Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного забезпечення системи управління роботою бібліотеки з використанням фреймворку Spring і платформи Apache Kafka»

Виконав: студент 4 курсу
групи 4ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Дятлюк І.С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ЗІ Войтович О.П.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«12» березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Дятлюку Івану Сергійовичу

1. Тема роботи – розробка програмного забезпечення системи управління роботою бібліотеки з використанням фреймворку Spring і платформи Apache Kafka.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: тип програми – веб система; модель розробки – ітеративна; СУБД для керування базою даних – MySQL; засоби керування чергами повідомлень – Apache Kafka; вхідні дані: інформація про книги, особисті дані користувачів, метадані запитів; вихідні дані: відображення списку книг та категорій, сповіщення, аналітичні дані та рекомендації; середовище розробки – IntelliJ Idea; мова програмування – Java; використанні додаткові бібліотеки та фреймворки – Spring (Boot, Data JPA, Web, Kafka).

4. Зміст текстової частини: вступ, обґрунтування вибору методу розробки та постановка задач дослідження, розробка моделі, методів та алгоритмів роботи програмного продукту; розробка програмного забезпечення; тестування програмного забезпечення; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; новизна і практична цінність одержаних результатів; порівняльний аналіз аналогів; архітектура системи; модель системи; блок-схема загального алгоритму роботи системи; розробка методу формування рекомендацій; розробка методу взаємодії з Telegram ботом; головна сторінка та налаштування боту; сторінка списку книг та

отримання сповіщень; тестування системи; апробація та публікація матеріалів бакалаврської дипломної роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О.В. к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задач дослідження	14.03.24 – 19.03.24	виконано
2	Розробка архітектури та моделі системи	19.03.24 – 26.03.24	виконано
3	Розробка загального алгоритму роботи системи	26.03.24 – 03.04.24	виконано
4	Розробка методу формування рекомендацій	03.04.24 – 9.04.24	виконано
5	Розробка методу взаємодії з Telegram ботом	09.04.24 – 17.04.24	виконано
5	Аналіз та вибір засобів для реалізації програмної системи	17.04.24 – 20.04.24	виконано
6	Розробка програмного забезпечення	20.04.24 – 19.05.24	виконано
7	Тестування системи	19.05.24 – 23.05.24	виконано
8	Оформлення матеріалів до захисту БДР	23.05.24 – 30.05.24	виконано

Студент _____ **І.С. Дятлюк**
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ **О. В. Романюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 72 сторінок формату А4, на яких є 29 рисунків, 2 таблиці, список використаних джерел містить 23 найменування.

У бакалаврській дипломній роботі було проведений детальний аналіз розробки автоматизованих систем управління роботою бібліотек. Сформульовано мету дослідження – підвищення ефективності управління роботою бібліотеки за рахунок впровадження таких сучасних технологій як штучний інтелект та телеграм боти, що сприятиме популяризації літератури та бібліотек.

Запропоновано вдосконалений метод формування рекомендацій для користувача, який, на відміну від аналогів, використовує розширений діапазон факторів та параметрів для формування рекомендацій та штучний інтелект, що дозволило підвищити точність формування рекомендацій. Також, запропоновано вдосконалений метод впровадження сповіщень, який, на відміну від класичного алгоритму, окрім використання електронної адреси, додатково використовує інтеграцію із Telegram застосунком, що дозволило покращити досвід користувача та загальний процес сповіщень, виведення рекомендацій та взаємодії з користувачем для отримання зворотного зв’язку.

Розроблено програмний застосунку автоматизованої системи управління роботою бібліотеки за допомогою мови програмування Java, зокрема фреймворку Spring та платформи Apache Kafka.

Отримані в бакалаврській дипломній роботі результати можна використати для впровадження в бібліотечну справу, зокрема для вдосконалення та автоматизації основних процесів.

Ключові слова: бібліотека, книги, автоматизована система, формування рекомендацій, чат-боти.

ABSTRACT

The bachelor's thesis consists of 72 A4 pages, which include 29 figures and 2 tables. The list of references contains 23 sources.

The bachelor's thesis provides a detailed analysis of the development of automated library management systems. The research goal is formulated as enhancing the efficiency of library management by implementing modern technologies such as artificial intelligence and Telegram bots, which will promote literature and libraries.

An improved method for generating user recommendations is proposed. Unlike its counterparts, this method uses a broader range of factors and parameters to generate recommendations and employs artificial intelligence, which has allowed for increased recommendation accuracy. Additionally, an improved notification method is proposed. Unlike the classical algorithm, it integrates with the Telegram application in addition to using email, improving the user experience and the overall process of notifications, recommendation delivery, and user interaction for feedback collection.

A software application for an automated library management system was developed using the Java programming language, specifically the Spring framework and the Apache Kafka platform.

The results obtained in the bachelor's thesis can be used for implementation in library operations, particularly for the enhancement and automation of core processes.

Keywords: library, books, automated system, recommendation generation, chatbots.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. В сучасному інформаційному суспільстві, де доступ до знань та інформації відіграє ключову роль у розвитку особистості та суспільства в цілому, бібліотеки залишаються важливими, оскільки можна виділити ряд ключових причин, що впливають на це:

1. Центр знань та освіти. Бібліотеки є важливими центрами накопичення, зберігання та поширення знань. Вони надають доступ до різноманітної літератури, актуальних досліджень та інших інформаційних ресурсів, які сприяють освіті, самоосвіті та розвитку особистості.

2. Культурна спадщина. Бібліотеки зберігають культурну спадщину суспільства, включаючи рідкісні та історичні видання, документи, картини та інші артефакти. Це допомагає зберегти історію та культурну ідентичність нації.

3. Громадський простір. Бібліотеки створюють простір для громадських зустрічей, обміну ідеями та культурного обміну. Вони часто виступають як місця для проведення лекцій, виставок, клубів за інтересами та інших заходів, що сприяють соціальній взаємодії.

4. Підтримка досліджень. Бібліотеки є важливими ресурсами для наукових досліджень та навчання. Вони надають доступ до актуальних наукових видань, журналів, баз даних та інших джерел, що допомагає в проведенні досліджень та розвитку науки.

Автоматизація управління бібліотекам є вагомим кроком у забезпеченні їх ефективності, покращені доступності [1]. Основною метою якої є підтримка існуючих фондів та впровадження сучасних технологій у бібліотечку сферу для популяризації. Основні переваги, як слідують за автоматизацією є:

1. Ефективне управління ресурсами. Автоматизація дозволяє оптимізувати процеси каталогізації, інвентаризації та видачі літератури, що зменшує час та зусилля, необхідні для цих операцій.

2. Зручний доступ до інформації. Системи управління бібліотеками дозволяють швидко знаходити та замовляти літературу через онлайн-каталоги, що зручно для користувачів.

3. Аналіз та звітність. Автоматизовані системи надають засоби для аналізу використання ресурсів бібліотеки, що дозволяє керівництву приймати обґрунтовані рішення щодо поліпшення роботи.

4. Збереження інформації. Системи управління дозволяють зберігати та організовувати інформацію про колекції, користувачів та інші аспекти діяльності бібліотеки, що забезпечує її надійність та стабільність.

Розробка автоматизованої системи управління бібліотеками, як мікросервісного програмного забезпечення має ряд переваг порівняно із монолітною архітектурою, які включають: легку масштабованість та швидкість впровадження змін. Це означає, що система буде стійка до високих навантажень та великої кількості даних і в першу чергу забезпечить безперебійну та ефективну роботу.

Актуальність розробки нової системи управління бібліотекою порівняно з існуючими аналогами полягає в необхідності вдосконалення та адаптації до сучасних вимог і технологій. Існуючі системи є застарілими, що впливає на ефективне використання ресурсів, також такі системи, в більшості випадків, є вузьконаправлені і базуються лише на вимогах конкретного бізнесу, або не задовольняють потреб користувачів у зручності та функціональності. Розробка нової системи дозволить врахувати сучасні тенденції в галузі інформаційних технологій, такі як мікросервісна архітектура, штучний інтелект [2], аналітика даних, використання різного роду ботів для взаємодії з користувачами, що забезпечить більшу гнучкість, продуктивність та зручність для користувачів.

Тому реалізація автоматизованої системи управління бібліотекою є актуально розробкою у наш час.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності управління роботою бібліотеки за рахунок впровадження таких сучасних технологій як штучний інтелект та телеграм боти, що сприятиме популяризації літератури та бібліотек.

Відповідно до поставленої мети потрібно вирішити такі завдання:

- здійснити аналіз стану питання та методів розв'язання задачі;
- спроектувати модель даних та архітектуру системи;
- розробити загальний алгоритм роботи системи;
- розробити методи обробки даних та формування рекомендацій;
- розробити методи взаємодії із Telegram ботом для формування сповіщень;
- виконати програму реалізацію системи для управління роботою бібліотеки;
- провести тестування систем;
- розробити інструкцію користувача;
- розробити системні вимоги програмної системи.

Об'єктом дослідження є процес управління роботою бібліотек.

Предметом дослідження є методи та засоби розробки програмного забезпечення для управління роботою бібліотек.

Методи дослідження. У процесі досліджень використовувалися такі методи дослідження: порівняльний аналіз функціональних характеристик програм-аналогів для виявлення їх недоліків та постановки задач розробки; теорія алгоритмів для розробки та візуалізації алгоритмів роботи програмного забезпечення; елементи математичної статистики для генерування статистичних даних у додатку; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Удосконалено метод формування рекомендацій для користувача, який, на відміну від аналогів, використовує розширений діапазон факторів та

параметрів для формування рекомендацій та штучний інтелект, що дозволило підвищити точність формування рекомендацій.

2. Удосконалено метод впровадження сповіщень, який, на відміну від класичного алгоритму, окрім використання електронної адреси, додатково використовує інтеграцію із Telegram застосунком, що дозволило покращити досвід користувача та загальний процес сповіщень, виведення рекомендацій та взаємодії з користувачем для отримання зворотного зв’язку.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень було запропоновано алгоритми та розроблено програмні засоби для вдосконалення процесу управління роботою бібліотеки.

Особистий внесок здобувача. Усі наукові результати, що викладені у бакалаврській дипломній роботі, отримано автором самостійно. У наукових працях, опублікованих у співавторстві, автору належать такі результати: переваги інтеграції сучасних технологій із програмним забезпеченням для ефективного управління ресурсами бібліотеки [1], впровадження гібридної системи рекомендацій із використанням штучного інтелекту [2].

Апробація та публікація матеріалів бакалаврської дипломної роботи. Результати роботи доповідалися на:

- LIII Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (Вінниця, 2024) [1];
- Молодь в науці: дослідження, проблеми, перспективи (МН-2024), Вінниця 2024 [2].

Публікації. За тематикою дослідження опубліковано 2 наукові праці у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз способів автоматизації управління бібліотекою

У сучасному світі бібліотеки стикаються з низкою викликів, пов'язаних зі зростаючим обсягом інформації, змінами в поведінці користувачів та потребою у підвищенні ефективності роботи. Автоматизація управління бібліотекою виступає як ефективне рішення для подолання цих викликів та забезпечення якісного обслуговування користувачів [1].

Існує два методи управління бібліотекою. Перший із них є найстарішим та найпопулярнішим у наш час є – традиційне управління, що полягає роботі зі паперовими каталогами, ручним веденням записів про видачі та повернення книг, підтримці фізичного зберігання бібліотечних ресурсів.

Традиційні методи ведення обліку в бібліотеках мають декілька недоліків, які ускладнюють роботу бібліотекарів та знижують зручність користування для читачів. По-перше, паперові каталоги та ручне ведення обліку видачі та повернення книг вимагають багато часу та зусиль, що робить процес трудомістким та незручним. Пошук інформації в таких каталогах може бути складним та часів затратним, а ручне ведення записів часто призводить до помилок. Фізичне зберігання книг на полицях обмежує доступ до них та ускладнює їх пошук.

По-друге, традиційні методи є неефективними у використанні ресурсів бібліотеки та управлінні її фондами. Складність відстеження використання книг та прийняття рішень щодо їх оновлення або заміни призводить до неефективного управління бібліотечними ресурсами. Неможливість пропонувати користувачам персоналізовані рекомендації та послуги обмежує їх задоволення від візиту в бібліотеку.

По-третє, традиційні методи також незручні для користувачів. Пошук книг за допомогою паперових каталогів може бути складним, а відсутність можливості резервування книг онлайн або отримування повідомлень про

новинки робить процес використання бібліотеки менш зручним для читачів. Незручності також виникають при оформленні видачі та повернення книг.

Ще одним недоліком є небезпека втрати та пошкодження інформації через фізичні носії, такі як паперові каталоги та записи. Фізичні книги піддаються ризику пошкодження або крадіжки. Нарешті, традиційні методи не можуть ефективно використовуватися в бібліотеках з великими фондами та великою кількістю користувачів, що обмежує їх масштабованість. Крім того, вони не сумісні з сучасними технологіями, такими як електронні книги, цифрові ресурси та онлайн-сервіси, що робить їх менш привабливими для сучасних бібліотек.

Іншим методом управління бібліотекою є впровадження автоматизованих систем. Ця стратегія передбачає використання сучасних технологій і програмного забезпечення для оптимізації процесів ведення обліку, організації ресурсів та забезпечення зручності користувачів. Застосування автоматизованих систем дозволяє замінити паперові каталоги та ручне ведення записів на електронні бази даних, що спрощує доступ до інформації та зменшує ризик помилок. Крім того, автоматизація управління бібліотекою дозволяє ефективно використовувати ресурси бібліотеки, враховуючи потреби користувачів, шляхом аналізу використання книг та рекомендаційних систем.

Однією з головних переваг автоматизованих систем є їх продуктивність та ефективність управління ресурсами. Автоматизовані системи дозволяють швидко відстежувати видачу та повернення книг, керувати фондом та встановлювати пріоритети для оновлення колекції. Крім того, вони можуть надавати користувачам інтерактивні сервіси, такі як онлайн-резервування книг та персоналізовані рекомендації, що покращує їх користувацький досвід.

Автоматизовані системи також спрощують взаємодію з сучасними технологіями та стандартами. Вони забезпечують сумісність з електронними книгами, цифровими ресурсами та онлайн-сервісами, що дозволяє бібліотекам залишатися конкурентоспроможними в цифровій епохі.

Отже, сучасні виклики, з якими стикаються бібліотеки, вимагають інноваційних підходів до управління та використання технологій. Автоматизація

управління бібліотекою не лише покращує ефективність роботи, але й забезпечує більшу доступність та зручність для користувачів, що є ключовими факторами у залученні та утриманні читачів. Ці системи дозволяють бібліотекам стати більш гнучкими у відповідях на зміни в запитах користувачів та тенденціях інформаційних технологій. Застосування сучасних технологій в управлінні бібліотекою є актуальним не тільки через збільшення ефективності обслуговування та оптимізацію ресурсів, але й з точки зору стратегічного розвитку. У порівнянні з традиційним управлінням, автоматизовані системи надають більшу швидкість, точність та зручність в обслуговуванні користувачів. Вони також забезпечують більшу масштабованість, дозволяючи бібліотекам ефективно працювати з великими обсягами інформації та великою кількістю користувачів.

1.2 Порівняльний аналіз аналогів систем управління роботи бібліотек

На сьогоднішній день існує ряд автоматизованих систем управління бібліотекою, які пропонують різноманітні можливості для оптимізації роботи бібліотек та покращення обслуговування користувачів. Проведення порівняльного аналізу цих систем допоможе визначити їх переваги та недоліки, а також визначити потенційні напрямки для вдосконалення власної автоматизованої системи управління бібліотекою.

Koha (див. рис. 1.1) – це безкоштовна автоматизована система управління бібліотеками з відкритим вихідним кодом, яка користується популярністю серед бібліотек завдяки своїй гнучкості та можливості налаштування [3]. Koha пропонує широкий спектр функцій, таких як електронний каталог, управління користувачами, цифрові ресурси та інструменти звітності.

Основний функціонал системи:

- електронний каталог з пошуком та фільтрацією за різними критеріями;
- управління користувачами з можливістю створення облікових записів, визначення прав доступу та відстеження історії видачі;
- цифрові ресурси, такі як електронні книги, статті та інші матеріали;

- інструменти звітності для аналізу використання бібліотечних ресурсів.

