єдина точка входу у веб-сервіс. Клієнтський додаток міг би здійснювати запит до кожного мікросервісу самостійно. Але при такому підході виникає маса обмежень - необхідність знати адресу кожного мікросервісу, робити запит за кожним шматком інформації окремо і самостійно агрегувати результат. Для вирішення такого роду проблем застосовують єдину точку входу. Забезпечує прийом користувацьких запитів і здійснює маршрутизацію у потрібні інфраструктурні сервіси, також забезпечує автентифікацію, тестування, віддачу статичного контенту, та динамічне управління трафіком.
- Service discovery - дозволяє автоматично визначати мережеві адреси для доступних додатків, які можуть динамічно змінюватися з причин масштабування, падінь та оновлень.
- Monitoring dashboard – сервіс, що забезпечує агрегацію метрик усіх сервісів.
- Message broker – це брокер повідомлень, який буде виконувати одразу декілька задач. Перша – це доставка запитів і відповідей між сервісами, де стабільність є критично важливою. Це реалізується за допомогою створення трьох черг, перша відповідає за доставку до підписника, інша за відповідь від підписника, третя зберігає повідомлення, які з певних причин не були оброблені і здійснює повторну відправку. Друга задача – це забезпечення збору і передачі метрик, для їх подальшого оброблення і візуалізації.
- Admin service – це користувальницький інтерфейс, який дозволяє переглядати всі дані моніторингу та здійснювати управління сервісами.

2.3.2 Монолітна архітектура

Для монолітної архітектури перелік бізнес-задач буде аналогічний із монолітною архітектурою. Тому, після модульної декомпозиції бізнес-задач було отримано архітектуру, що відображена на рисунку 2.5.

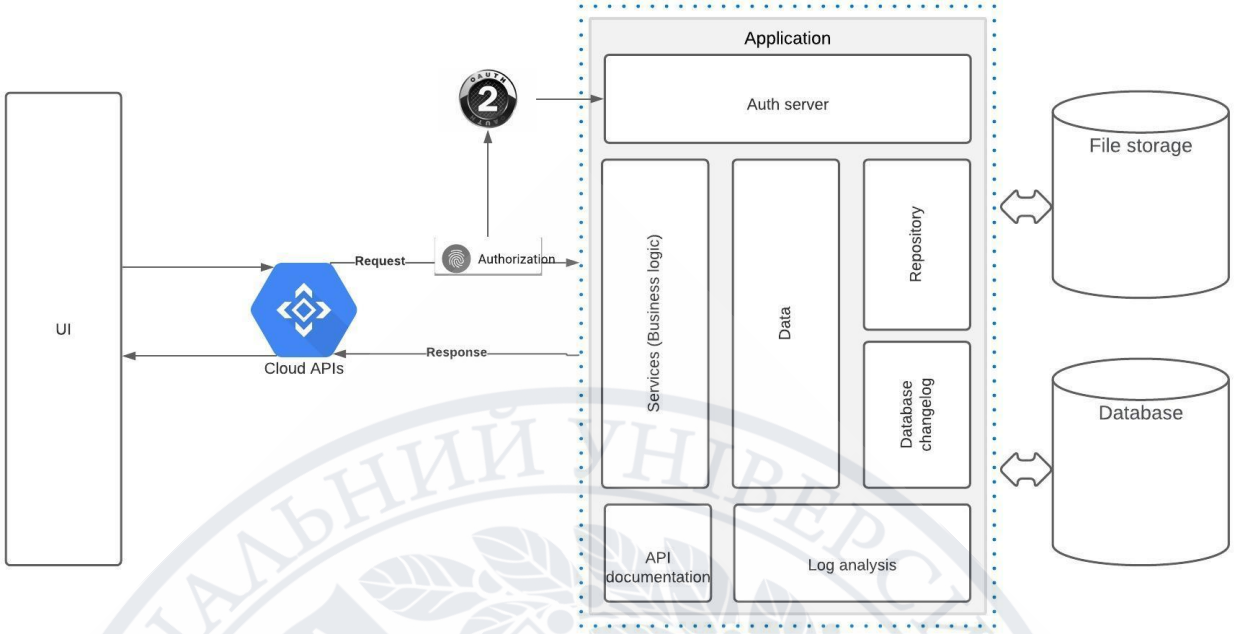

Рисунок 2.5- монолітна архітектура додатку

Розглянемо детальніше кожен компонент:

- Auth-server – модуль, що відповідає за авторизацію та автентифікацію користувача;
- Log analysis – це компонент, що відповідає за збір метрик;
- Repository – забезпечує взаємодію із базою даних;
- Data – забезпечує відображення даних у вигляді сутностей;
- Services(Business logic) – акумулює бізнес-логіку підсистеми;
- Auth server – сервер автентифікації та авторизації;
- API documentation – вичерпна документація по прикладному програмному інтерфейсу;
- UI – користувацький інтерфейс.

2.3.3 Порівняння монолітної та мікросервісної архітектури

Для зручності порівняння монолітної та мікросервісної архітектур було складено таблицю із наступними критеріями:

Таблиця 2.6 – Порівняння монолітної та мікросервісної архітектур

Критерій	Моноліт	Мікросервіси
Висока відмовостійкість	—	+

Продовження таблиці 2.6

Мережеві затримки	—	+
Складність операційної підтримки	—	+
Складність інтеграційного тестування	—	+
Швидке розгортання на сервері	—	+
Простота масштабування	—	+
Єдиний стек технологій	+	—

Враховуючи усі переваги та недоліки, варто використовувати монолітну архітектуру, оскільки це спрощує розгортання, взаємодію компонентів, на створення мінімальної версії продукту буде витрачено менше часу та ресурсів.

2.5 Архітектура розгортання додатку

Розгортання додатку є важливим процесом при розробці. Для здійснення розгортання необхідно виконати наступні операції: запустити автоматичне тестування додатку, якщо всі тести були задовільними – виконати додання нового коду до репозиторію. Після цього виконати build додатку. В результаті буде отримано додаток, що готовий до завантаження на сервер.

При запуску готового додатку на сервері, можна скористатись декількома варіантами: одразу запустити додаток, створити віртуальну машину із додатком або здійснити контейнеризацію. Найбільш оптимальним є варіант створення контейнеру із додатком. В результаті контейнеризації буде отримано нове зображення(docker image), яке необхідно помістити у реєстр.

Маючи нове зображення у реєстрі – можна здійснити створення контейнеру та його запуск сервері.

З огляду на процес розгортання, можна стверджувати що він є достатньо рутинною операцією, яка займає достатньо багато часу у розробників. Для оптимізації цього процесу необхідно здійснити автоматизацію за рахунок методу безперервної інтеграція та розгортання(CI/CD) [29].

Безперервна інтеграція(CI) – це процес внесення змін до кодової бази із подальшим ручним тестуванням.

Безперервна доставка(CD) – це процес автоматичного тестування та додання протестованого коду до репозиторію.

Постійне розгортання – це процес автоматичного оновлення додатку, який було отримано із CD.

Метод CI/CD дозволяє швидше здійснювати розгортання додатку, що дає можливість пришвидшити введення нових можливостей системи, а також виправляти старі недоліки.

Після впровадження методу CI/CD, процес розгортання буде виглядати наступним чином:

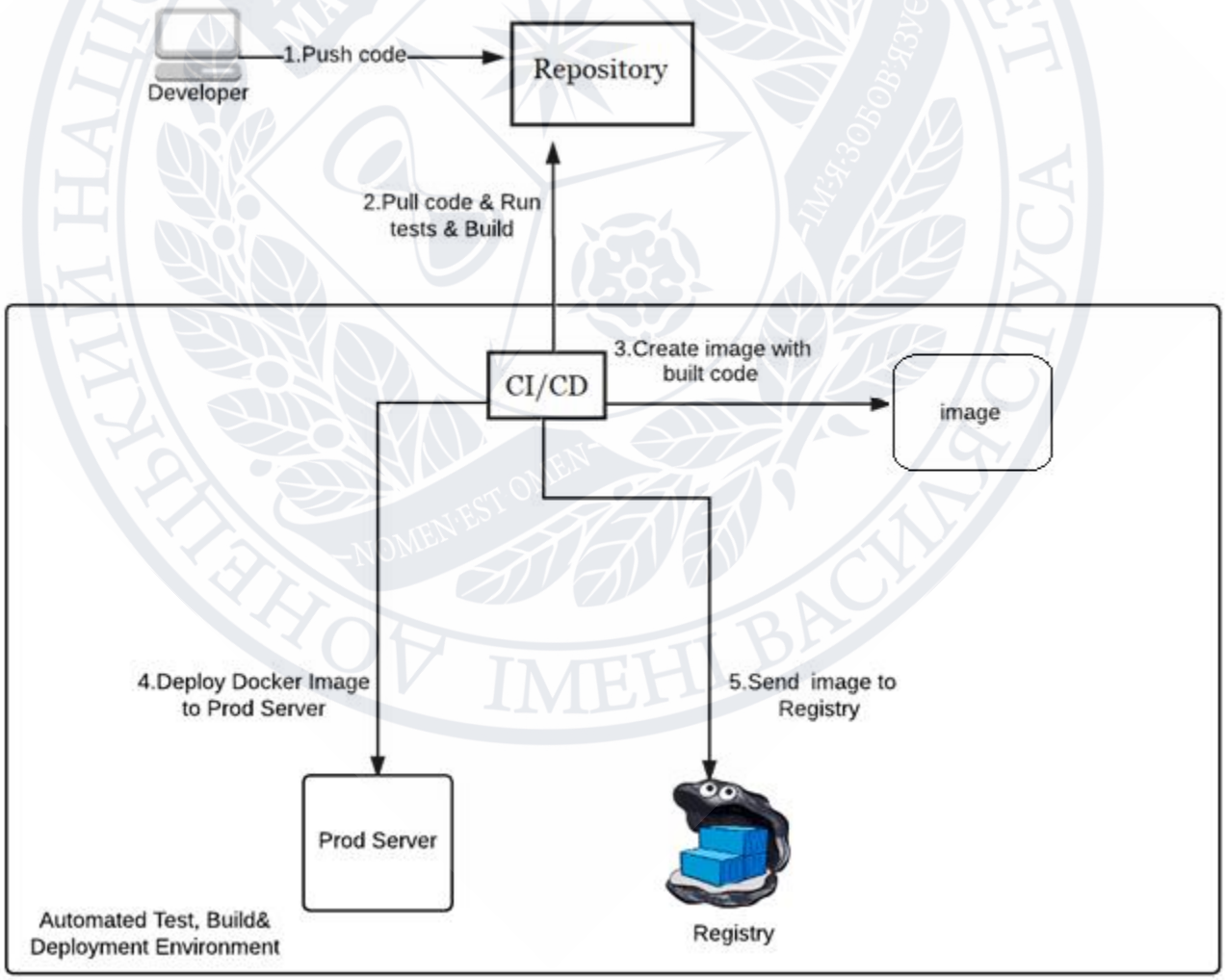

Рисунок 2.6 – процес безперевної інтеграції та розгортання [30]

Розглянемо детальніше кожний крок зображений на рисунку 2.6:

1. Розробник вносить зміни у код проєкту. Після чого здійснює фіксацію змін за допомогою системи контролю версій.
2. Система контроля версій здійснює зворотній виклик до технології, що реалізує CI/CD механізм. Внаслідок чого CI/CD технологія здійснює отримання актуальної кодової бази. Після отримання коду відбувається запуск автоматичного тестування та збірка додатку, якщо усі тести були вдалими.
3. Коли було отримано зібраний додаток, відбувається формування нового зображення.
4. Сформоване зображення поміщається у реєстр.
5. Після виконання усіх попередніх кроків, відбудеться створення контейнеру на основі останнього актуального зображення із подальшим оновленням на сервері, де розміщено додаток.

2.6 Висновок до розділу 2

У даному розділі було сформовано загальні вимоги до бази даних, архітектури додатку і процесу розгортання. У наступному розділі будуть розглядатись особливості процесу розробки і роботи із інтерфейсом підсистеми.

РОЗДІЛ 3
РОЗРОБКА ДІЮЧОГО ПРОТОТИПУ АВТОМАТИЗОВАНОЇ ПІДСИСТЕМИ ВИБОРУ НАУКОВОГО КЕРІВНИКА

3.1 Вибір інструментів для розробки

3.1.1 СУБД PostgreSQL

PostgreSQL – це об'єктно-реляційна база даних з відкритим вихідним кодом. Для запитів використовується розширена версія мови SQL. PostgreSQL було розроблено у 1986 році на основі проєкту Postgres [31].

У PostgreSQL реалізовано безліч функціональних можливостей, які допомагають розробникам створювати додатки, забезпечувати цілісність даних, мати високу відмовостійкість, а також здійснювати основні операції над даними. Також PostgreSQL є дуже гнучким: можна вказати власні типи даних, додати користувацькі функції, здійснювати запити на основі процедурних мов.

Архітектуру системи PostgreSQL визначають як клієнт-серверну і скаладається з наступних взаємодіючих процесів:

- Сервер – виконує управління файлами бази даних, керує з'єднаннями до бази даних із зовнішніх додатків, а також здійснює маніпуляції із даними від імені зовнішніх додатків. Якщо сервер встановлено на машину, то викликати його можна наступною командою у терміналі: postgres.
- Клієнт – зовнішній додаток, що намагається виконати маніпуляції із базою даних. Зовнішніми додатками можуть виступати графічні додатки, веб-додатки або спеціалізовані інструменти для адміністрування бази даних.

Взаємодія сервера і клієнта здійснюється через мережеве з'єднання TCP/IP. Також, PostgreSQL може одночасно працювати із декількома з'єднаннями. При спробі отримання нового з'єднання клієнтом – сервер ініціалізує новий процес або потік.

PostgreSQL забезпечує підтримку використання з'єднання SSL, що забезпечує шифрований зв'язок між клієнтом та сервером для безпечної взаємодії. Для цього необхідно ввімкнути дане налаштування у конфігураціях

серверу та перезавантажити його. Окрім SLL доступні наступні методи автентифікації:

- аутентифікація паролем – вимагає від користувача пароль при підключенні;
- GSSAPI – створений на основі сумісності із бібліотекою безпеки GSSAPI. Зазвичай використовується для серверів Kerberos та Microsoft Active Directory;
- Ідентифікація індикатора – заснована на протоколі RFC 1413;
- LDAP – делегує автентифікацію LDAP серверу;
- SSL – перевірка надісланого клієнтського SSL сертифікату;
- BSD – забезпечується структурою автентифікації BSD [31].

Для здійсненні віддаленого підключення, було налаштовано використання лише SSL-автентифікації, оскільки це є досить захищене з’єднанням.

Також існує безліч налаштувань серверу, які можна змінити у файлі конфігурацій або за допомогою терміналу. Основні налаштування, що були використані для налаштування PostgreSQL:

- shared_buffers – виконує встановлення максимального споживання оперативної пам’яті машини, за замовчуванням складає 128 мб;
- temp_buffers – встановлення максимального розміру пам’яті, яка використовується для зберігання тимчасового буферу у рамках окремого сеансу;
- max_prepared_transactions – встановлення максимальної кількості транзакцій, що одночасно перебувають у підготовленому стані;
- work_mem – встановлення максимального обсягу пам’яті, що використовується під час надходження запиту від клієнта, перед записом даних у тимчасові буфери;
- temp_file_limit – встановлення максимального дискового обсягу для тимчасових файлів згенерованих при запиті клієнта. Якщо транзакція намагається перевищити обмеження – відбудеться її ануляція;

- max_files_per_process – встановлення кожному підпроцесу сервера максимальної кількості одночасно відкритих файлів;
- effective_io_concurrency – встановлення обмеження на кількість одночасних операцій вводу-виводу, які виконуються асинхронно;
- old_snapshot_threshold – встановлення мінімальної кількості часу для того, щоб вважати поточний snapshot актуальним [32].

PostgreSQL підтримує роботу декількох серверів одночасно. Це забезпечує підвищену відмовостійкість. Якщо один сервер вийшов з ладу, інший делегує на себе усі запити. Коли усі сервери працюють коректно – відбувається балансування навантаження. Якщо сервер баз даних призначений тільки для читання, використати архітектурний шаблон головний-підлеглі(master-slaves) доволі просто. Але більшість баз даних реалізовують операції читання/запису, тому такі сервери набагато важче поєднати, оскільки виникає проблема синхронізації нових даних між серверами.

У PostgreSQL реалізовано декілька способів резервного копіювання:

- дамп SQL – забезпечується утилітою pg_dump. Відбувається генерація файлу із SQL командами, які містять створення таблиць та дані відповідних таблиць. Дампи, які було створено pg_dump є узгодженими, тобто вони представляють знімок бази даних у момент запуску утиліти. Також утиліта не блокує операції інших користувачів під час виконання.
- копіювання на рівні файлової системи – необхідно виконати безпосереднє копіювання файлів, які PostgreSQL використовує для зберігання даних. При цьому способі необхідно вимкнути сервер баз даних. Недоліком цього способу є те що неможливо відновити дані окремої таблиці чи таблиць, а тільки кластера повністю.
- безперервне архівування – необхідно налаштувати резервне копіювання на рівні файлової системи, а також виконати резервне копіювання файлів журналу реєстру змін файлів бази даних. Однак, цей спосіб також надає можливість відновлення даних лише кластером [33].

З огляду на простоту використання, для здійснення резервного копіювання було використано утиліту pg_dump.

3.1.2 Фреймворк Spring

Spring достатньо потужний інструмент для створення серверних додатків. Для користувацьких налаштувань фреймворку, необхідно розглянути фундаментальні процеси, на яких заснована вся екосистема Spring.

Spring Core постачається із реалізацією інверсії контролю та ін'єкцією залежностей. За допомогою цього процесу можна визначити компоненти та їх залежності без написання багатьох рядків зайвого коду, також це спрощує супроводження додатку і внесення нових змін, оскільки достатньо перевизначити компонент в місці оголошення.