До недоліків програмного продукту належать:

1. Обмежені можливості пошуку. Koha не пропонує розширених можливостей пошуку як, пошук за темою, автором, ISBN, жанром та іншими критеріями.

2. Відсутність управління електронними ресурсами. Koha не має вбудованих функцій для управління електронними ресурсами, такими як електронні книги, статті та інші матеріали.

3. Обмежені можливості формування персоналізованих рекомендацій. Koha може пропонувати лише базові рекомендації на основі історії видачі книг.

4. Відсутність інтеграції з телеграм ботом. Koha не має вбудованої інтеграції з Telegram-ботом та іншими застосунками для впровадження сповіщень.

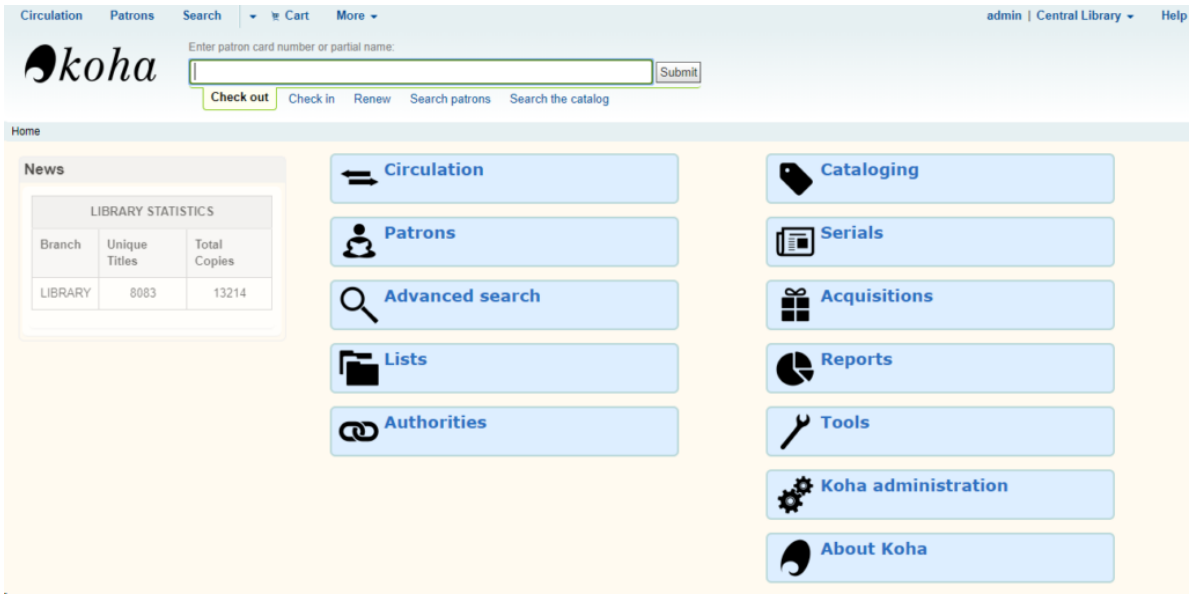

Рисунок 1.1 – Зображення програмного продукту «Koha»

FOLIO (див. рис. 1.2) – це безкоштовна автоматизована система управління бібліотеками з відкритим вихідним кодом, яка розроблена з урахуванням потреб сучасних бібліотек [4]. FOLIO пропонує модульну архітектуру, яка дозволяє бібліотекам вибирати лише ті функції, які їм необхідні.

Основний функціонал системи:

1. Електронний каталог з потужним пошуком та можливостями фільтрації.

2. Управління користувачами з можливістю створення облікових записів, визначення прав доступу та відстеження історії видачі.

3. Інструменти управління цифровими правами для контролю доступу до електронних ресурсів.

4. Підтримку мультимедійних ресурсів, таких як аудіо- та відеофайли.

До недоліків програмного продукту належать:

1. Відсутність інтеграції з телеграм ботом. FOLIO не має вбудованої інтеграції з Telegram-ботом та іншими застосунками для впровадження сповіщень.

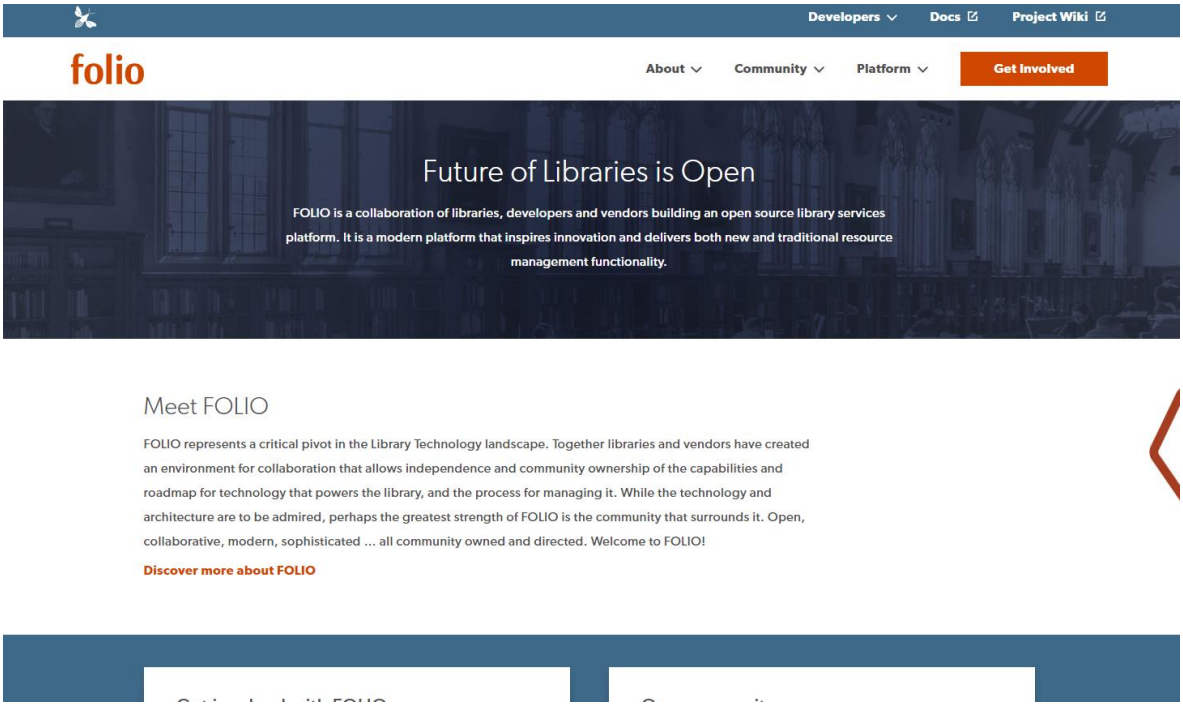

Рисунок 1.2 – Зображення програмного продукту «FOLIO»

Alma (див. рис. 1.3) – це комерційна автоматизована система управління бібліотеками від компанії Ex Libris, яка пропонує широкий спектр функцій та високий рівень підтримки [5]. Alma використовується в багатьох великих бібліотеках світу і пропонує ряд функцій, які орієнтовані на потреби великих інституцій.

Основний функціонал системи:

- електронний каталог з розширеними можливостями пошуку;

- управління користувачами з детальним профілюванням та ролями користувачів;

- управління електронними ресурсами, включаючи придбання, ліцензування та інтеграцію з базами даних;

- аналітика даних для отримання детальних звітів про використання бібліотечних ресурсів;

- широку інтеграцію з іншими системами, такими як бухгалтерське програмне забезпечення та системи управління навчальним процесом.

До недоліків програмного продукту належать:

1. Висока вартість. Alma є комерційною АСУБ, яка може бути дорогою для бібліотек з обмеженим бюджетом;

2. Складність використання. Alma може бути складною для налаштування та використання для бібліотек з обмеженими ресурсами.

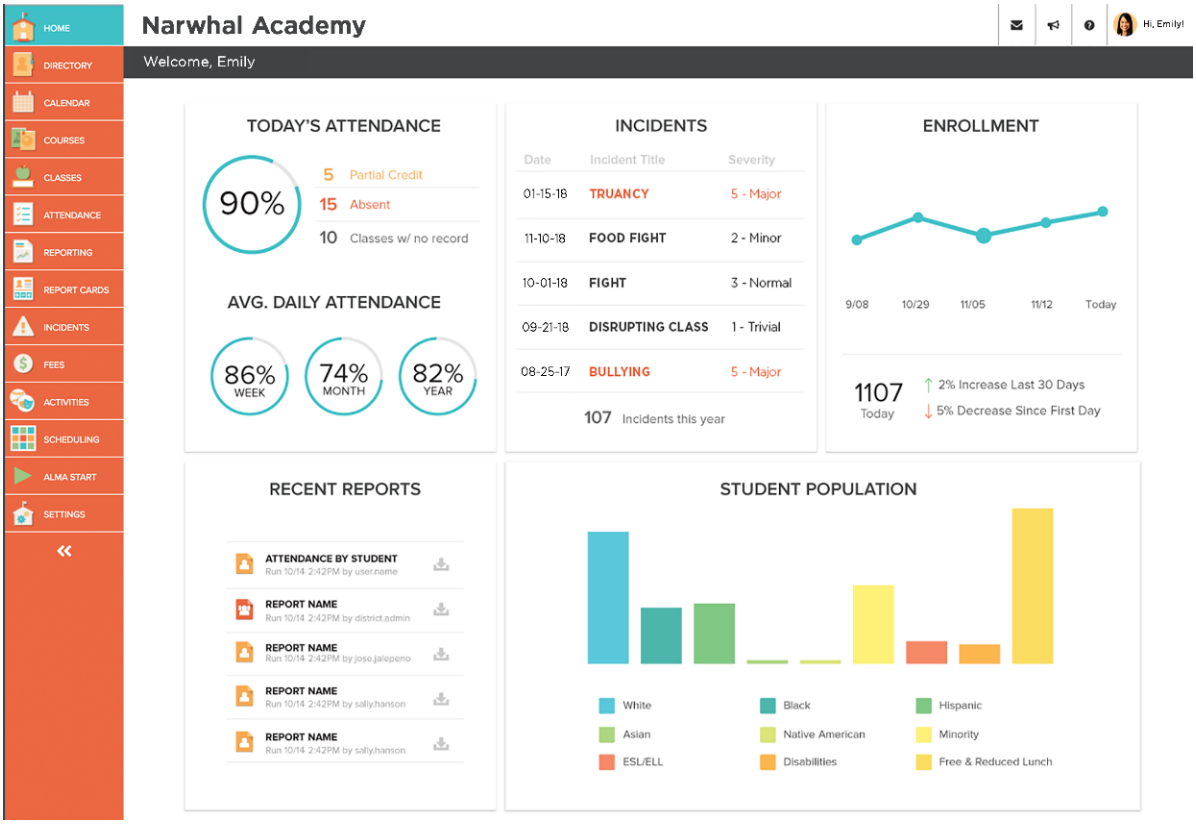

Рисунок 1.3 – Зображення програмного продукту «Alma»

Порівняльний аналіз особливостей розглянутих аналогів із власною розробкою подано в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз програмних продуктів аналогів

Критерії	Koha	FOLIO	Alma	Власна розробка
Наявність безкоштовної ліцензії	-	-	-	+
Розширена фільтрація книг	+/-	+	+	+
Управління користувачами	+	+	+	+
Підтримка цифрових ресурсів	+	+	+	-
Інтеграція сповіщень у чат ботах	-	-	-	+
Підсумок	1.5	3	3	4

Нижче наведено опис критерії, за якими було проведено порівняння:

1. Наявність безкоштовної ліцензії. Цей критерій оцінює доступність системи безкоштовно для користування. Деякі автоматизовані системи управління бібліотеками можуть надавати базовий функціонал безкоштовно, а додаткові можливості можуть бути доступні за плату. Наявність безкоштовної ліцензії може бути важливою для бібліотек з обмеженим бюджетом, що шукають ефективне рішення без додаткових витрат.

2. Розширена фільтрація книг. Цей критерій оцінює можливості системи здійснювати розширену фільтрацію книг за різними критеріями, такими як жанр, автор, рік видання тощо. Розширена фільтрація дозволяє користувачам швидко знаходити потрібні книги серед великого обсягу бібліотечних ресурсів.

3. Управління користувачами. Цей критерій оцінює можливості системи управління користувачами, включаючи реєстрацію нових користувачів, видачу книг, ведення обліку видачі та повернення книг, а також взаємодію з користувачами щодо їх запитів та потреб.

4. Підтримка цифрових ресурсів. Цей критерій оцінює можливості системи підтримувати цифрові ресурси, такі як електронні книги, аудіо-книги, журнали

тощо. Підтримка цифрових ресурсів може розширити доступ користувачів до бібліотечних матеріалів та підвищити їх зручність користування.

5. Інтеграція сповіщень у чат ботах. Цей критерій оцінює можливість системи інтегруватися з чат-ботами для надання користувачам сповіщень про новинки, нагадувань про здійснення повернення книг, а також для надання додаткової інформації та підтримки. Інтеграція сповіщень у чат ботах може покращити комунікацію з користувачами та забезпечити їм більш зручний доступ до бібліотечних послуг

Проаналізувавши результати порівняльного аналізу аналогів наведених у таблиці 1.1, варто зазначити актуальність та мету створення власної системи, що дозволить розширити та вдосконалити функціональні можливості існуючих програмних продуктів.

1.3 Аналіз напрямків удосконалення програмного забезпечення для автоматизації управління бібліотеками

Звернувши увагу на аналіз можливостей існуючих систем для управління робою бібліотеки, розглянемо можливі варіанти їх вдосконалення.

Сучасні виклики вимагають гнучкості, швидкості адаптації до змін і поліпшення сервісу для користувачів, що може бути досягнуто через впровадження новітніх технологій та підходів у розробці програмного забезпечення.

Запровадження мікросервісної архітектури є одним із ключових шляхів модернізації існуючих систем. Мікросервісна архітектура дозволяє розбити велику монолітну систему на менші, незалежні модулі, кожен з яких відповідає за окремий аспект функціональності. Це поліпшує гнучкість розвитку та підтримки системи, забезпечує краще масштабування та відмовостійкість, а також спрощує оновлення окремих частин системи без ризику для цілісності всього програмного забезпечення [6].

Вдосконалення процесів комунікації з користувачами через телеграм-боти може суттєво підвищити ефективність взаємодії з користувачами. Інтеграція

телеграм-ботів у систему управління бібліотекою надасть користувачам можливість швидко отримувати інформацію про доступність книг, стан замовлень, нагадування про повернення книг та інші бібліотечні послуги без необхідності відвідування самої бібліотеки. Це зробить сервіс більш доступним та зручним.

Впровадження різного роду методів для формування рекомендацій користувача стає важливим аспектом залучення та утримання користувачів. Застосування сучасних алгоритмів машинного навчання та аналізу даних для аналізу читацьких переваг та поведінки дозволить створювати персоналізовані рекомендації, які не тільки задовольняють потреби користувачів, але й спонукають їх до вивчення нових авторів та жанрів. Це збільшить використання бібліотечних ресурсів та підвищить загальну задоволеність користувачів.

1.4 Аналіз методів розв’язання поставленої задачі

Розробка автоматизованої системи управління бібліотеками передбачає декілька методів та підходів, що включають вибір мови програмування та вибору архітектури програмного забезпечення.

1. Монолітна архітектура. При цьому підході вся система розробляється як єдиний великий проект, де всі компоненти системи тісно інтегровані між собою. Такий підхід може полегшити розробку та тестування в короткостроковій перспективі, особливо для менших чи менш складних систем, але може стати складним для масштабування та підтримки в довгостроковій перспективі [7].

2. Мікросервісна архітектура. Цей метод передбачає розбиття функціоналу системи на незалежні модулі, які можуть бути розроблені, тестовані, розгорнуті та масштабовані незалежно один від одного. Мікросервіси можуть бути зв'язані за допомогою API, що дозволяє створювати більш гнучкі та адаптивні системи. Цей підхід підходить для великих систем, що вимагають високого рівня масштабування та гнучкості [8].

3. Serverless-архітектура. Serverless-підхід дозволяє розробникам зосередитися на коді, покладаючись на хмарні провайдери для управління

серверною інфраструктурою [9]. Це може значно знизити вартість та складність управління інфраструктурою, особливо для систем, де навантаження може суттєво коливатися.

Для розробки власної автоматизованої системи було вирішено використовувати мікросервісний підхід для розробки архітектури, оскільки цей підхід дозволяє кожному сервісу функціонувати незалежно, що спрощує процес додавання нових функцій, виправлення помилок та масштабування системи.

Крім того, мікросервісна архітектура покращує надійність системи, оскільки збій в одному сервісі не впливає безпосередньо на роботу інших сервісів. Це особливо важливо для бібліотечних систем, де потрібно забезпечити безперебійний доступ до каталогів та інших ресурсів. Завдяки ізоляції мікросервісів також спрощується процес впровадження змін та оновлень, знижуючи ризики для стабільності загальної системи.

Також, варто розглянути методи формування персоналізованих рекомендацій, оскільки цей функціонал відіграє важливу роль в контексті користувацького досвіду.

Системи рекомендацій на основі контенту фокусуються на аналізі тексту книг, описів, жанрів та інших метаданих для визначення схожості між ними. Вони рекомендують користувачам книги, що схожі на ті, які вони раніше обирали або читали. Ці системи відомі своєю простотою і здатністю ефективно працювати з новими книгами, які ще не набули популярності серед читачів. Однак, вони можуть не враховувати індивідуальні уподобання користувачів і обмежені у рекомендації творів, які сильно відрізняються від уже прочитаних.

Колаборативна фільтрація, з іншого боку, аналізує історію читання або покупок користувачів, щоб виявити зі схожими інтересами. Цей метод може рекомендувати книги, які сподобалися іншим користувачам з аналогічними уподобаннями, навіть якщо вони значно відрізняються від тих, що користувач вже читав. Основна проблема колаборативної фільтрації полягає у великій потребі в даних про користувачів та їхніх уподобаннях, а також у складнощах з

новими користувачами, для яких історія покупок або читання обмежена або відсутня.

Гібридні рекомендаційні системи комбінують елементи обох підходів, поєднуючи аналіз контенту з аналізом поведінки користувачів, що дозволяє врахувати як схожість книг, так і уподобання користувачів. Ці системи зазвичай забезпечують вищу точність рекомендацій та більш персоналізований досвід, але вони також є складнішими в розробці і потребують значно більше даних для ефективної роботи [10].

У власній розробці було вирішено застосувати принципи гібридних рекомендаційних систем для отримання рекомендацій вищої точності.

1.5 Постановка задач для розробки автоматизованої системи

Проаналізувавши питання розробки автоматизованих систем управління бібліотеками було визначено ряд задач, які необхідно виконати для розробки власної автоматизованої системи:

- здійснити аналіз стану питання та методів розв’язання задачі;
- спроектувати модель даних та архітектуру системи;
- розробити загальний алгоритм роботи системи;
- розробити метод обробки даних та формування рекомендацій;
- розробити метод взаємодії із чат-ботом для сповіщень;
- виконати програму реалізацію системи для управління роботою бібліотеки;
- провести тестування систем;
- розробити інструкцію користувача;
- розробити системні вимоги програмної системи.

Технічне завдання на розробку наведено в додатку А.

1.6 Висновки

У розділі було досліджено стан питання автоматизованих систем управління бібліотеками. Було проведено порівняльний аналіз систем аналогів:

Koha, FOLIO та Alma з власною розробкою. Також було описано можливі напрямки вдосконалення автоматизованої системи управління бібліотекою.

Було, також, розглянути можливі типи архітектур автоматизованих систем та методи формування рекомендацій та сповіщень для користувачів. Встановлено доцільність розробки системи за принципами мікросервісної архітектури та розроблено методи формування рекомендацій, що використовують широкий спектр параметрів для більшої точності рекомендацій. Було сформульовано задачі, які необхідно виконати для розробки системи.

2 РОЗРОБКА МОДЕЛІ, МЕТОДІВ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних та вихідних даних автоматизованої системи бібліотеки

Вхідні дані – це дані, які подаються до системи користувачем або іншим зовнішнім джерелом. Вони необхідні для виконання програми чи обробки в рамках певного процесу. Вихідні дані – це результати, які генерує програмне забезпечення після обробки вхідних даних. Вихідні дані також можуть мати різні форми і передаватись через різні канали.

Автоматизована система управління бібліотекою маніпулює вхідними даними, джерелом яких є безпосередньо користувач. Насамперед адміністратор чи технічний працівник закладу, що використовує автоматизовану систему, змушений авторизуватися у ній шляхом внесення особистих даних. Також користувач має змогу форматувати базу даних книг шляхом внесення усієї необхідної інформації, яка потім буде доступна не привілейованому користувача, інформація про книгу включає такі деталі як: назва книги, автор, код ISBN, короткий опис, посилання на титульне зображення книги та категорії до яких книга відноситься.

Також, не менш важливою є інформація, яка генерується під час процесу бронювання книги, оскільки не привілейований користувач має змогу забронювати книгу на певний термін шляхом вказання таких даних, як: ідентифікаційні номери книг, які користувач бажає забронювати та заплановану дату повернення, за рахунок цього система матиме змогу коректно опрацювати дані користувача та виконати свою роботу належним чином.

Вхідні дані користувача системи зберігаються у базах даних та чергах повідомлень. У процесі роботи системи усі дані проходять обробку залежно від їх типу та надсилаються користувачу або запускають інший процес, що призведе до додаткової обробки даних.

До вихідних даних автоматизованої системи належить інформація про книги та категорії, які були додані привілейованими користувачами, також, як вихідні дані слугують сповіщення користувачам, які відображаються у програмному інтерфейсі Telegram-боту.

2.2 Розробка архітектури автоматизованої системи

Розробка архітектури програмного забезпечення є одним із найважливіших етапів у створенні програмних продуктів, оскільки вона визначає основну структуру і організацію системи. Правильно підібрана архітектура сприяє кращому розумінню того, як компоненти програми взаємодіють між собою, роблячи код легшим для читання і управління. Також, вона дозволяє легко масштабувати продукт, адаптуючи його до зростання користувачів або обсягу даних.

Для розробки автоматизованої системи управління робою бібліотек було обрано мікросервісний підхід у побудові архітектури, оскільки цей підхід дозволяє розділити велику систему на вузькоспеціалізовані та незалежні компоненти, які легше розробляти, тестувати та розгортати окремо один від одного. Мікросервісна архітектура підвищує гнучкість системи, оскільки дозволяє внести зміни або оновлення до окремих частин системи без впливу на роботу всієї системи загалом. Це особливо важливо для бібліотечної системи, де можуть постійно з'являтися нові вимоги чи необхідність інтеграції з іншими інформаційними ресурсами. Завдяки мікросервісному підходу, кожен сервіс може бути оптимізований для виконання окремої функції, що підвищує загальну продуктивність та ефективність системи [11].

Для візуального представлення архітектури системи та зв’язків її компонентів (сервісів) було створено структурну діаграму, що зображена на рисунку 2.1. Діаграма відображає взаємодію основних сервісів системи таких як:

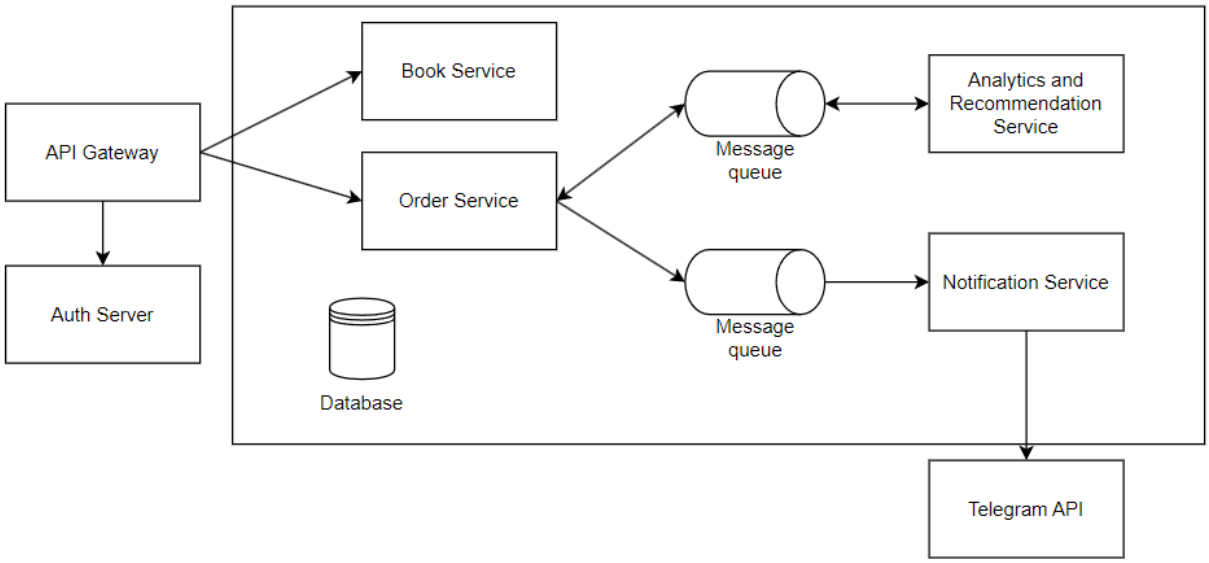

Рисунок 2.1 – Структурна діаграма автоматизованої системи

- Сервіс книг (Book Service) – це сервіс, що відповідає за додавання, редагуванням та видаленням книг та категорій, також цей сервіс визначає основні вимоги до формату даних про книги та його відображення у базі даних;

- Сервіс замовлень (Order Service) – це сервіс, що відповідає обробку запитів користувача для бронювання та повернення книг, цей сервіс є основним у системі оскільки делегує процеси для сервісів, що відповідають за сповіщення та формування аналітики та рекомендацій;

- Сервіс аналітики та рекомендацій (Analytics and Recommendation Service) – це сервіс, який призначений для обробки даних про дії користувача, а саме про сформовані бронювання та повернення;

- Сервіс сповіщень (Notification Service) – цей сервіс відповідає за взаємодію із Telegram API, він також визначає набір правил для чат-боту;

- API Gateway – це сервіс-посередник, що відповідає за делегування запитів користувачів на відповідні сервіси;

- Сервер автентифікації/авторизації (Auth Server) – це сервіс, що відповідає за автентифікацію та авторизацію користувачів системи.

- База даних (Database) – це база даних системи, яка відповідає за зберігання даних користувача, даних про книги та бронювання.

- Message queue – це черга повідомлень, що забезпечує взаємодію між сервісами.

Отже, було розглянуто загальну структурну діаграму системи, що дозволяє зрозуміти основні методи комунікації та схему взаємодії сервісів, що використовуватимуться при розробці.

2.3 Розробка моделі системи

Розробка моделі системи є невід'ємною частиною процесу створення програмного забезпечення, особливо для складних проектів, таких як система управління бібліотекою. Вона дозволяє наочно представити структуру, компоненти та взаємозв'язки системи, що допомагає розробникам, замовникам та іншим зацікавленим сторонам краще зрозуміти, як система буде працювати. Крім того, модель системи допомагає виявити потенційні проблеми та ризики на ранніх етапах розробки, що дозволяє уникнути дорогих помилок та переробок на пізніших стадіях проекту. Вона також служить єдиним джерелом інформації для всіх учасників проекту, полегшуючи комунікацію та спрощуючи процес розробки. Завдяки моделі система розбивається на більш прості та керовані компоненти, що дозволяє розробникам зосередитися на окремих частинах системи, не втрачаючи з поля зору загальну картину. Це, у свою чергу, сприяє підвищенню якості та надійності системи, оскільки модель допомагає забезпечити відповідність системи вимогам замовника та стандартам якості. У кінцевому підсумку, розробка моделі системи призводить до зменшення загальних витрат та скорочення часу розробки, оскільки дозволяє уникнути багатьох помилок та переробок.

Модель автоматизованої системи управління робою бібліотек з складається із ряду компонентів, таких як: інтерфейс користувача, клієнтська частина системи, серверна частина системи та база даних. Ці компоненти дозволяють забезпечити ефективну та відмовостійку працездатність системи.

Клієнтська частина системи відповідає за взаємодію користувача з інтерфейсом та візуалізацію даних, отриманих від сервера. Вона забезпечує:

1. Отримання та обробку вхідних даних. Клієнтська частина збирає дані, введені користувачем через інтерфейс (дані для авторизації, інформацію для бронювання), перевіряє їх на коректність та відправляє запити до сервера.

2. Відображення даних. Після отримання відповіді від сервера, клієнтська частина обробляє отримані дані та відображає їх користувачеві у зручному та зрозумілому вигляді.

Серверна частина відповідає за ряд завдань за переадресацію запитів між відповідними сервісами за допомогою шаблонного рішення «Discovery Service» [12], обробку даних користувача (авторизацію та автентифікацію), обробку даних про книги та формування персоналізованих рекомендацій користувача, формування сповіщень для користувача через прикладний програмний інтерфейс Telegram.

Модель системи може бути представлена за допомогою схеми, яку зображено на рисунку 2.2.

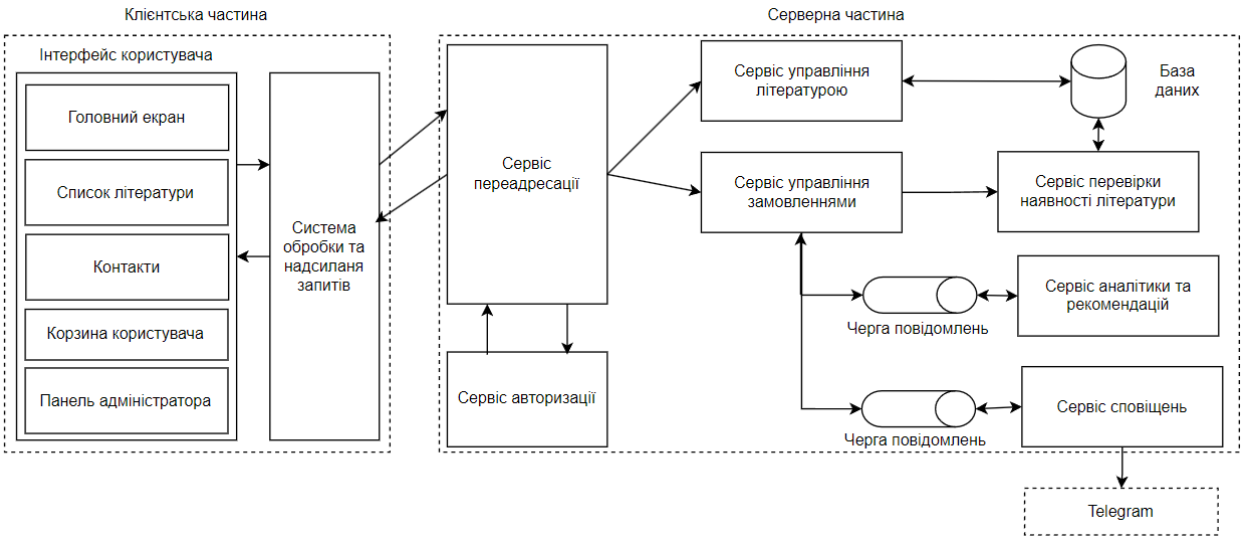

Рисунок 2.2 – Зображення системи

Як висновок, у підрозділі було описано розробку моделі системи, вказавши її основні компоненти та зобразивши схему, що представляє розширені концепції у архітектурі програмного застосунку.

2.4 Розробка загального алгоритму роботи системи

Розробка загального алгоритму системи є фундаментальним кроком у створенні автоматизованої системи управління бібліотекою, оскільки він визначає логіку її роботи та послідовність взаємодії компонентів.