За ініціалізацію, збір та налаштування Bean-класів відповідає інтерфейс ApplicationContext, який представлений декількома реалізаціями, вибір яких покладається на розробника [34]. Ініціалізація IoC контейнера відбувається наступним чином:

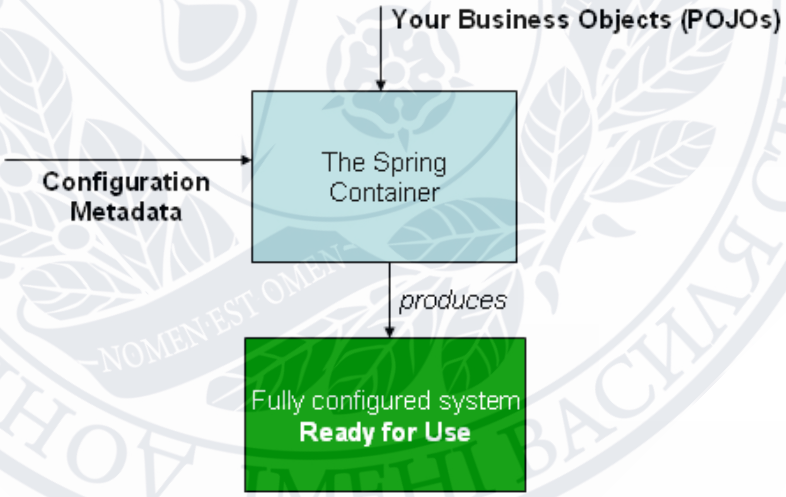

Рисунок 3.1 ініціалізація IoC контейнера[34]

Спочатку класи поєднується із конфігураційними метаданими, після чого ApplicationContext здійснює обробку та запуск додатку. Конфігураційні метадані вказує розробник за допомогою xml файлу або анотацій.

Контейнер виконує управління одним або декількома bean екземплярами. Кожний екземпляр може містити наступні дані:

- Назва класу – за замовчуванням вказується фактичний клас реалізації компоненту;
- Елементи конфігурації поведінки компоненту – визначають поведінку у контейнері, а саме область дії, життєвий цикл;
- Посилання на інші bean класи, що необхідні для належного функціонування поточного класу;
- Користувацькі параметри конфігурації класу.

При розробці було оголошено bean класи для налаштування авторизації та автентифікації, бізнес-логіки, взаємодії із базою даних, створення запланованих завдань та кінцевих точок додатку(endpoint).

Spring MVC – це фреймворк що базується на основі Spring Core та технології Servlet, здійснює забезпечення отримання, обробки та відправлення запитів [35].

Беручи за основу Java EE, Spring ініціалізує головний Servlet, який має назву DispatcherServlet. Імплементації класу конфігуруються за допомогою xml файлу, анотацій або поєднання цих способів.

Для більшості додатків наявність єдиного WebApplicationContext є простим і достатнім. Також можливо мати контекстну ієрархію, де WebApplicationContext спільно використовується в декількох DispatcherServlet екземплярах, кожен із яких має власну дочірню WebApplicationContext конфігурацію [36].

Кореневий WebApplicationContext має містити компоненти інфраструктури, такі як сховища даних та бізнес-служби, які потрібно використовувати в декількох Servlet екземплярах. Ці bean класи фактично успадковуються і можуть бути перевизначені (тобто повторно оголошені) у дочірньому сервлеті WebApplicationContext, який зазвичай містить локальні bean класи для даного Servlet. Рисунок 3.2 ілюструє це співвідношення.

DispatcherServlet делегує спеціальним bean класам функції обробки запитів і створення відповідей, до них належать наступні:

- HandlerMapping – зв’язує запит із обробником для попередньої та подальшої обробки;

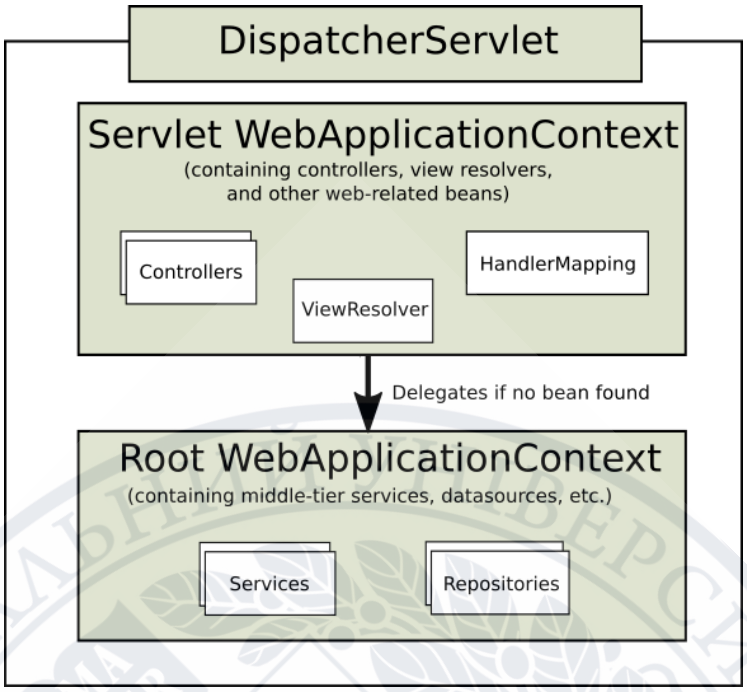

Рисунок 3.2 - інфраструктура WebApplicationContext [34]

- HandlerAdapter – допомагає DispatcherServlet викликати потрібний обробник для запиту, обробник вказується за допомогою анотації;
- HandlerExceptionResolver – використовується для вирішення помилок, які виникли в ході обробки запиту;
- LocaleResolver – підтримка інтернаціоналізації;
- ThemeResolver – здійснює підтримку тем для користувачів;
- MultipartResolver – здійснює підтримку обробки запиту, який містить multipart вміст;
- FlashMapManager – передача атрибутів із одного запиту іншому.

DispatcherServlet здійснює обробку запитів наступним чином :

- WebApplicationContext здійснює пошук і прив’язку запиту із контролером, в якому буде здійснюватися подальша обробка та формуватися відповідь. За замовчуванням взаємопов’язаний із DispatcherServlet;
- Розпізнання локалізації прив’язаної до запиту, щоб встановити локалізовану обробку запиту. Якщо система не використовує локалізацію - поточний крок ігнорується.

- Здійснюється розпізнання теми, встановлена користувачем. Розпорядник тем прив’язаний до запиту. Якщо теми не використовуються – поточний крок ігнорується.
- При надходженні запиту, який має декілька складових частин(наприклад: файл та JSON дані), запит загортається в MultipartHttpServletRequest для подальшої обробки іншими елементами.
- Здійснюється пошук відповідного обробника. Якщо обробник знайдено, запускається обробка запиту, щоб підготувати відповідь.
- Якщо обробка запиту виконується без помилок – сервер здійснює відправку відповіді на запит. Якщо виникає помилка, подальша обробка здійснюється у HandlerExceptionResolver.

При розробці було використано обробку помилок за допомогою HandlerExceptionResolver. Також оголошено декілька bean класів для обробки запитів.

Spring MVC дозволяє виступати в якості клієнта. Для доступу до інших кінцевих точок можна скористатись двома способами [37]:

- RestTemplate – синхронний клієнт для виконання HTTP запитів. Це REST-клієнт, який надає простий API.
- WebClient – неблокуючий реактивний клієнт для виконання HTTP запитів. Підтримується Spring починаючи із версії 5.0. Має суттєві переваги над RestTemplate, а саме: асинхронна обробка, використання меншої кількості апаратних ресурсів; функціональне програмування.

Додаток може виступати в якості клієнта, якщо необхідно автоматизувати процес наповнення бази даних. Для забезпечення цієї функції було створено кінцеві точки, які приймають Groovy-скрипти в тілі запиту, після чого відбувається

Окрім стандартних HTTP запитів , Spring MVC підтримує з’єднання на основі веб-сокетів. Веб-сокети були використані для взаємодії додатку до СУБД, а ще взаємодії користувацького інтерфейсу та додатку.

Spring Security – це фреймворк що базується на основі Spring Core та забезпечує захист системи, авторизацію та автентифікацію. Spring Security постачається із великою кількістю готових рішень, які необхідно лише правильно конфігурувати.

Найбільш захищеним способом зберігання паролів є їх перетворення через адаптивні односторонні функції. Spring Security містить інтерфейс PasswordEncoder, який є головним поставником адаптивних односторонніх функцій. Реалізація інтерфейсу PasswordEncoder залежить від потреб розробника. Spring Security пропонує декілька реалізацій: Bcrypt , PBKDF2, Scrypt і Argon 2.

Для реалізації безпечного зберігання паролів було використано Bcrypt, оскільки він генерує досить стійкий до зламів паролів хеш при цьому навмисно працюючи повільно. Також було встановлено параметру strength значення 15, що забезпечує приблизний час перевірки пароля близько 1 секунди.

Spring Security здійснює захист системи від CSRF атак, надаючи два механізми: Token Pattern та налаштування атрибуту SameSite у файлах cookie.

У програмній реалізації було налаштовано захист системи за допомогою Token Pattern. Цей процес включає в себе генерацію і надсилання користувачеві токен. Під час наступного запиту у http заголовку має міститись згенерований токен. Після цього здійснюється порівняння токену що надійшов із очікуваним токеном. У разі розбіжностей – запит буде вважатись атакою і буде повернуто код з помилкою.

Також Spring Security забезпечує підтримку єдиного входу на основі Oauth2.0 або ж OpenID. Ця підтримка значно спрощує інтеграцію додатку у інші системи.

3.1.3 Docker та Docker-compose

Docker – це інструмент із відкритим вихідним кодом, що спрощує створення, розгортання та запуск додатків за рахунок контейнеризації [38]. Механізм контейнеризації включає процес додання усіх необхідних залежностей для повноцінного функціонування додатку. Перевагою такого підходу є

можливість запуску контейнера на будь-якій машині де встановлено Linux, при цьому налаштування контейнера не використовують загальних налаштувань машини. За відносно короткий проміжок часу Docker став промисловим стандартом, що значно спрощує його використання.

Docker забезпечує ізольованість контейнерів, тобто маючи два різних контейнери, які знаходяться у різних мережах, отримати доступ із першого контейнера у другий стає неможливим. Ізольованими також є і шари контейнера.

Для спрощення розгортання декількох контейнерів було використано інструмент Docker Compose [39]. Він дозволяє визначити налаштування запуску контейнерів у YAML файлі [40]. Вказавши лише команду для запуску «docker-compose up» - відбудеться розгортання усіх сервісів, що вказані у файлі налаштувань.

У файлі налаштувань було визначено три сервіси:

- backend – забезпечує функціонування веб-додатку. При створенні контейнера відбувається додання “.jar” файлу, який містить зібраний веб-додаток.
- db – СУДБ PostgreSQL.
- pgadmin4 – інструмент для спрощеного адміністрування СУДБ PostgreSQL.

Для налаштувань сервісів були використані наступні параметри:

- environment – через змінні середовища забезпечується передача важливих даних, таких як логіни і паролі для взаємодії із СУБД.
- Image – зображення яке слід використати для створення контейнеру;
- Restart- налаштування політики перезапуску сервісу;
- Volumes – налаштування механізму збереження даних. При корекних налаштуваннях спрощується створення резервних копій та здійснення міграцій;
- Ports – здійснює прокидання загальнодоступного порту із локальної машини до контейнера;

- depends_on – визначає порядок включення сервісів. У конфігурації вказано, що сервіс із веб-додатком не зможе запуститись перш ніж відбудеться старт СУБД. PostgreSQL.

Після ініціалізації усіх сервісів відбувається створення локальної мережі. Для кожного контейнеру буде надана можливість доступу до інших контейнерів у поточній мережі. Хост, за яким контейнер буде доступний у мережі, визначається ідентичним іменем контейнеру.

Лістинг файлу налаштувань наведено у додатку А.

3.2 Розробка структури бази даних

Проєктування бази даних здійснено на основі реляційної моделі. Згідно до вимог, було виділено таблиці. В результаті було тримано наступну базу даних для автоматизованої системи вибору наукового керівника:

Схему бази даних було нормалізовано. Тобто схема відповідає наступним вимогам:

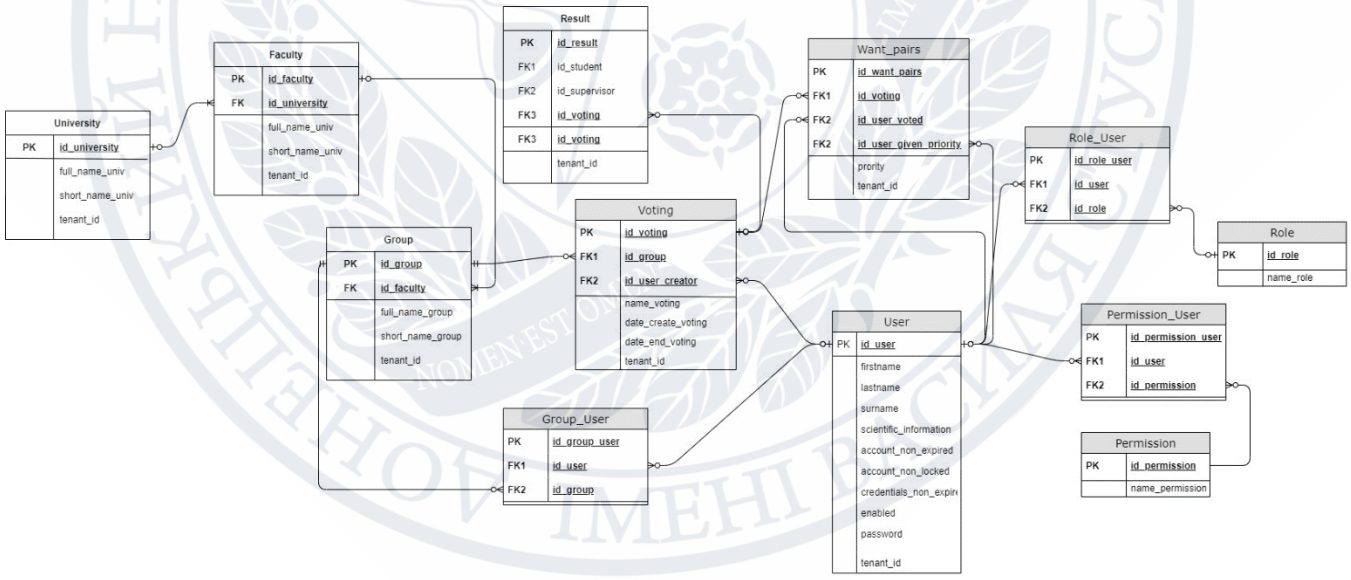

Рисунок 3.3 - схема бази даних

- усі атрибути є простими типами даних;
- записи залежать лише від основного ключа таблиці;
- забезпечується відсутність збитковості даних.

Розглянемо більш детальний опис ключових таблиць , що зображені на схемі. Таблиця 3.1 забезпечує зберігання інформації про навчальні групи.

Таблиця 3.1 – Groups

Назва	Тип	Опис
id_group	UUID	Унікальний ідентифікатор
id_faculty	UUID	Ідентифікатор факультету
full_name_group	Varchar(2048)	Повна назва групи
short_name_group	Varchar(512)	Коротка назва групи
start_study	Timestamp	Початок навчання
end_study	Timestamp	Завершення навчання
tenantId	UUID	Унікальний ідентифікатор орендаря

Зберігання усієї необхідної інформації про користувачів, а також інформацію про поточний стан аккаунту забезпечується таблицею таблицю 3.2.

Таблиця 3.2 – Users

Назва	Тип	Опис
id_user	UUID	Унікальний ідентифікатор
last_name	Varchar(512)	Прізвище
first_name	Varchar(512)	Імя
surname	Varchar(512)	По-батькові
email	Varchar(512)	Корпоративна або власна пошта
password	Varchar(512)	Хешований пароль користувача
enabled	Boolean	Чи активний аккаунт
account_non_expired	Boolean	Чи дійсні права доступу
account_non_locked	Boolean	Чи не заблокований аккаунт

Продовження таблиці 3.2

date_end_voting	Timestamp	Дата завершення опитування
tenantId	UUID	Унікальний ідентифікатор орендаря

Забезпечення нормалізації бази даних здійснюється за рахунок реляційних відношень «Один до одного», «Один до багатьох» і «Багато до багатьох». Таблиця 3.3 демонструє відношення «Багато студентів до багатьох груп». Це дає змогу забезпечити коректне і не збиткове зберігання даних, коли студент навчається у декількох групах одночасно.

Таблиця 3.3 – Users_Groups

Назва	Тип	Опис
id_group_user	UUID	Унікальний ідентифікатор
id_ group	UUID	Навчальна група
id_user	UUID	Користувач

Для забезпечення коректного зберігання , можливості створення відкладених опитувань і їх автоматичного завершення, було виділено структуру, описану у таблиці 3.4.

Таблиця 3.4 – Voting

Назва	Тип	Опис
id_voting	UUID	Унікальний ідентифікатор
id_group	UUID	Група для якої проводиться опитування
user_creator	UUID	Користувач, що створив опитування

Продовження таблиці 3.4

name_voting	Varchar(1024)	Назва опитування
date_start_voting	Timestamp	Дата початку опитування

Лістинг із описом усіх таблиць наведено у додатку Б.

3.3 Експорт та імпорт даних використовуючи excel файл

Microsoft Excel – це табличний процесор, який забезпечує роботу із електронними таблицямих [41] Є дуже зручним форматом для імпорту та експорту даних. Також, за рахунок його популярності, використання автоматичного заповнення даних на основі excel файлів, значно спрощує використання системи. Тому для автоматизованої підсистеми була розроблена можливість імпорту та експорту даних за рахунок excel файлів.

Для здійснення експорту, необхідно вибрати сутність та натисну кнопку:

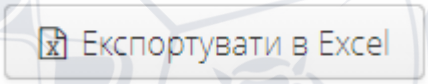

Рисунок 3.4 - кнопка для експорту у Excel

Після натискання, буде автоматично згенеровано та завантажено на локальну машину excel файл. Файл буде містити дані у відповідності до полей сутності, де в якості стовпців виступають назви полів.

Для імпорту даних із Excel файлу, необхідно вибрати сутність та натисну кнопку:

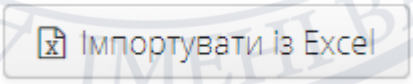

Рисунок 3.5 - кнопка для імпорту із Excel файлу

Після чого обрати необхідний файл на локальній машині та завантажити його. Файл повинен містити дані, які відповідають полям сутностей. Наприклад, для імпорту користувачів слід створити та заповнити файл згідно наведеної структури на рисунку 3.6. Я разі наявності некоректних даних буде відображено попереджувальне вікно із вказанням на помилку.