Алгоритм описує, як система реагує на різні події та запити, як дані передаються між компонентами, які обчислення виконуються та які результати очікуються. Це не лише полегшує процес розробки та тестування, а й підвищує якість та надійність системи, оскільки дозволяє врахувати всі можливі сценарії використання та уникнути помилок. Крім того, чітко визначений алгоритм спрощує супроводження та модернізацію системи, а також слугує важливою частиною документації проекту.

Для автоматизованої системи управління бібліотекою було розроблено загальний алгоритм роботи системи (див. рис. 2.3 та рис .2.4).

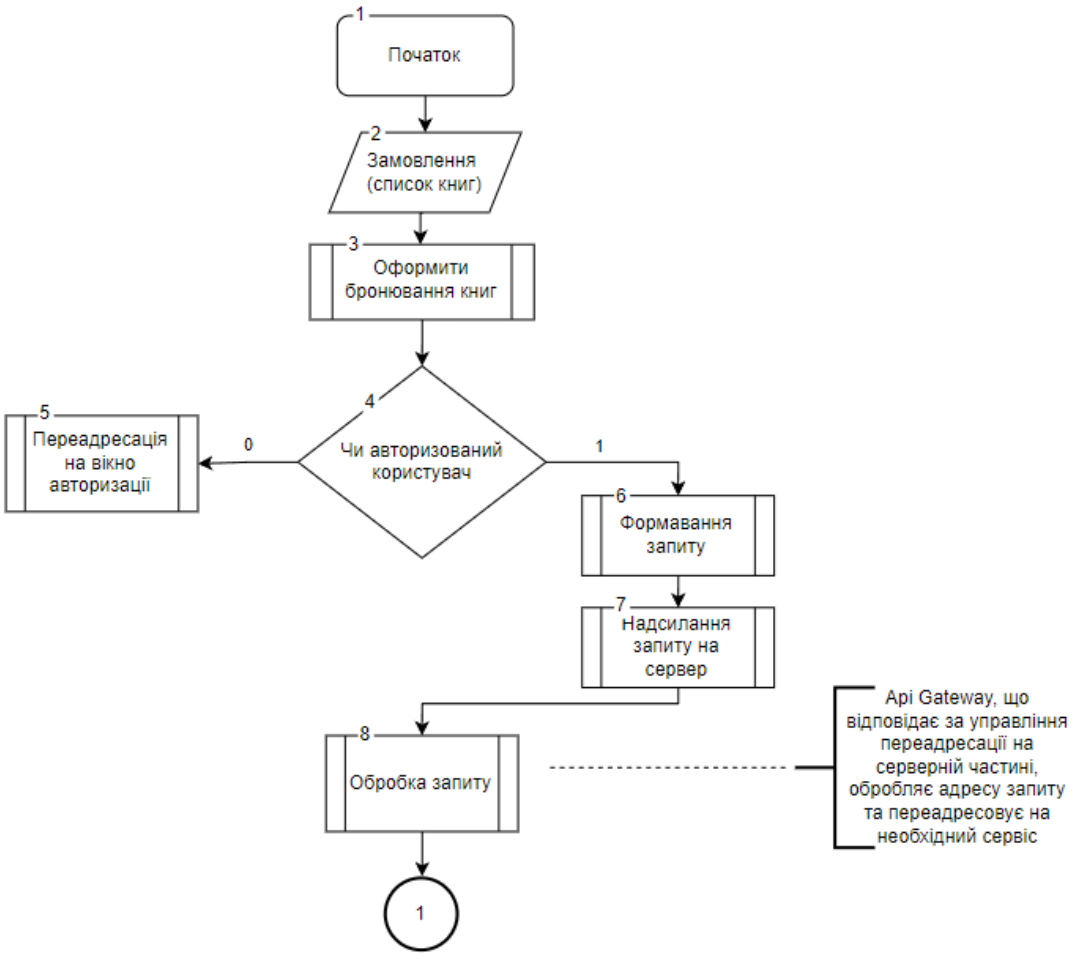

Рисунок 2.3 – Зображення загального алгоритму системи №1

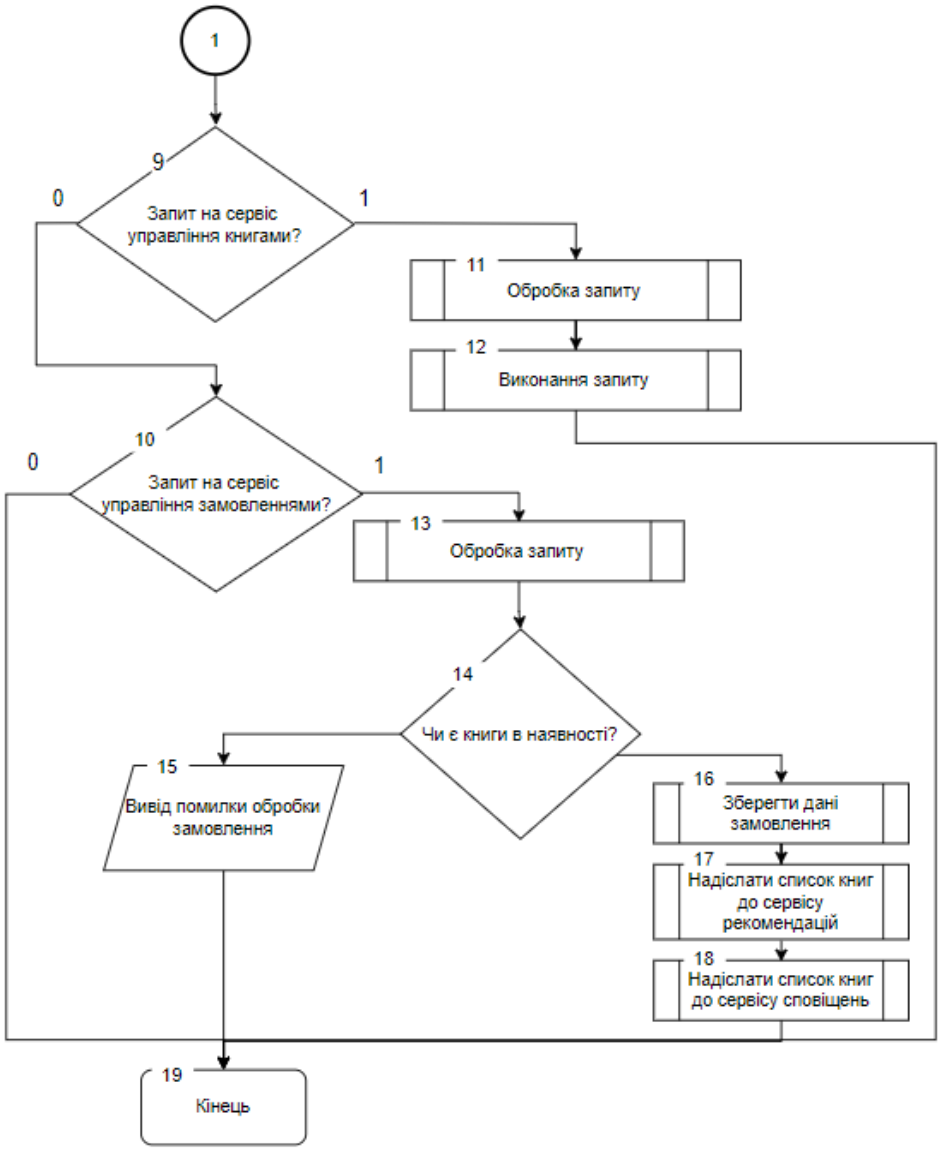

Рисунок 2.4 – Зображення загального алгоритму системи №2

Опис кроків загального алгоритму роботи системи:

1. Початок блок-схеми.

2. Вхідні дані алгоритму – це замовлення, що являє собою список обраних книг користувачем.

3. Оформити бронювання книг: клієнтська частина зберігає обрані книги користувача протягом сеансу відвідування веб сайту та після підтвердження замовлення клієнтська частина сформує запит до серверної частини у вигляду посилання.

4. Перевірка чи авторизований користувач: клієнтська частина перевіряє чи локальне сховище браузера (local storage) містить токен авторизації користувача.

5. Переадресація на вікно авторизації: у разі не виконання попередньої умови, користувача буде переадресоване на вікно авторизації/реєстрації, після чого користувач зможе продовжити формування замовлення.

6. Формування запиту: у разі виконання умови 4, клієнтська частина сформує запит із використання токену авторизації користувача.

7. Надсилання запиту на сервер: клієнтська частина надішле запит на ендпоінт серверної частини, а саме «Discovery service» (сервіс, що відповідає за делегування сервісами).

8. Обробка запиту: «Discovery service» отримує запит за допомогою контролерів та переадресовує його на необхідний сервіс.

9. Запит на сервіс управління книгами: умовне позначення процесу вибору необхідного сервісу.

10. Запит на сервіс управління замовленнями: умовне позначення процесу вибору необхідного сервісу.

11. Обробка запиту: у разі переадресації на сервіс управління книгами, запит буде прийнятий контролером та оброблений відповідно до початкових вимог користувача.

12. Виконання запиту: виконання запиту на стороні сервісу управління книгами.

13. Обробка запиту: у разі переадресації на сервіс управління замовленнями, запит буде прийнятий контролером та оброблений відповідно до початкових вимог користувача.

14. Перевірка наявності книг: сервіс управління замовленнями виконає запит на сервіс перевірки наявності книг (сервіс перевірки наявності книг відповідає за облік даних про книги, що зберігаються у базі даних системи).

15. Вивід помилки обробки замовлення: у разі не виконання пункту 14, сервіс управління замовленнями надішле відповідь на клієнтську частину із кодом помилки.

16. Зберегти дані замовлення: у разі виконання пункту 14, система управління замовлення збереже дані замовлення в базу даних, для подальшої можливості відтворення історії.

17. Надіслати список книг до сервісу рекомендацій: система управління замовленнями «опублікує» поточне замовлення у чергу повідомлень, що стосується сервісу, що відповідає за формування рекомендацій.

18. Надіслати список книг до сервісу сповіщень: системи управління замовленнями «опублікує» поточне замовлення у чергу повідомлень, що стосується сервісу сповіщень.

19. Кінець роботи загального алгоритму.

Отже, було у цьому підрозділі було розглянуто та описано загальний алгоритм роботи програмної системи, а саме процес взаємодії користувача із системою.

Отже, у цьому підрозділі було обґрунтовано актуальнісь розробки загального алгоритм системи та описано власний загальний алгоритм роботи для розроблюваної системи.

2.5 Розробка методу обробки даних та формування рекомендацій

Розробка методу обробки даних та формування рекомендацій є ключовим аспектом створення сучасної та ефективної автоматизованої системи управління бібліотекою. Цей алгоритм відіграє важливу роль у забезпеченні персоналізованого та зручного користувацького досвіду, а також у оптимізації роботи бібліотеки.

Важливість методу обробки даних:

1. Ефективне управління даними. Алгоритм обробки даних дозволяє автоматизувати збір, зберігання, аналіз та оновлення інформації про книги, користувачів, бронювання та інші аспекти роботи бібліотеки. Це зменшує ризик

помилок, пов'язаних з ручною обробкою даних, та забезпечує актуальність інформації.

2. Пошук та фільтрація. Алгоритм дозволяє користувачам швидко та зручно знаходити потрібні книги за різними критеріями (назва, автор, жанр, ключові слова тощо). Він також може враховувати історію пошуку та вподобання користувача для надання більш релевантних результатів.

3. Аналіз даних. Алгоритм може аналізувати дані про користувачів та їхню взаємодію з системою, щоб виявляти тенденції, популярні книги, затребувані жанри та інші корисні відомості. Ця інформація може бути використана для покращення роботи бібліотеки, формування колекції та планування заходів.

Важливість методу формування рекомендацій:

1. Персоналізація. Методи формування рекомендацій дозволяють надавати користувачам персоналізовані рекомендації щодо книг, які можуть їм сподобатися. Це підвищує задоволеність користувачів та стимулює їх до активнішого використання бібліотеки;

2. Підвищення зацікавленості. Рекомендації можуть допомогти користувачам відкрити для себе нові книги та авторів, розширити свій кругозір та підвищити інтерес до читання;

3. Оптимізація роботи бібліотеки. Аналіз даних та формування рекомендацій дозволяють бібліотеці краще розуміти потреби своїх користувачів та пропонувати їм найбільш релевантні послуги. Це сприяє підвищенню ефективності роботи бібліотеки та збільшенню її популярності.

Розроблюваний метод для обробки даних та формування рекомендацій, що запроваджений у системи описує такі дії:

1. Збір даних. Реалізація сховища даних – використовуємо Spring Data JPA для роботи з базою даних (MySQL) для зберігання явних та неявних даних про користувачів та книги. Парсинг даних про книги: Використовуємо бібліотеки для парсингу даних з описів книг.

2. Обробка та аналіз даних. Контентна фільтрація – використовуємо бібліотеки Java для обробки природної мови для виділення ключових слів та тем з описів книг, також використовуємо дані з історії замовлень користувача.

3. Формування рекомендацій. Ранжування – сортуємо список рекомендацій за релевантністю, використовуючи різні метрики (наприклад популярність).

4. Відображення рекомендацій. За допомогою черги повідомлень надсилаємо повідомлення до сервісу обробки замовлень, який в подальшому переадресує дані на сервіс сповіщень, який у свою чергу надішле повідомлення із рекомендаціями користувачу в Telegram.

Як результат дослідження, було розроблено власний метод обробки даних та формування рекомендацій, що використовує широкий спектр параметрів, таких як: автори книг, жанри та історія попередніх замовлень користувача

2.6 Розробка методу взаємодії із чат-ботом

Розробка методу взаємодії системи управління бібліотекою з Telegram-ботом має велике значення для покращення користувацького досвіду, розширення функціональності та підвищення ефективності роботи бібліотеки. Telegram є популярним месенджером, який використовується багатьма користувачами. Інтеграція з Telegram-ботом дозволяє читачам отримувати доступ до послуг бібліотеки безпосередньо зі свого смартфона, що значно спрощує взаємодію та робить її більш зручною.

Покроковий опис методу взаємодії з чат-ботом описано далі.

1. Ініціалізація бота:

- реєстрація бота в Telegram через BotFather та отримання токену доступу;

- налаштування вебхуків для отримання оновлень від Telegram;

- створення обробника оновлень на сервері, який буде реагувати на повідомлення від користувачів.

2. Авторизація користувача:

- користувач надсилає будь-яке повідомлення боту;

- бот отримує оновлення від Telegram про нове повідомлення;

- обробник оновлень на сервері перевіряє, чи є користувач вже зареєстрованим у системі.

3. Надсилання сповіщень:

- система отримує повідомлення з черги повідомлень, що надсилається від системи управління замовленнями та генерує сповіщення про книги (коли термін бронювання книги вичерпаний або коли з'явилися нові рекомендації);

- обробник сповіщень на сервері отримує ці сповіщення;

- обробник знаходить у базі даних Telegram ID користувачів, які мають отримати сповіщення;

- бот надсилає сповіщення користувачам через Telegram API.

4. Обробка команд користувача:

- Користувач може надсилати боту команди (наприклад, /start, /help, /settings);

- обробник оновлень розпізнає ці команди та виконує відповідні дії (наприклад, відображає інформацію про бота, налаштування сповіщень тощо).

Отже, у підрозділі було детально описано метод взаємодії системи із Telegram ботом та обґрунтовано переваги застосування у системі.

2.7 Проєктування макетів графічного інтерфейсу користувача

Інтерфейс користувача відповідає за взаємодію користувачів з системою. У проекті автоматизованої системи управління бібліотекою, інтерфейс користувача виконує такі ключові функції: візуалізація інформації, формування макету для створення бронювань та авторизації користувача.

Візуалізація інформації. Інтерфейс відображає всю необхідну інформацію про книги, включаючи назву, автора, жанр. Це дозволяє користувачам швидко та зручно знаходити потрібні книги. Для кращого представлення компонентів, що

відповідають за візуалізацію даних, було розроблено схему, що зображена на рисунку 2.5.