	A	B	C	D	E	F
1	**Login**	**First Name**	**Last Name**	**Middle Name**	**Email**	**Мобільний**
2	asfd	asdf	asdf	asdf	asfd@sdg.com	asdf
3	ggggg	asdf	sadfasdf	asdfasdf	ggggg@gggg	4654564654
4	mbeta77	asdfasdf	asfdasdfasdf	asdfasdfasdfas	mbeta77@gmail.com	3213412341234
5	mbeta55	fasdfasdf	sadfas	asdfasdf	mbeta55	234523452345
6	test33	dasd	asdas	asdasd	test33@gamil.com	234234234
7	adfawef	gdsf	sdf	gdfg	adfawef@gmail.com	4353453425
8	mbeta	sadf	asdf	asdf	mbeta@gamil.von	adsgagds
9	aaaaa	Юрій	Сергійович	Антонов	aaaaa@admin	wert
10	admin_donnu	admindonnu	admindonnu	admindonnu	admin@donnu.edu.ua	123123

Рисунок 3.6 - приклад структури Excel файлу для імпорту користувачів

3.4 Тестування роботи алгоритму Гайла-Шеплі

Для виконання аналізу даного алгоритму, була виконана програмна реалізація на мові програмування Java. Використовуючи генератор випадкових чисел(для ЗВО і абітурієнтів використано різні числові проміжки для наочності роботи) було створено вхідні дані відповідно до табл. 3.5, 3.6, 3.7

Таблиця 3.5 – пріоритетності, встановлені користувачами

ID абітурієнта	Пріоритетність		
	max		min
10	1	3	2
11	1	2	3
12	3	2	1
13	3	2	1
14	3	1	2

Таблиця 3.6 – пріоритетності, встановлені ЗВО

ID ЗВО	Пріоритетність				
	max				min
1	14	11	12	13	10
2	13	14	12	11	10
3	14	10	12	13	11

В результаті роботи алгоритму, було отримано результат(табл. 3.8), який вважається оптимальним розподілом.

Таблиця 3.7 – кількість студентів, що може прийняти ЗВО

ID ЗВО	Кількість вільних місць на спеціальності
1	1
2	2
3	2

Таблиця 3.8 – результат роботи алгоритму

ID ЗВО	ID абітурієнта
1	11
2	13, 12
3	14, 10

Також для перевірки визначеності алгоритму, було проведено повторні тести із однаковими вхідними даними(відповідно до таблиць 3.5, 3.6, 3.7). Після виконання 1000 тестів, було отримано однаковий результат, що відповідає таблиці 3.8.

Розглянемо тривалість роботи алгоритму у мілісекундах(вісь ординат) в залежності від кількості ЗВО та абітурієнтів(вісь абсцис):

На рисунку видно, що при збільшенні кількості ЗВО та абітурієнтів – середній час роботи алгоритму збільшується.

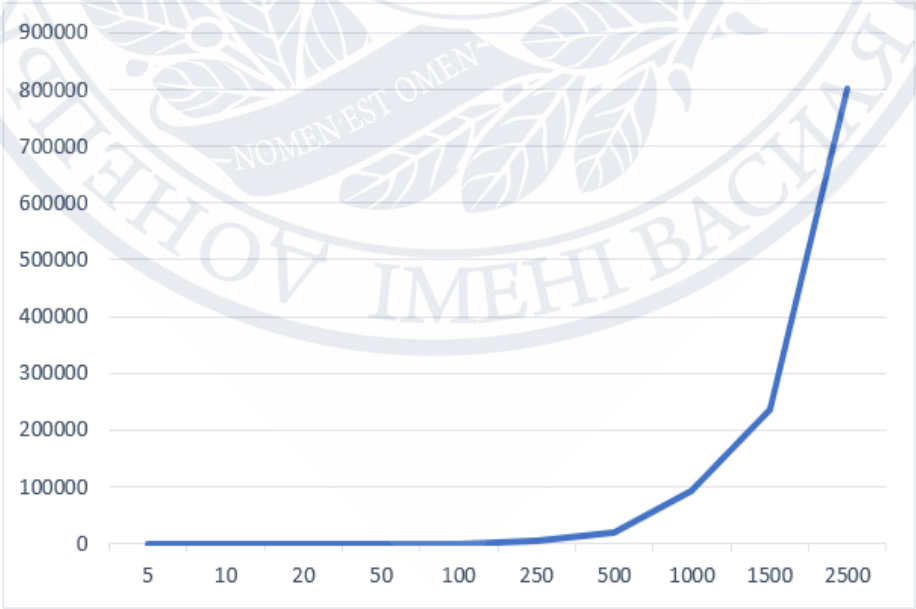

Рисунок 3.7 – час роботи алгоритму в залежності від кількості вхідних даних

3.5 Опис роботи з інтерфейсом

Здійснивши запуск додатку у браузері буде відкрито сторінку авторизації. Доступ до всіх екранів, налаштувань та можливостей є у користувачів із роллю admin. Для інших ролей буде лише обмежено кількість екранів і пунктів меню. Тому розглянемо інтерфейс, з точки зору адміністратора ресурсу:

Рисунок 3.8 – сторінка для автентифікації користувача

Після здійснення авторизації та автентифікації, користувачеві із роллю admin буде відображена стартова сторінка із пунктами меню:

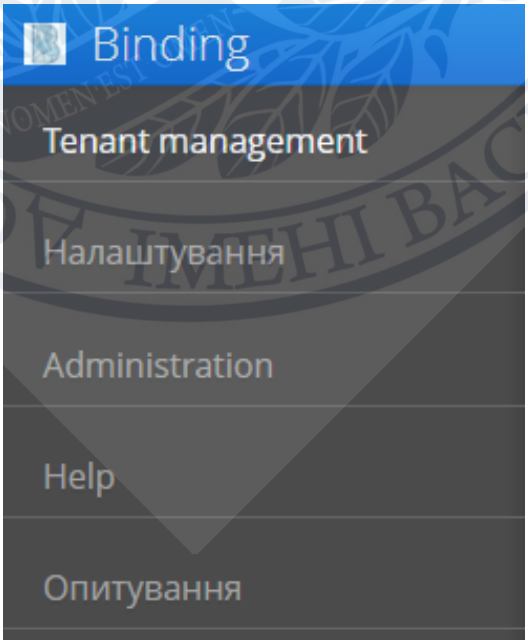

Рисунок 3.9- пункти меню для користувача із роллю admin

У пункті меню «Tenant manager» можливо додати декілька орендарів. Це забезпечить розмежування даних на рівні організацій. Цей пункт необхідно використовувати, якщо система функціонує як окремий і нелажний сервіс. Якщо ж здійснено інтеграцію до іншої системи, поточний пункт меню можна ігнорувати.

Для здійснення tenant-налаштувань, необхідно вибрати пункт меню «Tenant manager», після чого натиснути на пункт «Tenants». В результаті буде відкрито наступний екран:

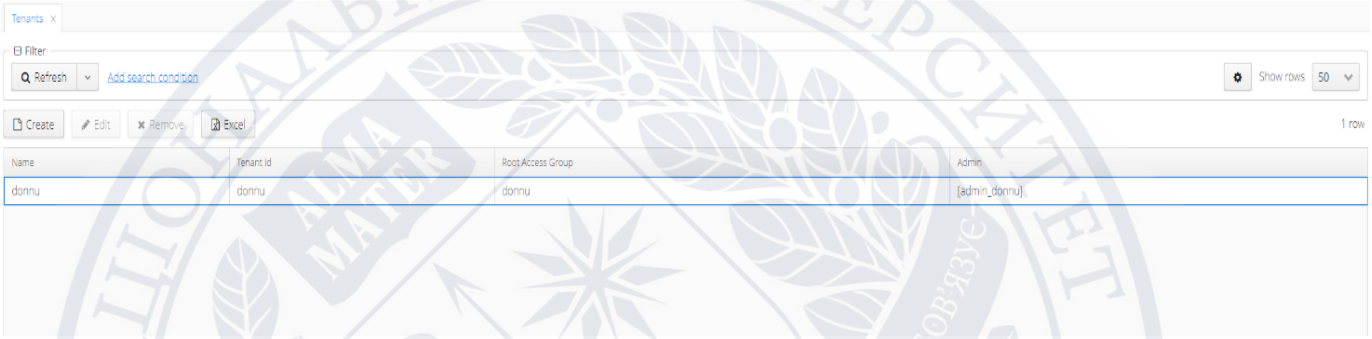

Рисунок 3.10 – відображення усіх орендарів

Для створення нового орендаря необхіно натиснути на кнопку «Create» і заповнити дані у спливаючому вікні, яке продемонстроване на рисунку 3.11.

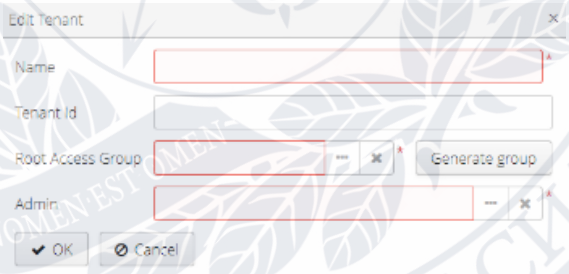

Рисунок 3.11- спливаюче вікно для створення нового орендаря

Пункт меню «Налаштування» створено для забезпечення зручного адміністрування локальним адміністраторам. У ньому містяться наступні підпункти, зображені на рисунку 3.12.

Підпункти «Університети», «Факультети», «Групи(навчальні)», «Користувачі», «Редактор опитувань», «Мультитенантність» містять дуже схожу практичну складову. Тому розглянемо підпункт меню «Групи».

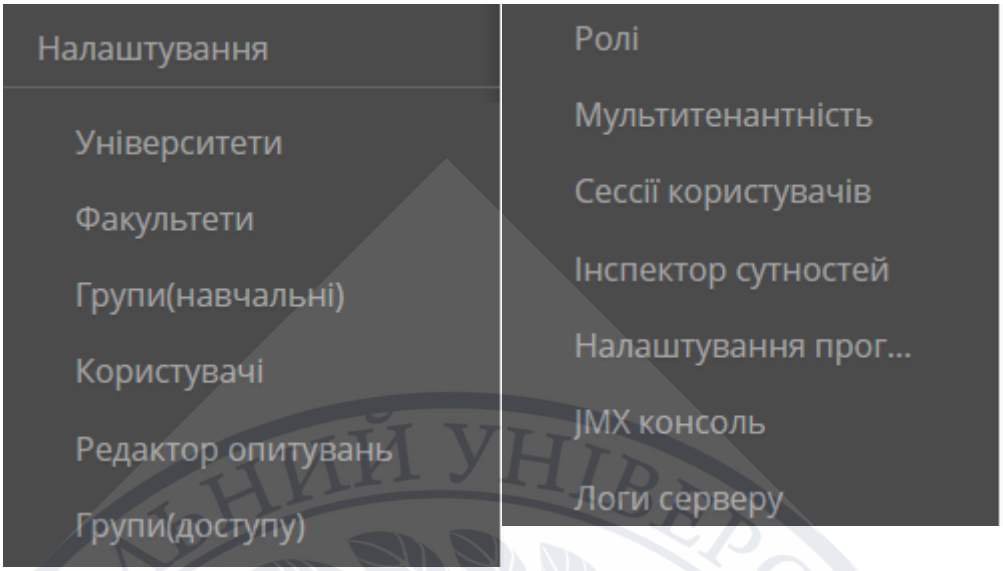

Рисунок 3.12- пункт меню "Налаштування"

Якщо не вказано університет, то буде відображено усі групи, які доступні для поточного користувача. При виборі університету і(або) факультету, буде відображено групи у поточні області.

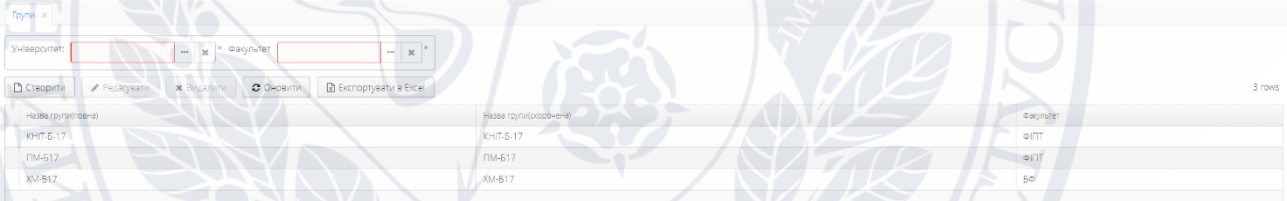

Рисунок 3.13 - відображення усіх груп

При натисканні кнопки «Створити», буде відображено спливаюче вікно для створення групи, яке продемонстровано на рисунку 3.14.

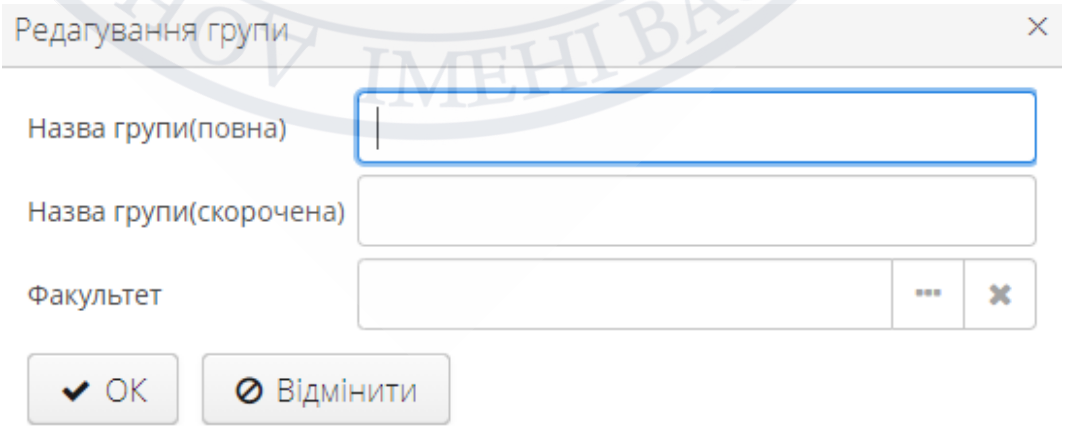

Рисунок 3.14 – створення нової групи

Для редагування групи, необхідно вибрати із списку наявну групу та натиснути «Редагувати»:

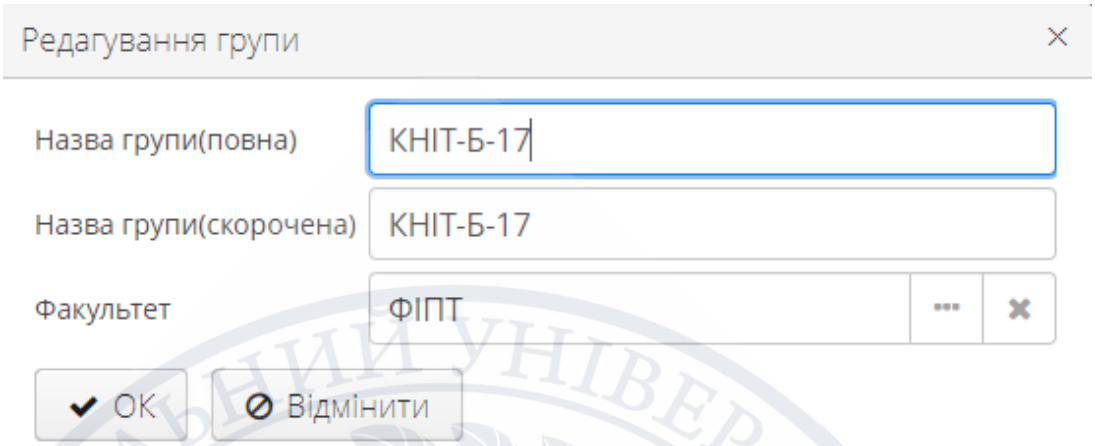

Рисунок 3.15 – редагування групи

Якщо необхідно видалити групу, слід обрати наявну в списку групу та натиснути кнопку «Видалити». В результаті буде відкрито спливаюче вікно із підтвердженням:

Кнопка «Оновити» слугує для синхронізації наявної інформації. Оскільки в

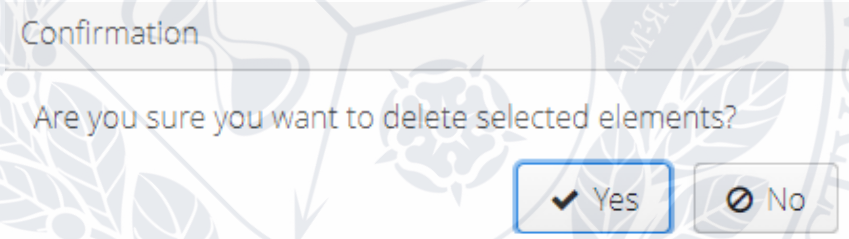

Рисунок 3.16 – підтвердження видалення запису

додатку може знаходитись одночасно декілька адміністраторів, здійснивши зміну певної сутності одним користувачем, іншим ці зміни не будуть відображені одразу. Тому для актуалізації даних слід використовувати поточну кнопку.

Пункт меню «Сесії користувачів» призначений для моніторингу активності у системі на поточний момент(рис. 3.17). Забезпечується можливість

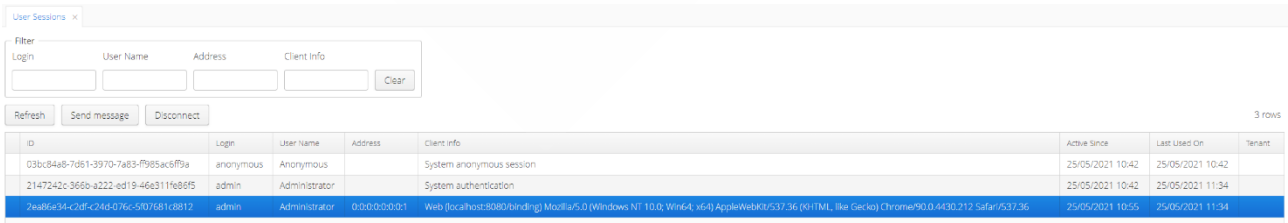

Рисунок 3.17 - відображення усіх активних сесій

відключення користувача, відправка спливаючого повідомлення, а також фільтрації та пошук активних сесій.

«Налаштування програми» містить усі можливі конфігурації, які можливо здійснити без перезавантаження серверу. Для цього використано технологію Java – Mbeans. Рисунок 3.18 демонструє налаштування SMTP серверу.

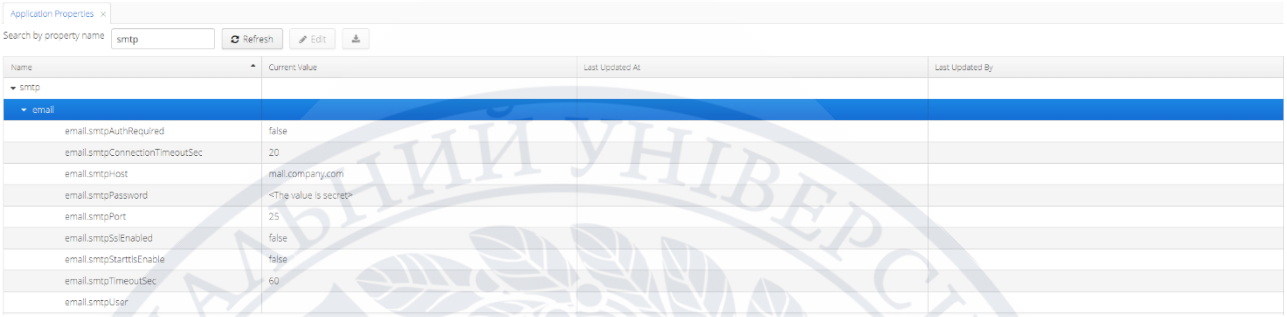

Рисунок 3.18 – налаштування SMTP серверу

«Логи серверу» забезпечує відображення поточних метрик, які було створено сервером. Застосовуєть для моніторингу коректності роботи та пошуку помилок, якщо такі виникають.