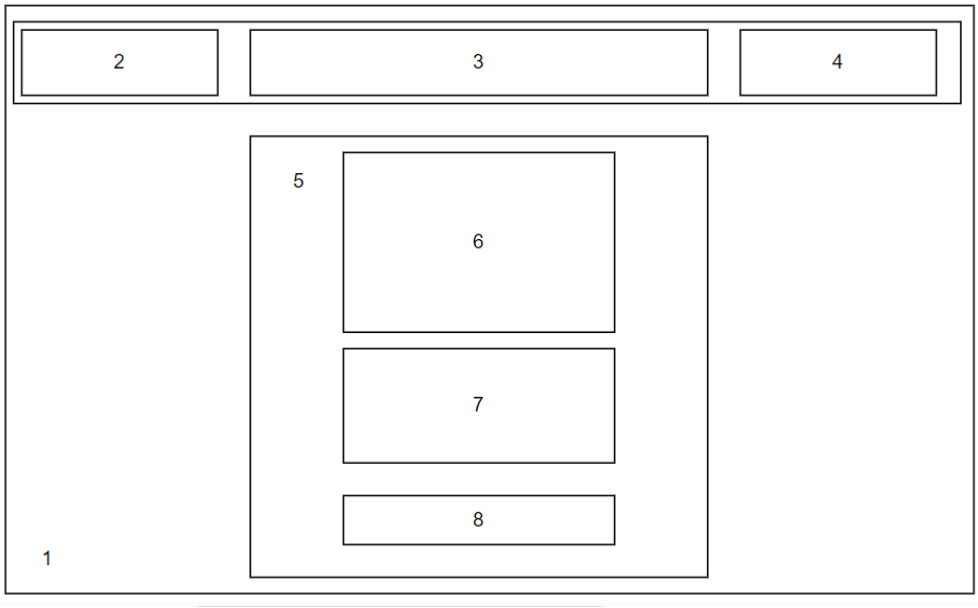

Рисунок 2.5 – Зображення схеми побудови екрану інтерфейсу користувача, що відповідає за відображення списку книг

Опис усіх елементів схеми:

1. Головний екран сторінки.

2. Логотип програмного застосунку (у реалізованому варіанті представлений, як «Книжка» + «Система управління бібліотекою»).

3. Навігаційна панель користувача, що включатиме наступні елементи: головна сторінка, сторінка списку літератури та сторінка, що відповідає за контакти розробника.

4. Додаткові функціональні елементи – це елементи, що відповідають за авторизацію користувача, можливість зміни теми веб сайту (можливі варіанти тем: день, ніч), функціональний елемент корзини замовлення та елемент переходу до сторінки адміністратора.

5. Стандартизований контейнер подання інформації про книгу, він використовується для налаштування подання інформації про книги структуровано списком чи у табличному вигляді.

6. Зображення книги.

7. Інформація про книгу, секція інформації про книгу міститиме таку інформацію як: назва книги, автор, короткий опис та жанр.

8. Функціональний елемент «Додати книгу» - це кнопка, що дозволяє додати книгу до списку замовлення.

Створення бронювань. Інтерфейс дозволяє користувачам створювати бронювання на книги, які наразі недоступні. Це забезпечує зручність для читачів, оскільки вони можуть бути впевнені, що отримають потрібну книгу, коли вона стане доступною.

Авторизація користувачів. Інтерфейс забезпечує можливість користувачам авторизуватися в системі, використовуючи свої облікові дані (логін та пароль). Це дозволяє відстежувати історію користування кожного читача, надавати персоналізовані рекомендації та контролювати доступ до певних функцій системи, таких як створення бронювань та використання панелі адміністратора. Для кращого представлення компонентів, що відповідають за авторизацію користувачів, було розроблено схему, що зображена на рисунках 2.6 та 2.7.

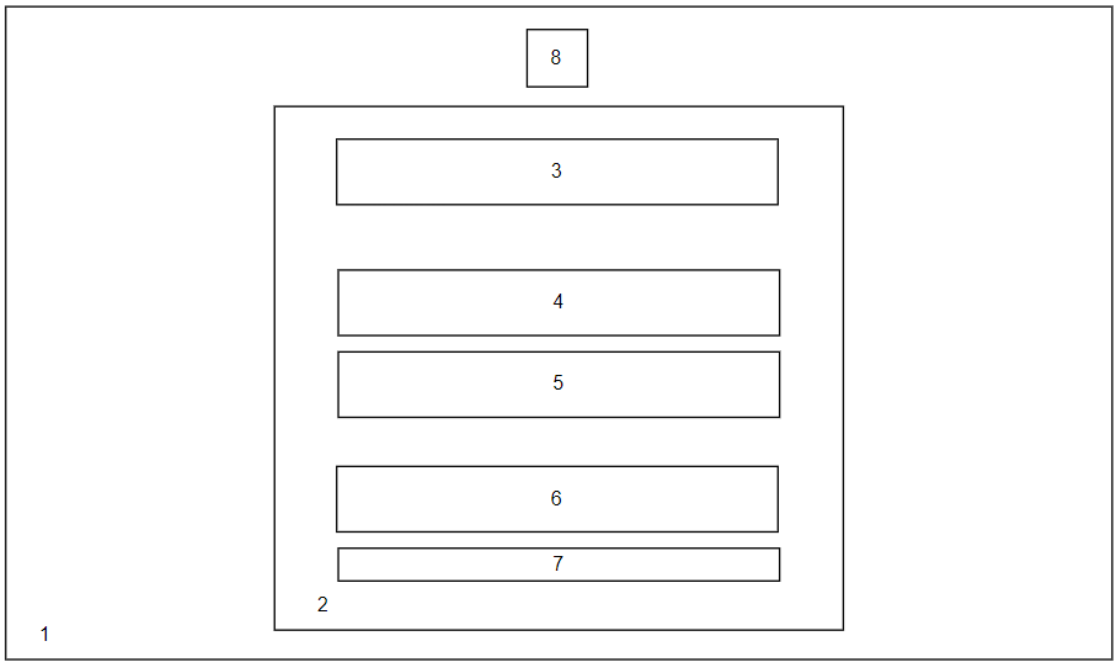

Рисунок 2.6 – Зображення схеми побудови вікна входу в існуючий аккаунт

Опис усіх елементів схеми:

1. Головний екран.
2. Контейнер, що відповідає за елементи вікна авторизації.
3. Елемент заголовку, що вказуватиме на призначення вікна авторизації.
4. Поле вводу даних електронної пошти користувача для авторизації.
5. Поле вводу даних паролю користувача для авторизації.
6. Секція для переадресації на реєстрацію користувача.
7. Функціональний елемент входу в обліковий запис.
8. Функціональний елементи закриття вікна авторизації

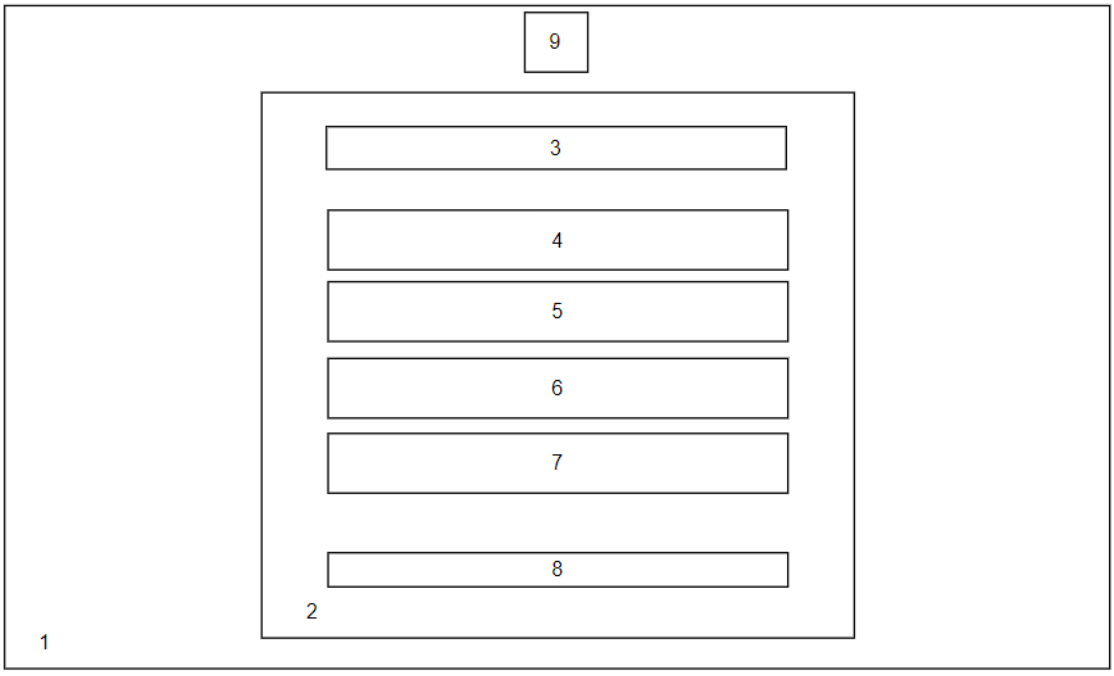

Рисунок 2.7 – Зображення схеми побудови вікна реєстрації нового користувача

Опис усіх елементів схеми:

1. Головний екран.
2. Контейнер, що відповідає за елементи вікна авторизації.
3. Елемент заголовку, що вказуватиме на призначення вікна реєстрації.
4. Поле вводу даних електронної пошти користувача для реєстрації.
5. Поле вводу даних номера телефону користувача для реєстрації.

6. Поле вводу даних імена та прізвища користувача для реєстрації.

7. Поле вводу даних пароля користувача для реєстрації.

8. Функціональний елемент входу в обліковий запис.

9. Функціональний елементи закриття вікна авторизації.

Отже, у підрозділі було описано особливості розробки графічного інтерфейсу користувача для системи та зображено приклади схематичного позиціювання графічних компонентів.

2.8 Висновки

У розділі було проаналізовано вхідні та вихідні дані автоматизованої системи для управління бібліотеками, розроблено архітектуру системи. Також було описано модель системи, загальний алгоритм роботи, розроблено методи обробки даних та формування рекомендацій, метод взаємодії із чат-ботом та розробку макетів графічного інтерфейсу користувача. Для кращого детальнішого аналізу архітектури та алгоритмів системи було розроблено ряд UML-діаграм.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Вибір мови програмування для розробки автоматизованих систем управління бібліотеками – це важливий етап, який визначає подальші можливості та характеристики проекту. Кожна мова має свої особливості та переваги, які важливо врахувати при прийнятті рішення. Для варіантного аналізу було обрано найпопулярніші мови розробки автоматизованих веб систем: Java, C# та Python.

Java, завдяки своїй платформонезалежності, є однією з найбільш популярних мов програмування для розробки автоматизованих систем управління бібліотеками. Вона має широку підтримку середовищ розробки, таких як IntelliJ IDEA та Eclipse, а також велику спільноту розробників і великий набір інструментів і бібліотек, зокрема фреймворк Spring, який полегшує розробку великих та складних систем. Крім того, Java відома своєю високою надійністю, безпекою та масштабованістю, що робить її ідеальним вибором для великих проектів [13].

C# – мова програмування, що підтримується платформою .NET і часто використовується для розробки веб-додатків під Windows з використанням фреймворка ASP.NET. C# має високу продуктивність та широкий набір інструментів, а також підтримує багато модерних функцій програмування, таких як асинхронність та паралельне програмування [14]. Однак, вона має обмежену підтримку для платформ, що не є Windows, і її використання може бути більш обмеженим у порівнянні з Java.

Python, з іншого боку, славиться своєю простотою та лаконічністю коду, що робить його привабливим вибором для швидкої розробки прототипів та простих веб-додатків. Вона має великий екосистему бібліотек та фреймворків, таких як Django та Flask, що сприяє розробці веб-додатків [15]. Однак, Python може мати меншу продуктивність порівняно з Java на великих обсягах даних або

великих системах, і він може не мати такої ж широкої підтримки для підприємницького середовища, як Java.

Порівняльний аналіз мов програмування наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування

Критерій	Java	C#	Python
Паралелізм та багатопоточність	+	+	+
Платформонезалежність	+	-	+
Підтримка інструментів та бібліотек	+	+	+
Спільнота розробників	+	+	+
Надійність та безпека	+	+	+
Масштабованість	+	+	-
Підсумок	6	5	5

Також, важливо звернути увагу на середовища розробки, оскільки інтегровані середовища розробки (IDE) відіграють ключову роль у розробці програмного забезпечення, надаючи розробникам потужні інструменти для написання, тестування та розгортання додатків. Для розробки на Java, зокрема використовуючи фреймворк Spring, є кілька популярних IDE, які можуть бути використані:

1. Eclipse – це одне з найдавніших і найбільш розповсюджених середовищ розробки, яке підтримує багато мов програмування, включаючи Java. Для розробки на Spring, Eclipse пропонує численні плагіни, такі як Spring Tools 4, які полегшують розробку і тестування програм. Eclipse відомий своїм гнучким налаштуванням та широкими можливостями інтеграції, що робить його вибором для багатьох розробників, які цінують стабільність і розширені функції [16].

2. IntelliJ IDEA – це середовище розробки від JetBrains, яка включає розширену підтримку для веб-розробки та мобільних додатків. Для Java і Spring, IntelliJ IDEA Ultimate пропонує вбудовану підтримку фреймворку Spring, включаючи автоматичне конфігурування проектів, інтелектуальне виявлення

багів і продуктивність розробки завдяки інтеграції з базами даних і веб-сервісами. Це середовище є вибором для розробників, які вимагають високої продуктивності та інтеграції з додатковими мовами програмування та інструментами [17].

3. VSCode – це легке, але потужне IDE, яке стає все більш популярним серед розробників Java. Хоча воно не надає стільки функцій "з коробки" як Eclipse або IntelliJ, його можна значно розширити за допомогою плагінів. Для Spring розробки можна використовувати розширення, такі як Spring Boot Extension Pack, яке допомагає в управлінні залежностями, створенні та виконанні Spring Boot додатків [18].

Серед Eclipse, IntelliJ IDEA та VSCode, IntelliJ IDEA виділяється рядом переваг, що роблять її оптимальним вибором для розробки автоматизованої системи управління бібліотекою на Java з використанням Spring Framework:

1. Орієнтованість на підтримку Java та Spring. IntelliJ IDEA розроблена компанією JetBrains, яка також створила Spring Framework. Це забезпечує найглибшу та найповнішу інтеграцію з Spring серед усіх IDE. Розробники отримують доступ до інтелектуального автодоповнення, рефакторингу, навігації, перевірки коду та інших функцій, спеціально оптимізованих для Spring.

2. Висока продуктивність та швидкість. IntelliJ IDEA відома своєю швидкою роботою та ефективністю, навіть при роботі з великими проектами. Вона використовує індексацію коду та інші оптимізації, що дозволяє швидко реагувати на дії розробника, скорочує час компіляції та пришвидшує запуск додатків.

3. Потужні інструменти відлагодження та тестування. IntelliJ IDEA надає розробникам широкий спектр інструментів для відлагодження та тестування коду, включаючи інтеграцію з популярними фреймворками тестування. Це допомагає швидко виявляти та виправляти помилки, забезпечуючи високу якість коду та надійність системи.

4. Ергономічний та налаштовуваний інтерфейс. IntelliJ IDEA має інтуїтивно зрозумілий та зручний інтерфейс, який можна легко налаштувати під індивідуальні потреби. Це дозволяє розробникам зосередитися на коді, а не на інструментах, що підвищує продуктивність та комфорт роботи.

5. Широка екосистема плагінів. IntelliJ IDEA підтримує велику кількість плагінів, які розширюють її функціональність та дозволяють інтегрувати її з іншими інструментами та технологіями. Це робить її універсальним середовищем розробки, яке можна адаптувати під будь-які потреби проекту.

Хоча IntelliJ IDEA є комерційним продуктом, її переваги в плані продуктивності, функціональності та зручності використання роблять її вигідною інвестицією для розробки складних та відповідальних проектів, таких як автоматизована система управління бібліотекою. Отже, для розробки автоматизованої системи було вирішено використовувати мову програмування Java та середовище розробки IntelliJ IDEA, порівнявши усі переваги та недоліки обраних технологій та засобів.

3.2 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача (UI) є критично важливим етапом у створенні будь-якої програмної системи, включаючи автоматизовану систему управління бібліотекою. UI виступає мостом між користувачем та функціональністю системи, забезпечуючи ефективну та приємну взаємодію.

Ось основні причини, чому розробка UI є важливою [19]:

1. Перше враження та залучення користувачів. Інтерфейс користувача - це перше, що бачить користувач при взаємодії з системою. Зручний, інтуїтивно зрозумілий та естетично привабливий UI створює позитивне перше враження та заохочує користувачів до подальшої взаємодії.

2. Зручність використання та ефективність. Добре продуманий UI дозволяє користувачам легко та швидко знаходити потрібну інформацію, виконувати необхідні дії та досягати своїх цілей. Це підвищує ефективність роботи з системою та задоволеність користувачів.

3. Доступність. UI повинен бути доступним для всіх користувачів, включаючи людей з обмеженими можливостями. Це означає, що він повинен бути зрозумілим та зручним у використанні для людей з різними рівнями комп'ютерної грамотності, віком, культурним походженням та фізичними можливостями.

4. Підвищення продуктивності. Зручний та ефективний UI дозволяє користувачам швидше та легше виконувати свої завдання, що підвищує їхню продуктивність та ефективність роботи.

5. Зниження помилок. Інтуїтивно зрозумілий UI допомагає користувачам уникати помилок при виконанні дій, що підвищує надійність та безпеку системи.

6. Підвищення лояльності користувачів Позитивний досвід взаємодії з системою завдяки зручному UI сприяє формуванню лояльності користувачів та їхньому бажанню продовжувати користуватися системою.

Для автоматизованої системи управління робою бібліотек було розроблено наступні компоненти інтерфейсу користувача: головний екран, екран списку літератури та екран контактів розробника. Інтерфейс користувача системи автоматизації управління бібліотеки було реалізовано за допомогою технологій HTML, CSS та Javascript. Оскільки HTML, CSS та JavaScript є ідеальним поєднанням для створення інтерфейсу користувача автоматизованої системи управління бібліотекою завдяки своїй універсальності, доступності та гнучкості. Ці стандартні веб-технології підтримуються всіма сучасними браузерами, що забезпечує кросплатформеність та доступність системи для користувачів з різних пристроїв. Користувачам не потрібно встановлювати додаткове програмне забезпечення, що робить систему більш зручною. HTML дозволяє створювати структуру сторінок, CSS відповідає за їх візуальне оформлення, а JavaScript забезпечує інтерактивність та динамічну поведінку [20]. Разом вони дозволяють створювати адаптивні інтерфейси, які підлаштовуються під різні розміри екранів та пристроїв.

Головний екран користувача – це центральний елемент інтерфейсу, який надає доступ до основних функцій системи. Головний екран користувача системи управління бібліотекою зображено на рисунку 3.1 та 3.2.

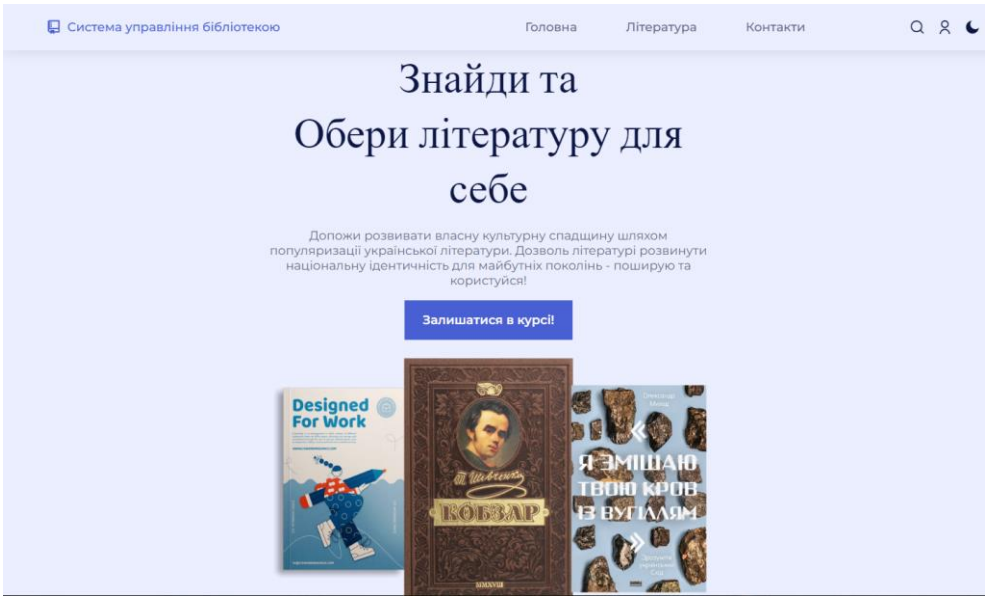

Рисунок 3.1 – Зображення головного екрану системи частина 1

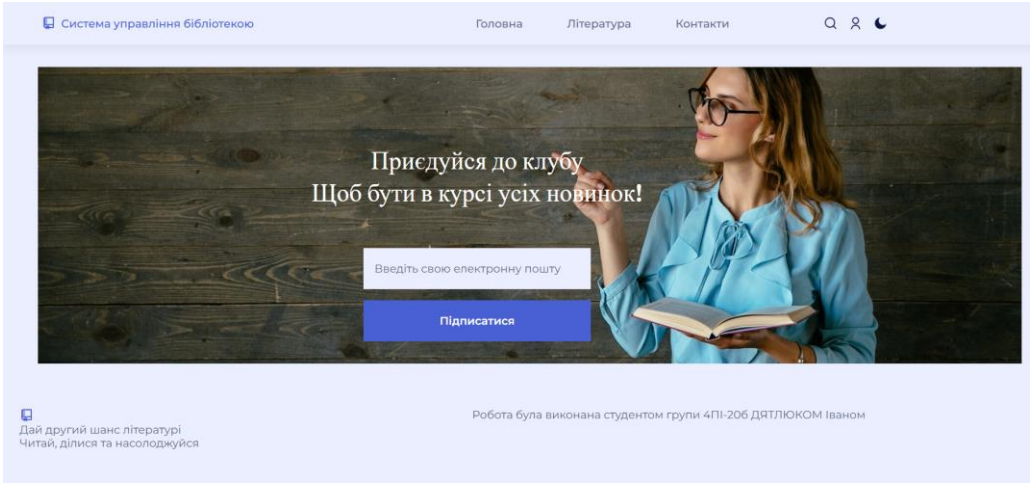

Рисунок 3.2 – Зображення головного екрану системи частина 2

Ця веб-сторінка є головною сторінкою системи управління бібліотекою. Вона містить основне меню з посиланнями на інші розділи сайту, такі як "Головна", "Література" та "Контакти". На головній сторінці банер з гаслом бібліотеки та кнопкою "Залишатися в курсі!". Нижче розташований слайдер з зображеннями книг, який демонструє новинки або популярні видання. Також є

розділ для приєднання до клубу, де користувачі можуть залишити свою електронну пошту для отримання новин. Сторінка має входу та реєстрації користувачів, а також можливість зміни теми оформлення (світла/темна) (див. рис. 3.3). Ці функції реалізовані за допомогою JavaScript, який забезпечує інтерактивність сторінки. В цілому, сторінка має простий та зрозумілий дизайн, фокусуючись на основних функціях системи управління бібліотекою.

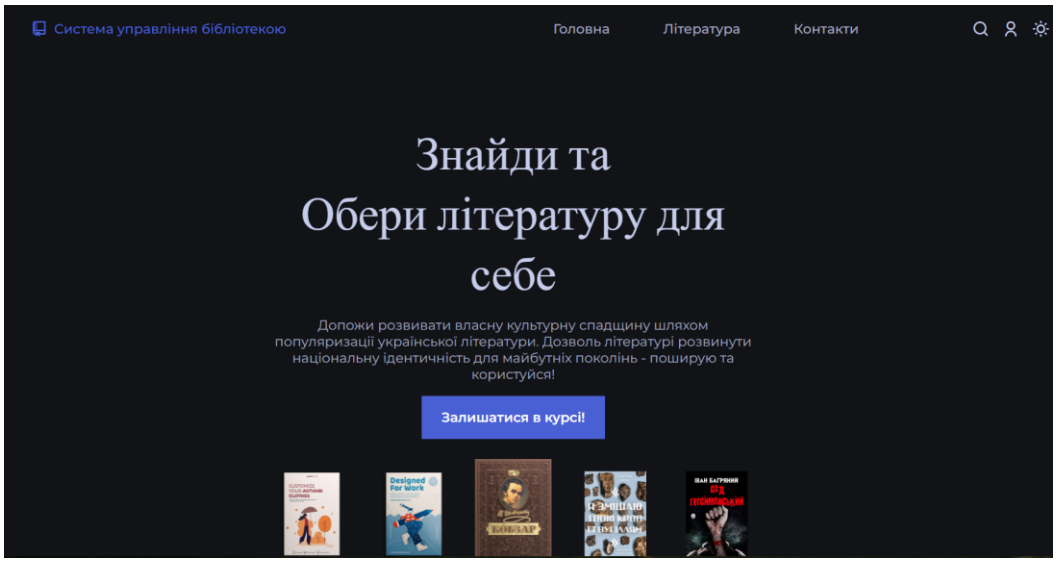

Рисунок 3.3 – Зображення головного екрану з використанням темної теми

Екран списку літератури – це екран, що надає користувачеві усю необхідну та актуальну інформацію про наявні книги, на цьому екрані зображено список усіх доступних книг із додатковою інформацією.

Ця веб-сторінка є каталогом книг в автоматизованій системі управління бібліотекою. У верхній частині сторінки розташовано логотип бібліотеки, меню для навігації по сайту (з посиланнями на головну сторінку, список літератури та контакти), а також іконки для пошуку, перегляду кошика, входу/реєстрації та зміни теми оформлення сайту.

Основна частина сторінки містить заголовок "Список літератури" та список книг у вигляді карток. Кожна картка містить зображення обкладинки книги, її назву, автора та короткий опис. Також є кнопка "Додати", яка дозволяє

користувачам додавати книгу до свого кошика для подальшого замовлення (див. рис. 3.4).

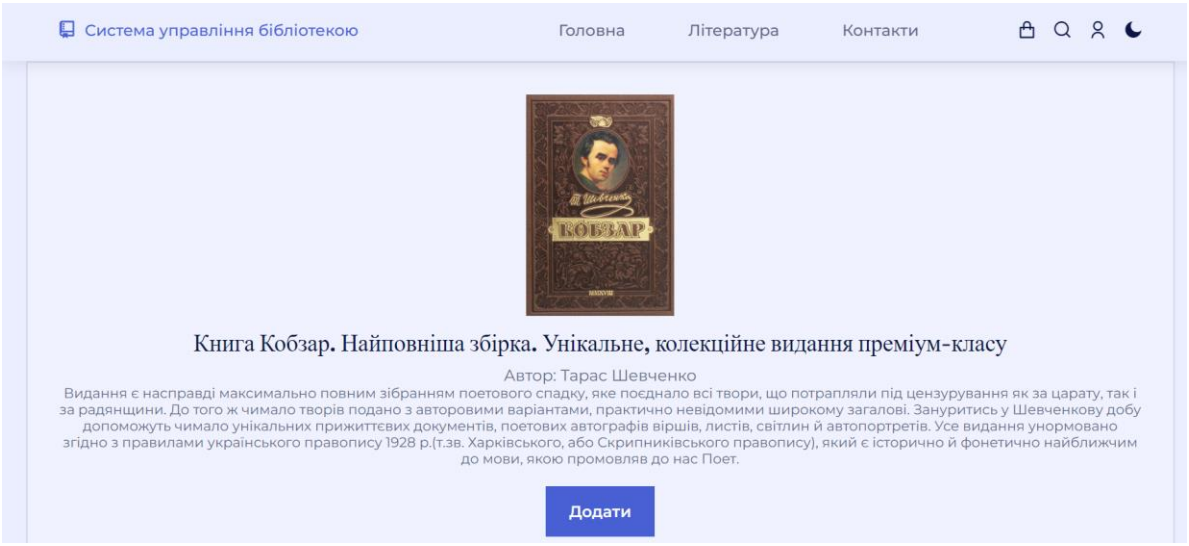

Рисунок 3.4 – Зображення сторінки «Література»

Екран «Контакти» сторінка, що надає інформацію про розробника системи управління бібліотекою, включаючи його ім'я, групу, короткий опис проекту та контактну інформацію. Сторінка також містить стандартні елементи, такі як шапка, пошук, форми авторизації та реєстрації, та підвал, що забезпечує зручну навігацію по сайту та доступ до основних функцій (див. рис. 3.5).

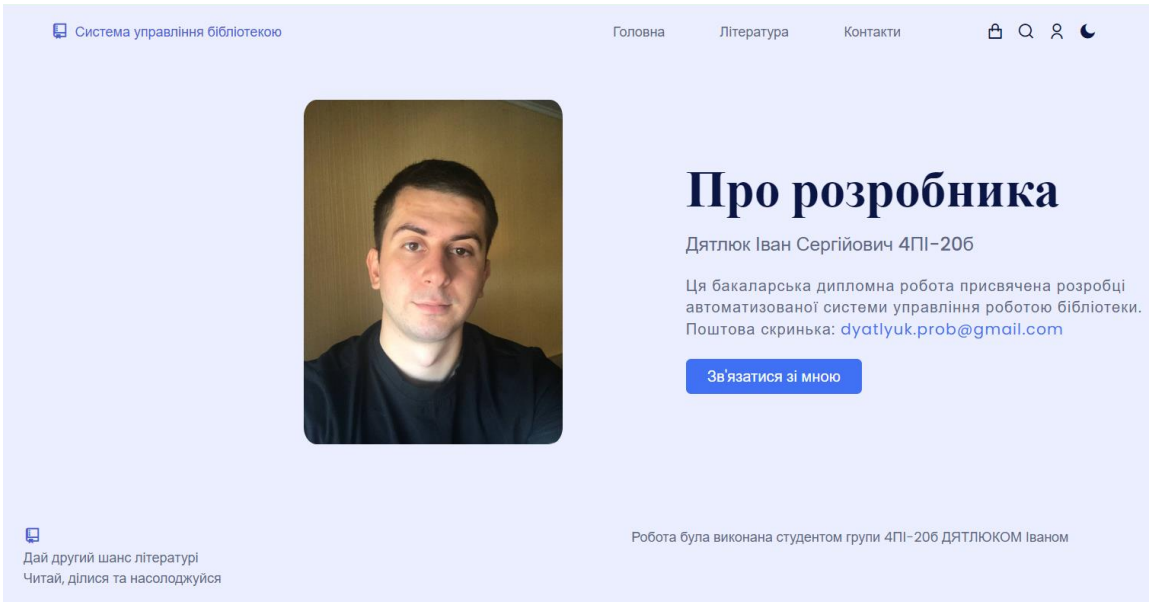

Рисунок 3.5 – Зображення екрану «Контакти»

Отже, у підрозділі роботи, було обґрунтовано розробку важливість розробки інтерфейсу користувача для системи управління робою бібліотеки та детально описано розроблюванні сторінки для системи.

3.3 Розробка сервісу обробки даних та формування рекомендацій

Розробка сервісу обробки даних та формування рекомендацій – це одне із основних завдань проєкту, що полягає у налаштуванні окремого сервісу, що відповідатиме за зчитування даних для подальшого формування персоналізованих рекомендацій.

Сервіс обробки даних та формування рекомендацій був розроблений з використанням Apache Kafka, потужної платформи обміну повідомленнями. До проекту Spring Boot була додана залежність spring-kafka, що надала інструменти для роботи з Kafka.

Була створена конфігурація споживача, де визначили ключові параметри, такі як адреси Kafka брокерів, унікальний ідентифікатор групи споживачів, а також правила для розшифровки ключів та значень повідомлень.