У пункті меню «Help» реалізовано підказки для користувача по користуванню системою.

Пункт меню «Administration» забезпечує більш глобальні налаштування додатку у порівнянні із «Налаштування програми». Надається можливість відновлення видалених сутностей, змінювати рівень перегляду метрик(рис. 3.19), доступ до архіву метрик, доступ до усіх можливих налаштувань серверу, поточні системні параметри серверу(споживання оперативної пам’яті та дискового простору), а також створення запланованих завдань(Scheduler Task, ST).

Для створення та проходження опитування необхідно вибрати у меню «Опитування». Користувачеві буде доступний список усіх опитувань(рисунок 3.20) в межах його прав доступу. Активними вважаються опитування, дата

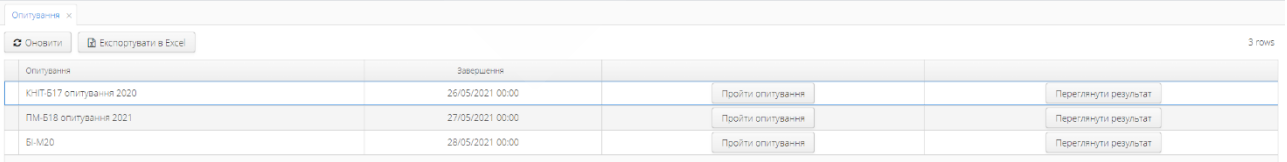

Рисунок 3.20 - відображення усіх опитувань

Group	Parameter	Current value	Average for 1m	Average for uptime
⊟ Application				
	Start Time	25/05/2021 10:41		
	Uptime	4h 30m 46s		
⊟ Memory				
	Heap Memory Usage (Max = 466,432 KB)	156,112 KB	141,481 KB	128,288 KB
	Non-Heap Memory Usage (Max = 0 KB)	180,326 KB	179,849 KB	169,667 KB
	Free Physical Memory Size (Total = 16,151,876 KB)	4,447,000 KB	4,721,032 KB	4,463,092 KB
	Free Swap Space Size (Total = 34,838,748 KB)	6,832,540 KB	7,046,833 KB	6,077,508 KB
⊟ CPU and Threads				
	System CPU Load	12%	5%	5%
	Process CPU Load	0%	0%	0%
	Thread Count (Peak = 82)	77	76	76
⊟ Database				
	Active Connections (Max = 0)	0	0	0
	Idle Connections	0	0	0
	Active Transactions	0	0	0
	Transactions per Second	0	0.013	0.009
⊟ Requests				
	User Sessions	3	3	3
	Web Requests per Second	0.2	0.325	0.013
	Middleware Requests per Second	0	0.013	0.012
	CUBA Scheduled Tasks per Second	0	0	0
	Web Spring Scheduled Tasks per Second	0.4	0.425	0.4
	Middleware Spring Scheduled Tasks per Second	1.6	1.625	1.518

Рисунок 3.19 - відображення системних параметрів додатку

завершення яких менша за поточну та відсутній результат опитування. Решта опитувань вважаються архівними.

У користувачів є можливість пройти активне опитування та вказати бажані пріоритети. Також переглянути результат. Якщо результат відсутній, буде показано попереджувальне вікно:

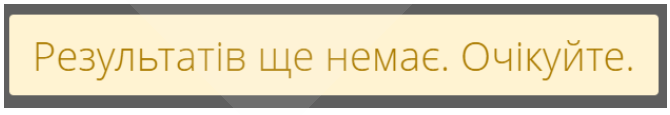

Рисунок 3.21 – попереджувальне вікно при відсутності результату опитування

Також, при завершенні розподілу, буде здійснюватися автоматична відправка результатів усім користувачам, які приймали участь.

Для відслідковування завершення опитування і формування автоматичної відправки повідомлень було використано технологію Spring Sheduler, яка дозволяє створити заплановані завдання.

Для користувачів передбачено можливість налаштувань профілю(рис. 3.22).

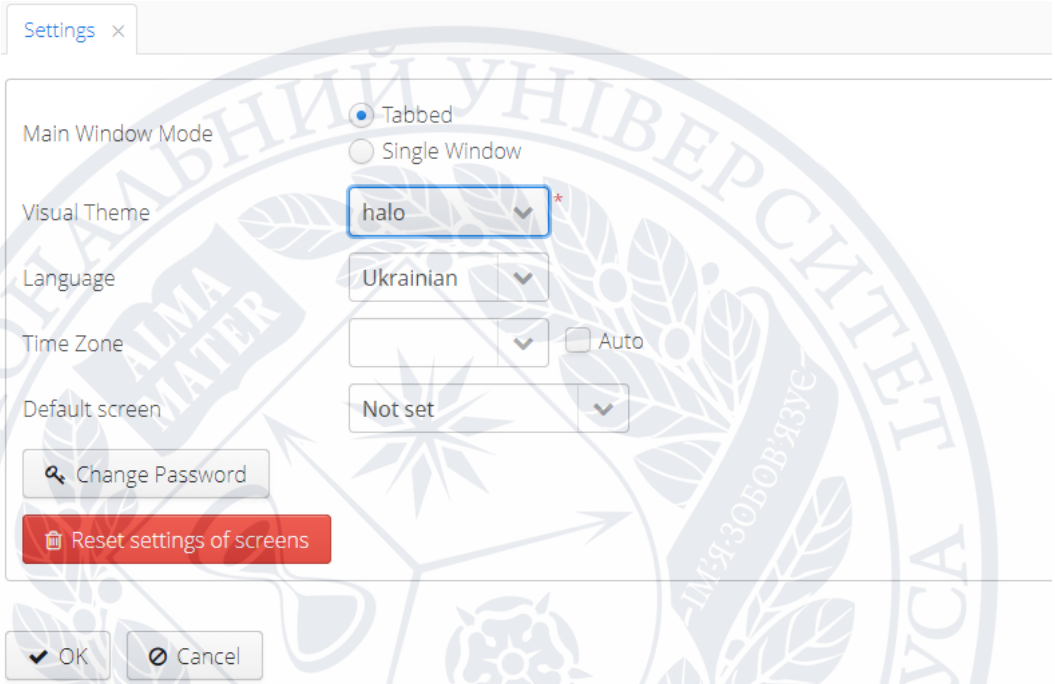

Рисунок 3.22 – налаштування профілю

Користувачеві надається можливість змінити візуальну тему, мову, встановити свій часовий пояс, а також змінити пароль до аккаунту.

3.5 Висновок до розділу 3

У даному розділі було розглянуто особливості налаштувань компонентів та розгортання додатку. Також продемонстровано структури бази даних. Наведено та описано особливості використання користувацького інтерфейсу.

ВИСНОВКИ

В ході виконання бакалаврської роботи було досліджено і виділено основні етапи процесу обрання наукового керівника. Здійснено аналіз переваг та недолік аналогів. Також проведено огляд інструментів які використовуються у Web-розробці.

На основі отриманих результатів було побудовано комп'ютерно математичну модель підсистеми. Для вирішення задачі автоматичного розподілу досліджено алгоритм Гайла-Шеплі.

Докладно вивчена інфраструктура Spring Framework та Vaadin, брокер повідомлень RabbitMQ, бібліотека Hibernate, база даних PostgreSQL, REST архітектура, мікросервісна та монолітна архітектура.

Реалізовано веб-додаток на основі REST архітектури. Веб-додаток має можливість функціонувати як незалежний сервіс або ж бути інтегрованим до іншої системи.

Завдання було виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Infermedica. URL: https://infermedica.com/ (дата звертання: 16.03.2021)
2. PM case study: a patient-doctor platform from scratch. URL: https://medium.com/@adamkiss_57182/pm-case-study-a-patient-doctor-platform-from-scratch-3f473b4786ea (дата звертання: 17.03.2021)
3. Електронний кабінет вступника. URL: https://osvita.ua/consultations/64931/ (дата звертання: 14.04.2021)
4. Рейтинг вступників у ІС Конкурс. URL: https://www.vstup.info/2020/246/i2020i246p710654.html (дата звертання 14.04.2021)
5. Herbert Schildt. Java: The Complete Reference, 10th Edition, New York, 2018. 1344 с.
6. Felipe Gutierrez. Introducing Spring Framework: A Primer, 2014. 369 с.
7. Deepu K Sasidharan, Sendil Kumar N. Full Stack Development with JHipster, Birmingham, 2020. 428 с.
8. Ian Roughley. Practical Apache Struts 2 Web 2.0 Projects, New York, 2007. 358 с.
9. Alejandro Duarte. Data-Centric Applications with Vaadin 8: Develop and maintain high-quality, Mumbai, 2018. 204 с.
10. John Larsen. Get Programming with JavaScript, New York, 2016. 432 с.
11. Dhruti Shah. NS Guidebookode.J: Comprehensive guide to learn Node.js, 2020. 318 с.
12. Cory Gackenheimer. Introduction to React, New York, 2015. 144 с.
13. Callum Macrae. Vue.js: Up and Running: Building Accessible and Performant Web Apps, New York, 2018. 174 с.
14. Mike McGrath. GO Programming in easy steps: Discover Google’s Go language (golang), Warwickshire, 2020. 364 с.
15. David R. Brooks. Guide to HTML, JavaScript and PHP: For Scientists and Engineers, Eagleville, 2011. 428 с.
16. Shoham, Yoav; Leyton-Brown, Kevin. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, New York, 2009. 504 с.

17. В.Є. Бахрушин. Зарахування вступників до закладів вищої освіти як задача багатокритеріального прийняття рішень за умов невизначеності. Запоріжжя, 2019.

18. Systems and software engineering — Architecture description. URL: https://www.iso.org/ru/standard/50508.html (дата звертання: 20.04.2021)

19. Requirements Architecture Part 1: What is Requirements Architecture and Why is it Important?URL: https://seilevel.com/requirements/requirements-architecture-part-1-what-is-requirements-architecture-and-why-is-it-important (дата звертання: 20.04.2021)

20. Martin Fowler. Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Architecture, Boston, 2011. 560 с.

21. Dana Moore. Professional Rich Internet Applications: Ajax and Beyond, New York, 2007. 600 с.

22. Vasudeva Varma. oftware Architecture: A Case Based Approach, Delhi, 2009. 280 с.

23. Balbhadra S. Pradhan. Three-tier Client/server Architecture and the Use of Tuxedo as a Middleware: For Building and Managing Applications, Denver, 2001. 224 с.

24. What is a multi layered software architecture? URL: https://hub.packtpub.com/what-is-multi-layered-software-architecture/ (дата звертання: 01.05.2021)

25. Dirk Krafzig, Karl Banke, Dirk Slama. Enterprise SOA: Service-oriented Architecture Best Practices, Indianapolis, 2004. 418 с.

26. Chellammal Surianarayanan, Gopinath Ganapathy, Raj Pethuru. Essentials of Microservices Architecture: Paradigms, Applications, and Techniques, New York, 2020. 314 с.

27. Bogunuva Mohanram Balachandar. RESTful Java Web Services: A pragmatic guide to designing and building, Mumbai, 2017. 420 с.

28. Master-Slave. URL: http://www.dossier-andreas.net/software_architecture/masterslave.html (дата звертання: 05.05.2021)

29. Rafał Leszko. Continuous Delivery with Docker and Jenkins: Create Secure Applications by Building Complete CI/CD Pipelines, Krakow, 2019. 350 с.

30. Continuous Deployment via GitLab, Jenkins, Docker and Slack. URL: https://medium.com/@ahmetatalay/continous-deployment-via-gitlab-jenkins-docker-and-slack-5d08836d01e0 (дата звертання: 20.05.2021)

31. Client Authentication. URL: https://www.postgresql.org/docs/current/client-authentication.html (дата звертання: 19.05.2021)

32. Richard Stones, Neil Matthew. Beginning Databases with PostgreSQL: From Novice to Professional, New York, 2006. 664 с.

33. Chapter 25. Backup and Restore. URL: https://www.postgresql.org/docs/current/backup.html (дата звертання: 16.05.2021)

34. Core Technologies. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html (дата звертання: 20.05.2021)

35. Alex Bretet. Spring MVC Cookbook, Birmingham, 2016. 466 с.

36. Spring Web Contexts. URL: https://www.baeldung.com/spring-web-contexts (дата звертання: 20.05.2021)

37. Josh Long, Kenny Bastani. Cloud Native Java: Designing Resilient Systems with Spring Boot, Boston, 2017. 648 с.

38. Gabriel N. Schenker. Learn Docker - Fundamentals of Docker 18.x: Everything you need to know, Birmingham, 2018. 398 с.

39. Randall Smith. Docker Orchestration, Mumbai, 2017. 284 с.

40. Overview of Docker Compose. URL: https://docs.docker.com/compose/ (дата звертання: 25.05.2021)

41. Довідка Excel. URL: https://support.microsoft.com/uk-UA/excel (дата звертання: 02.05.2021)

ДОДАТОК А.

Лістинг файлу налаштувань для Docker Compose

```
version: "3"

services:

  backend:
    #image: binding
    build: ./build
    restart: unless-stopped
    environment:
     - DATASOURCE_HOST=db
     - DATASOURCE_DBNAME=${DB_USER}
     - DATASOURCE_USERNAME=${DB_USER}
     - DATASOURCE_PASSWORD=${DB_PASS}
    depends_on:
     - db
    ports:
     - "8050:8080"

  db:
    image: postgres:11.7-alpine
    restart: unless-stopped
    environment:
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASS}
      - PGDATA=/var/lib/postgresql/data/pgdata
    volumes:
```

```
    - db-data:/var/lib/postgresql/data/pgdata
  ports:
    - "5433:5432"

 pgadmin4:
  image: dpage/pgadmin4
  restart: unless-stopped
  volumes:
    - pgadmin4-data:/var/lib/pgadmin
  environment:
    PGADMIN_DEFAULT_EMAIL: ${PGADMIN_USER}
    PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_PASS}

volumes:
  pgadmin4-data:
  db-data:

networks:
  default:
```

ДОДОАТОК Б.

Опис схеми бази даних

Таблиця University.

Назва	Тип	Опис
id_university	UUID	Унікальний ідентифікатор
full_name_university	Varchar(2048)	Повна назва університету
short_name_university	Varchar(512)	Коротка назва університету
tenantId	UUID	Унікальний ідентифікатор орендаря

Таблиця Faculty.

Назва	Тип	Опис
id_faculty	UUID	Унікальний ідентифікатор
id_university	UUID	Ідентифікатор університету
full_name_faculty	Varchar(2048)	Повна назва факультету
short_name_faculty	Varchar(512)	Коротка назва факультету
tenantId	UUID	Унікальний ідентифікатор орендаря

Таблиця Groups.

Назва	Тип	Опис
id_group	UUID	Унікальний ідентифікатор

id_faculty	UUID	Ідентифікатор факультету
full_name_group	Varchar(2048)	Повна назва групи
short_name_group	Varchar(512)	Коротка назва групи
start_study	Timestamp	Початок навчання
end_study	Timestamp	Завершення навчання
tenantId	UUID	Унікальний ідентифікатор орендаря

Таблиця Tenants.

Назва	Тип	Опис
id_tenant	UUID	Унікальний ідентифікатор
tenant_name	Varchar(512)	Назва орендаря

Таблиця Users.

Назва	Тип	Опис
id_user	UUID	Унікальний ідентифікатор
last_name	Varchar(512)	Прізвище
first_name	Varchar(512)	Імя
surname	Varchar(512)	По-батькові
email	Varchar(512)	Корпоративна або власна пошта
password	Varchar(512)	Хешований пароль користувача
enabled	Boolean	Чи активний аккаунт
account_non_expired	Boolean	Чи дійсні права доступу
account_non_locked	Boolean	Чи не заблокований аккаунт

tenantId	UUID	Унікальний ідентифікатор орендаря

Таблиця Users_Groups.

Назва	Тип	Опис
id_group_user	UUID	Унікальний ідентифікатор
id_ group	UUID	Навчальна група
id_user	UUID	Користувач

Таблиця Users_Faculty.

Назва	Тип	Опис
id_faculty_user	UUID	Унікальний ідентифікатор
id_faculty	UUID	Факультет на якому працює викладач
id_user	UUID	Користувач

Таблиця Permission.

Назва	Тип	Опис
id_permission	UUID	Унікальний ідентифікатор
name_permission	Varchar(512)	Назва дозволу

Таблиця Access_Group.

Назва	Тип	Опис
id_access_group	UUID	Унікальний ідентифікатор
name_access_group	Varchar(512)	Назва групи доступу

Таблиця Access_Group_Permission.

Назва	Тип	Опис
id_access_group_permission	UUID	Унікальний ідентифікатор

id_access_group	UUID	Група доступу
id_permission	UUID	Дозвіл

Таблиця Access_Group_Users.

Назва	Тип	Опис
id_access_group_user	UUID	Унікальний ідентифікатор
id_access_group	UUID	Група доступу
id_user	UUID	Користувач

Таблиця Role.

Назва	Тип	Опис
id_role	UUID	Унікальний ідентифікатор
name_role	Varchar(512)	Назва ролі

Таблиця Users_Role.

Назва	Тип	Опис
id_role_user	UUID	Унікальний ідентифікатор
id_role	UUID	Роль
id_user	UUID	Користувач

Таблиця Voting.

Назва	Тип	Опис
id_voting	UUID	Унікальний ідентифікатор
id_group	UUID	Група для якої проводиться опитування
name_voting	Varchar(1024)	Назва опитування
date_end_voting	Timestamp	Дата завершення опитування

tenantId	UUID	Унікальний ідентифікатор орендаря

Таблиця Want_Pairs.

Назва	Тип	Опис
id_want_pairs	UUID	Унікальний ідентифікатор
id_user_voted	UUID	Користувач, що пройшов опитування
id_user_given_priority	UUID	Користувач, якому встановлено пріоритет
priority	long	Пріоритетність
tenantId	UUID	Унікальний ідентифікатор орендаря

Таблиця Result.

Назва	Тип	Опис
id_result	UUID	Унікальний ідентифікатор
id_voting	UUID	Опитування
id_user_given_priority	UUID	Науковий керівник
id_user_student	UUID	Студент
tenantId	UUID	Унікальний ідентифікатор орендаря