Після цього був створений сам споживач, який був підписаний на тему в Kafka, що в свою чергу отримувала повідомлення від сервісу управління замовленнями. Цей споживач почав отримувати повідомлення з цієї теми та обробляти їх відповідно до логіки сервісу. Отримавши повідомлення, споживач аналізує дані, формує персоналізовані рекомендації для користувачів та зберігає їх у базі даних. Ці рекомендації потім використовуються системою для надання користувачам релевантних пропозицій щодо книг. Щоб забезпечити високу продуктивність та масштабованість, споживач налаштований на роботу в групі споживачів. Це дозволяє розподілити навантаження між кількома екземплярами сервісу та забезпечити ефективну обробку великої кількості повідомлень.

Таким чином, завдяки інтеграції з Apache Kafka та ретельно продуманому алгоритму споживача, сервіс обробки даних та формування рекомендацій забезпечує користувачам бібліотеки персоналізований та зручний досвід взаємодії, пропонуючи їм книги, які відповідають їхнім інтересам та

вподобанням. Завдяки інтеграції з Apache Kafka та ретельній розробці алгоритму споживача, сервіс обробки даних та формування рекомендацій став невід'ємною частиною автоматизованої системи управління бібліотекою, забезпечуючи її користувачам персоналізований та зручний досвід взаємодії.

3.4 Розробка сервісу сповіщень

Сервіс сповіщень є важливою частиною автоматизованої системи управління бібліотекою, забезпечуючи зв'язок між системою та користувачами через Telegram. Його основна мета — своєчасно інформувати читачів про важливі події, такі як затримка повернення книг та надсилання персоналізованих рекомендацій.

Розробка цього сервісу включала наступні кроки:

Інтеграція з Telegram API. Для взаємодії з Telegram було використано Telegram Bot API, що дозволило створити бота та налаштувати його функціональність.

Детальні кроки налаштування інтеграції з Telegram API:

1. Відкрити спеціалізованого Telegram боту для створення власних ботів (див. рис. 3.6).

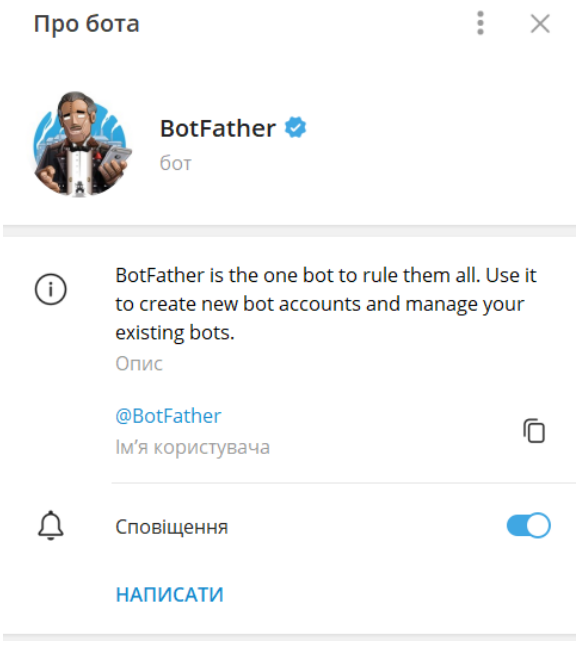

Рисунок 3.6 – Зображення інформації про бота

2. Натиснути на кнопку «start» для того, щоб розпочати роботу з чатом та отримати доступ до розширених можливостей (див. рис. 3.7).

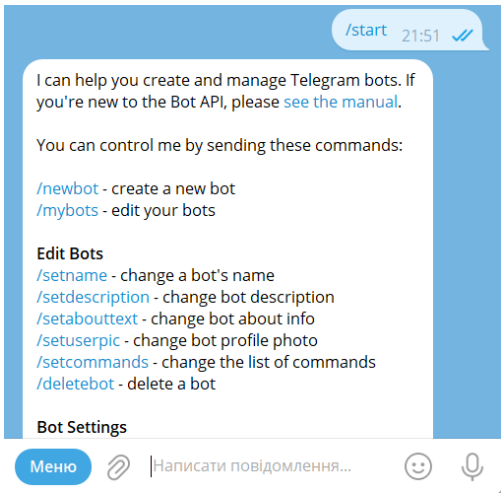

Рисунок 3.7 – Зображення діалогового вікна

3. Наступним кроком є вибір функції /newbot, після чого потрібно буле вказати локальну та глобальну назву бота. Як результат виконання попередніх дій, отримуємо усі необхідні дані для взаємодії бота із сервісом (див. рис. 3.8).

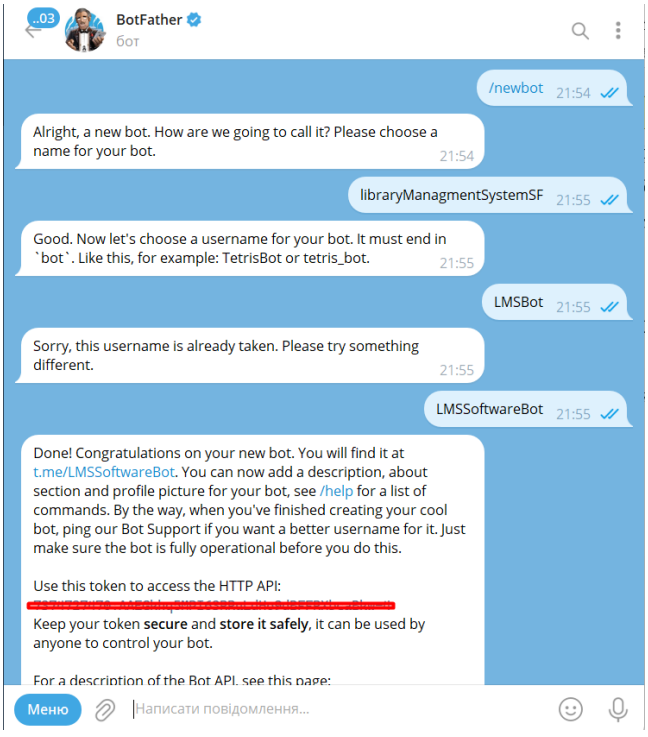

Рисунок 3.8 – Зображення повідомлення із генерацією коду доступу до створеного боту

Після налаштування боту для сповіщень на стороні Telegram, надається можливість реалізації власного бота і управління його логікою роботи, для цього необхідно створити відповідний клас, що реалізовуватиме базовий необхідний функціонал Telegram ботів.

Створення власного класу бота в системі управління бібліотекою дозволяє реалізувати необхідний функціонал та логіку роботи бота, адаптуючи його до специфічних потреб бібліотеки. Цей клас може містити такі методи:

Обробка команд:

- onStartCommand: обробляє команду /start, яка зазвичай використовується для ініціалізації бота та підписки користувача на сповіщення;

- onHelpCommand: обробляє команду /help, яка надає користувачеві інформацію про доступні команди та функції бота;

- onStopCommand: обробляє команду /stop, яка призупиняє роботу сповіщень для користувача.

Обробка повідомлень:

- onMessageReceived: обробляє текстові повідомлення від користувачів. Тут реалізований основний функціонал по формуванню відповіді користувачу;

- onCallbackQuery: обробляє натискання кнопок в інтерактивних повідомленнях.

Відправка повідомлень:

- sendMessage: відправляє текстове повідомлення користувачеві.

Наступними етапами розробки сервісу є налаштування підключення до Kafka серверу та налаштування отримання повідомлень.

Для отримання повідомлень від сервісу управління замовленнями було налаштовано підключення до Apache Kafka. Використано бібліотеку spring-kafka, яка надає зручні інструменти для роботи з Kafka в Spring Boot.

Для конфігурації споживача було створено клас конфігурації KafkaConsumerConfig, де визначено:

- адреси Kafka брокерів (bootstrap.servers);

- ідентифікатор групи споживачів (group.id);

- десеріалізатори для ключів та значень повідомлень (key.deserializer, value.deserializer).

Для створення споживача Kafka було створено клас споживача NotificationConsumer, який підписаний на тему Kafka, з якої надходять повідомлення від сервісу управління замовленнями. У методі listen споживача реалізовано логіку обробки повідомлень про затримки та рекомендації.

Таким чином, як результат налаштування взаємодії програми з сервісом управління замовленнями та інтеграцією його із Telegram для налаштувати боту сповіщень, було реалізовано сервіс, який відповідає поставленим вимогам по роботі із користувачам задля сповіщення його стосовно замовлених книг та надсилання рекомендаційної літератури.

3.5 Розробка сервісу управління книгами

Сервіс управління книгами – це сервіс, що надає функціонал для управління літературою в системі, основне призначення його – управління життєвим циклом об’єкта інформації літератури у системі. Управління життєвим циклом полягає у: додаванні нової літератури, редагування та видалення існуючої та фільтрація.

Сервіс управління книгами в системі автоматизації бібліотеки реалізовано на основі Spring Boot, використовуючи MySQL для зберігання даних, MapStruct для мапінгу об'єктів, Spring Web для створення REST API, Lombok для спрощення коду, та Spring Data JPA для взаємодії з базою даних.

Основні моделі даних:

- Book: Основна сутність, що представляє книгу в системі. Поля:
  - id (Long): унікальний ідентифікатор книги, генерується автоматично;
  - title (String): назва книги (обов'язкове поле);
  - author (String): автор книги (обов'язкове поле);
  - isbn (String): міжнародний стандартний номер книги (ISBN) (обов'язкове поле, унікальне);

- category (String): категорія книги (обов'язкове поле);

- description (String): опис книги (необов'язкове поле).

- coverImage (String): посилання на обкладинку книги (необов'язкове поле).

- BookRequest: DTO (Data Transfer Object) для передачі даних про книгу від клієнта до сервера при створенні або оновленні книги.

- BookResponse: DTO для передачі даних про книгу від сервера до клієнта.

Для можливості взаємодії сервісу із зовнішніми елементами системи було розроблено BookController, що відповідає за обробку HTTP-запитів, пов'язаних з книгами. Використовує @RestController та @RequestMapping("/books") для визначення базового шляху до ендпоінтів. Має такі методи:

- createBook: приймає BookRequest, створює нову книгу та зберігає її в базі даних;

- getAll: повертає список всіх книг у вигляді BookResponse;

- getById: повертає книгу за її ідентифікатором у вигляді BookResponse;

- getAllByCategory: повертає список книг певної категорії у вигляді BookResponse;

- deleteById: видаляє книгу за її ідентифікатором.

Також було запроваджено клас BookService, що містить бізнес-логіку для роботи з книгами. Використовує BookRepository для взаємодії з базою даних. Використовує BookMapper для перетворення між сутностями Book та DTO BookRequest і BookResponse. Основні методи класу:

- createBook: створює нову книгу на основі BookRequest, перетворює її на Book та зберігає в базі даних;

- getAll: отримує всі книги з бази даних, перетворює їх на BookResponse та повертає список;

- getById: отримує книгу за ідентифікатором з бази даних, перетворює її на BookResponse та повертає;

- getAllByCategory: отримує всі книги певної категорії з бази даних, перетворює їх на BookResponse та повертає список;

- deleteById: видаляє книгу за ідентифікатором з бази даних.

Також було реалізовано ряд класів, що відповідають за збереження та форматування даних:

- BookRepository – інтерфейс, що розширює JpaRepository та надає методи для роботи з книгами в базі даних. Spring Data JPA автоматично генерує реалізацію цього інтерфейсу на основі його методів.

- BookMapper – інтерфейс, що визначає методи для перетворення між сутностями Book та DTO BookRequest і BookResponse.MapStruct автоматично генерує реалізацію цього інтерфейсу.

У результаті дослідження розробки сервісу, що відповідальний за управління книгами, було описано основні програмні компоненти програмного коду.

3.6 Висновки

У ході виконання третього розділу бакалаврської роботу було проведено варіантний аналіз і обґрунтування засобів для реалізації системи управління робою бібліотек, в результаті якого було аргументовано переваги використання мови програмування Java та інтегрованого середовища розробки IntelliJ IDEA для реалізації власної системи. Було розроблено інтерфейс користувача описавши усі основні сторінки такі як: головний екран, екран списку літератури та екран контактів. Було також розглянуто розробку таких сервісів та обговорено деталі їх реалізації: сервіс обробки даних та формування рекомендацій, сервіс сповіщень та сервіс управління книгами.

4 ТЕСТУВАНЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення відіграє критичну роль у забезпеченні якості та надійності продукту перед його впровадженням у реальне виробниче середовище. Ось кілька важливих аргументів, що обґрунтовують важливість тестування [21]:

1. Виявлення помилок. Тестування дозволяє виявити та виправити помилки в програмному забезпеченні до того, як вони вплинуть на користувачів. Це допомагає уникнути виникнення серйозних проблем, які можуть призвести до негативного впливу на бізнес або користувачів.

2. Забезпечення якості. Тестування допомагає впевнитися, що програмне забезпечення відповідає вимогам та очікуванням користувачів. Це забезпечує задоволення від використання продукту і позитивне сприйняття бренду.

3. Економія часу і коштів. Виявлення та виправлення помилок на ранніх етапах розробки коштує значно менше, ніж у випадку, якщо вони виявляться на пізніших етапах або після випуску продукту. Таким чином, тестування дозволяє зекономити час і гроші на подальшій розробці.

4. Збереження репутації. Помилки в програмному забезпеченні можуть пошкодити репутацію компанії і призвести до втрати довіри користувачів. Тестування допомагає запобігти цим негативним наслідкам і зберегти добру репутацію бренду.

Тестування програмного забезпечення може включати різні види тестування, які виконуються для різних цілей та на різних етапах розробки продукту. Ось деякі з найпоширеніших видів тестування [22]:

1. Функціональне тестування. Цей вид тестування перевіряє, чи працюють функції програмного забезпечення відповідно до вимог специфікації. Він перевіряє правильність реакції програми на різні вхідні дані та сценарії використання.

2. Тестування навантаженням. Цей вид тестування визначає, як програмне забезпечення працює під навантаженням, коли його використовують багато користувачів або обробляються великі обсяги даних. Він допомагає виявити обмеження продукту та оптимізувати його продуктивність.

3. Юзабіліті тестування – вид тестування, який оцінює, на скільки легко користувачі можуть взаємодіяти з програмним забезпеченням. Він аналізує інтерфейс користувача, навігацію та загальний досвід використання продукту.

4. Тестування безпеки – вид тестування, що перевіряє програмне забезпечення на наявність можливих вразливостей та загроз безпеці. Він включає в себе тестування на проникнення, аналізує захист даних та інші аспекти безпеки.

5. Автоматизоване тестування – вид тестування, що використовує автоматизовані скрипти для виконання тестів. Він дозволяє прискорити процес тестування, зменшити витрати та покращити його якість.

6. Регресійне тестування. Цей вид тестування перевіряє, чи не виникли нові помилки або проблеми в програмному забезпеченні після внесення змін або виправлень. Він допомагає забезпечити стабільність та надійність продукту після кожного випуску.

Для тестування власної автоматизованої системи було запроваджено функціональне тестування, оскільки це дозволяє перевірити, чи відповідає система вимогам та очікуванням щодо її функціональності. Функціональне тестування дозволяє перевірити, чи працюють всі функції системи так, як очікується, і чи забезпечується коректна обробка вхідних даних. Також цей вид тестування допомагає виявити потенційні помилки та недоліки в роботі системи, що дозволяє їх виправити перед випуском системи в експлуатацію.

4.2 Тестування основного функціоналу

Для тестування системи було запроваджено ряд функціональних тестів, які дозволяють перевірити ефективність виконання програми.

Тест кейси, які відносяться до процесу валідації та управлінням даними книг на стороні контролера:

Тест-кейс №1. Створення книги з валідними даними.

Передумова: користувач системи авторизований із привілейованим доступом (адміністратор).

Кроки:

1. Відправити POST-запит на /books з валідними даними книги у форматі JSON (за допомогою форми додавання книг).

2. Перевірити, що отримано HTTP-статус 201 Created (відбувається на стороні програмної реалізації клієнтської частини).

3. Перетворити JSON-відповідь на об'єкт BookDto (відбувається на стороні програмної реалізації серверної частини).

4. Перевірити, чи повернутий об'єкт книги відповідає очікуваним даним (окрім автоматично згенерованого ID) (відбувається на стороні програмної реалізації серверної частини).

5. Видалити створену книгу з бази даних після завершення тесту (відбувається на стороні програмної реалізації серверної частини).

Очікуваний результат: створений об'єкт книги повертається у відповіді та зберігається в базі даних. HTTP-статус 201 Created.

Результат: отриманий результат збігається з очікуваним, створений об'єкт книги повертається у відповіді та об’єкт збережений в базі даних. Отриманий HTTP-статус 201 Created.

Тест-кейс №2. Отримання всіх книг.

Передумова: немає передумов.

Кроки:

1. Відправити GET-запит на /books (клієнтська частина автоматично відправляє запит при переході на сторінку списку літератур).

2. Перевірити, що отримано HTTP-статус 200 OK (відбувається на стороні програмної реалізації серверної частини).

3. Перетворити JSON-відповідь на масив об'єктів BookDto (відбувається на стороні програмної реалізації серверної частини).

4. Порівняти отриманий список книг з попередньо визначеним списком стандартних книг, перевіряючи їх кількість та вміст (відбувається на стороні програмної реалізації серверної частини).

Очікуваний результат: повний список книг повертається у відповіді. HTTP-статус 200 OK.

Результат: отриманий результат збігається з очікуваним, повний список книг повертається у відповіді. Отриманий HTTP-статус 200 OK.

Тест-кейс №3. Отримання книги за валідним ID.

Передумова: немає передумов.

Кроки:

1. Вибрати довільну книгу зі списку книг.

2. Відправити GET-запит на /books/{id}, де {id} - це ID вибраної книги (відбувається автоматично на стороні клієнтської частини).

3. Перевірити, що отримано HTTP-статус 200 OK (відбувається на стороні програмної реалізації серверної частини).

4. Перетворити JSON-відповідь на об'єкт BookDto (відбувається на стороні програмної реалізації серверної частини).

5. Перевірити, чи повернутий об'єкт книги відповідає очікуваним даним вибраної книги (відбувається на стороні програмної реалізації серверної частини).

Очікуваний результат: книга з заданим ID повертається у відповіді. HTTP-статус 200 OK.

Результат: отриманий результат збігається з очікуваним, книга з заданим ID повертається у відповіді. Отриманий HTTP-статус 200 OK.

Тест-кейс №4. Отримання книги за невалідним ID.

Передумова: немає передумов

Кроки:

1. Відправити GET-запит на /books/{id}, де {id} - це невалідний ID книги.

2. Перевірити, що отримано HTTP-статус 404 Not Found.

Очікуваний результат: Помилка 404 Not Found при запиті з невалідним ID.

Результат: отриманий результат збігається з очікуваним, отриманий HTTP-статус 404 Not Found.

Тест-кейс 5. Видалення книги за валідним ID.

Передумова: користувач системи авторизований із привілейованим доступом (адміністратор).

Кроки:

1. Відправити DELETE-запит на /books/{id}, де {id} - це ID тестової книги (за допомогою панелі адміністратора).

2. Перевірити, що отримано HTTP-статус 204 No Content.

3. Відправити повторний GET-запит на /books, щоб отримати список всіх книг.

4. Перевірити, що тестова книга більше не присутня у списку.

Очікуваний результат: Книга успішно видалена з системи. HTTP-статус 204 No Content.

Результат: отриманий результат збігається з очікуваним, книга успішно видалена. Отриманий HTTP-статус 204 No Content.

Також варто розглянути тест кейси, які полягали у тестуванні роботи системи з базою даних. Для тестування системи із базою даних було створено відповідний клас. Цей клас тестів перевіряє роботу BookRepository, який є ключовим компонентом для взаємодії з базою даних у системі управління бібліотекою. Він використовує фреймворк Spring Data JPA для спрощення операцій з базою даних.

Детальний опис тестів:

Тест-кейс №6. Пошук книг за категорією з валідним ID.

Передумова: Перед виконанням тесту, за допомогою анотації @Sql, в базу даних додаються тестові дані (книги та категорії).

Кроки:

1. Визначити ідентифікатор категорії, для якої потрібно знайти книги.

2. Викликати метод findAllByCategoriesId з цим ідентифікатором.

3. Перевірити, що кількість повернутих книг дорівнює кількості книг, пов'язаних з цією категорією в тестових даних.

4. Перевірити, чи перша книга в результаті має непустий набір категорій (це підтверджує, що зв'язок з категоріями встановлений коректно).

5. Після завершення тесту, тестові дані видаляються з бази даних.

Очікуваний результат: Метод findAllByCategoriesId повертає всі книги, що належать до вказаної категорії.

Результат: отриманий результат збігається з очікуваним, отримано всі книги, що належать вказаній категорії.

Тест-кейс №7. Пошук книги за валідним ID.

Передумова: Тестова книга зберігається в базі даних.

Кроки:

1. Зберегти тестову книгу в базі даних.

2. Викликати метод findById з ідентифікатором збереженої книги.

3. Перевірити, чи повернута книга не є null.

4. Перевірити, чи всі поля книги (крім id) збігаються з полями тестової книги.

Очікуваний результат: Метод findById коректно повертає книгу за її ідентифікатором.

Результат: отриманий результат збігається з очікуваним, отримано книгу, що відповідає ідентифікаторові.

Тест-кейс №8. Пошук всіх книг.

Передумова: немає передумов.

Кроки:

1. Визначити очікувану кількість книг, що має бути повернена (в даному випадку, це кількість книг в тестових даних).

2. Створити об'єкт Pageable для пагінації результатів (встановлено розмір сторінки 10).

3. Викликати метод findAll з цим об'єктом Pageable.

4. Перевірити, чи кількість книг на першій сторінці результатів збігається з очікуваною кількістю.

Очікуваний результат: Метод findAll коректно повертає всі книги, доступні в базі даних.

Результат: отриманий результат збігається з очікуваним, отримано всі книги, що наявні в базі даних під час тесту.

Ці тести демонструють найкращі практики тестування репозиторіїв Spring Data JPA, забезпечуючи перевірку ключових операцій з базою даних та гарантуючи коректну роботу системи управління бібліотекою.

Для забезпечення максимальної точності виконання програмного коду, було також запроваджені тести на рівні сервісів (частини коду, що відповідає за логіку роботи програмної системи).

Основним класом, що запроваджує тести на рівні сервісів є BookServiceTest. Цей клас використовує фреймворк Mockito для створення моків (імітацій) репозиторію (BookRepository) та мапера (BookMapper), що дозволяє ізолювати бізнес-логіку від деталей взаємодії з базою даних та перетворення об'єктів.

Тести зосереджені на перевірці методів сервісу BookService, таких як save, getAll, getById, update та deleteById.

Тест-кейс №9. Збереження книги з валідними даними.

Передумова: немає передумов.

Кроки:

1. Налаштувати моки репозиторію та мапера, щоб вони повертали очікувані значення при виклику їх методів.

2. Викликати метод save сервісу з коректними вхідними даними.

3. Перевірити, що повернутий об'єкт BookDto збігається з очікуваним.

Очікуваний результат: Метод save повертає об'єкт книги з коректними даними.

Результат: отриманий результат збігається з очікуваним, метод повернув книгу з очікуваними даними.

Тест-кейс №10. Отримання книг з пагінацією.

Передумова: немає передумов.

Кроки:

1. Налаштувати мок репозиторію, щоб він повертав сторінку з однією тестовою книгою.

2. Викликати метод getAll з об'єктом Pageable, що вказує на пагінацію.

3. Перевірити, що повернутий список книг містить лише один елемент - очікувану книгу.

Очікуваний результат: метод getAll повертає сторінку з книгами відповідно до заданої пагінації.

Результат: отриманий результат збігається з очікуваним, метод повернув сторінку з книгами відповідно до заданої пагінації.

Тест-кейс №11. Отримання книги за валідним ID.

Передумова: немає передумов.

Кроки:

1. Налаштувати мок репозиторію, щоб він повертав тестову книгу при запиті за її ідентифікатором.

2. Викликати метод getById з існуючим ідентифікатором.

3. Перевірити, що повернутий об'єкт книги збігається з очікуваним.

Очікуваний результат: метод getById повертає книгу за її ідентифікатором, якщо книга існує.

Результат: отриманий результат збігається з очікуваним, метод повернув книгу, що відповідає ідентифікаторові.

Тест-кейс №12. Отримання книги за невалідним ID.

Передумова: немає передумов.

Кроки:

1. Налаштувати мок репозиторію, щоб він повертав порожній Optional при запиті за неіснуючим ідентифікатором.

2. Викликати метод getById з неіснуючим ідентифікатором.

3. Перевірити, що було викинуто виняток EntityNotFoundException з очікуваним повідомленням про помилку.

Очікуваний результат: метод getById викидає виняток EntityNotFoundException, якщо книга з вказаним ідентифікатором не знайдена.

Результат: отриманий результат збігається з очікуваним, метод повернув очікувану помилку, через відсутність книги за ідентифікатором у базі даних.

Тест-кейс №13. Видалення книги за валідним ID.

Передумова: немає передумов.

Кроки:

1. Налаштувати мок репозиторію, щоб він успішно видаляв книгу при запиті за її ідентифікатором.

2. Викликати метод deleteById з існуючим ідентифікатором.

3. Перевірити, що метод завершився без винятків.

Очікуваний результат: метод deleteById успішно видаляє книгу з системи.

Результат: отриманий результат збігається з очікуваним, після виконання методу книга була видалена із бази даних.

Тест-кейс №14. Видалення книги за невалідним ID.

Передумова: немає передумов.

Кроки:

1. Налаштувати мок репозиторію, щоб він повертав порожній Optional при запиті на видалення неіснуючої книги.

2. Викликати метод deleteById з неіснуючим ідентифікатором.

3. Перевірити, що було викинуто виняток EntityNotFoundException з очікуваним повідомленням про помилку.

Очікуваний результат: метод deleteById викидає виняток EntityNotFoundException, якщо книга з вказаним ідентифікатором не знайдена.

Результат: отриманий результат збігається з очікуваним, метод повернув очікувану помилку, оскільки книга за ідентифікатором була відсутня у базі даних.

Важливі аспекти, які були застосовані під час тестування:

1. Ізоляція бізнес-логіки. Використання моків дозволяє зосередитися на тестуванні саме бізнес-логіки сервісу, не зачіпаючи роботу репозиторію та мапера;

2. Покриття різних сценаріїв. Тести охоплюють основні сценарії використання сервісу (створення, отримання, оновлення, видалення) та враховують можливі помилки (відсутність книги).

Отже, у ході роботи над підрозділом бакалаврської дипломної роботи було запроваджено та описано ряд тест-кейсів, що відповідають за тестування різних шарів програмної системи.

4.3 Розробка інструкції користувача

Інструкція користувача є важливим документом, який супроводжує будь-яке програмне забезпечення, особливо веб-застосунки. Вона відіграє ключову роль у забезпеченні ефективної взаємодії користувачів з продуктом та має низку переваг [23]:

1. Полегшення використання. Інструкція надає користувачам чіткі та зрозумілі кроки для використання всіх функцій веб-застосунку. Це особливо важливо для складних систем або тих, що містять нові або незвичні елементи інтерфейсу.

2. Зменшення кількості звернень до служби підтримки. Наявність детальної інструкції дозволяє користувачам самостійно вирішувати більшість проблем, що виникають під час роботи з застосунком. Це знижує навантаження на службу підтримки та економить час як користувачів, так і розробників.

3. Підвищення задоволеності користувачів. Коли користувачі можуть легко та швидко розібратися з роботою застосунку, вони отримують позитивний досвід взаємодії з ним. Це підвищує їхню лояльність та сприяє поширенню інформації про продукт.

Для автоматизованої системи управління бібліотекою було розроблено власну інструкцію користувача, що описана далі.

Робота користувача з системою управління бібліотекою розпочинається із переходу на основну сторінку веб сайту, що представляє клієнтську частину проєкту. На головній сторінці користувач має змогу виконати наступні дії (див. рис. 4.1):

1. Змінити тему забарвлення веб сайту (світлу/темну).
2. Авторизуватися/зареєструватися у систему
3. Перейти на іншу сторінку: сторінку списку книг або сторінку контактів.
4. Під’єднати розповсюдження рекомендацій до його номеру телефону натиснувши кнопку «Залишатися в курсі!» або залишити адресу електронної пошти у відповідному полі внизу сторінки (див. рис. 4.2).

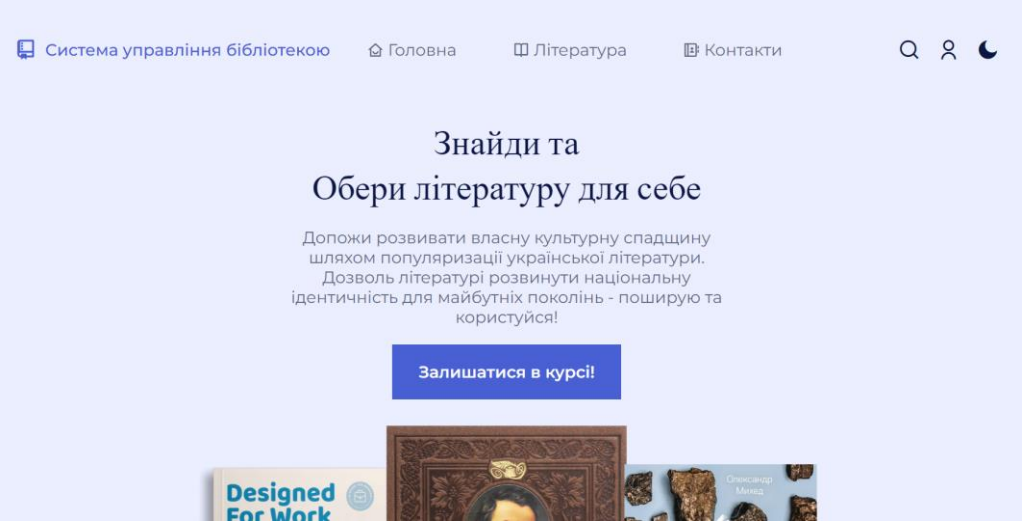

Рисунок 4.1 – Зображення головної сторінки системи №1

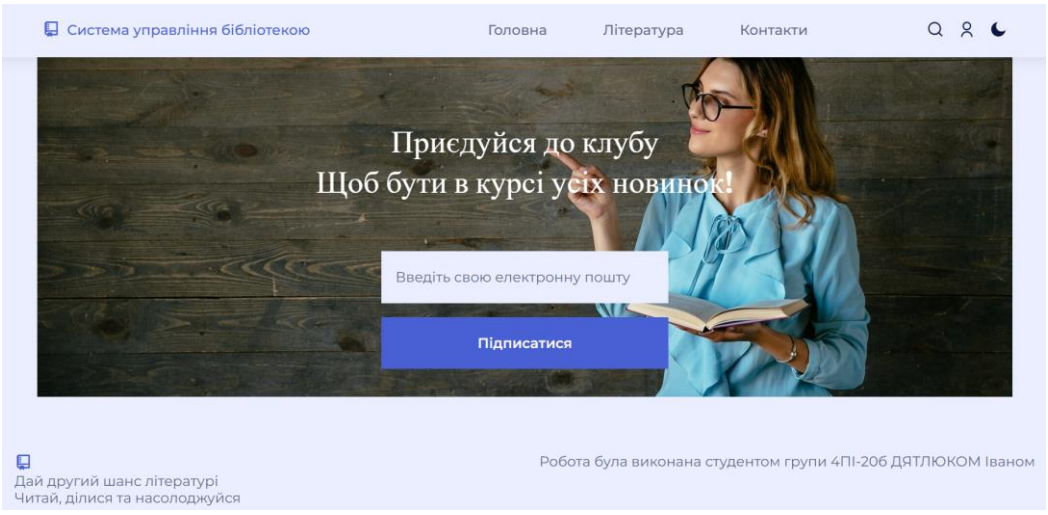

Рисунок 4.2 – Зображення головної сторінки системи №2

Для повноцінної роботи користувача з системою, йому необхідно провести авторизацію (за наявності облікового запису в систему) або реєстрації (у випадку якщо користувач користується системою вперше).

Зображення із вікнами авторизації та реєстрації продемонстровано на рисунках 4.3 та 4.4.

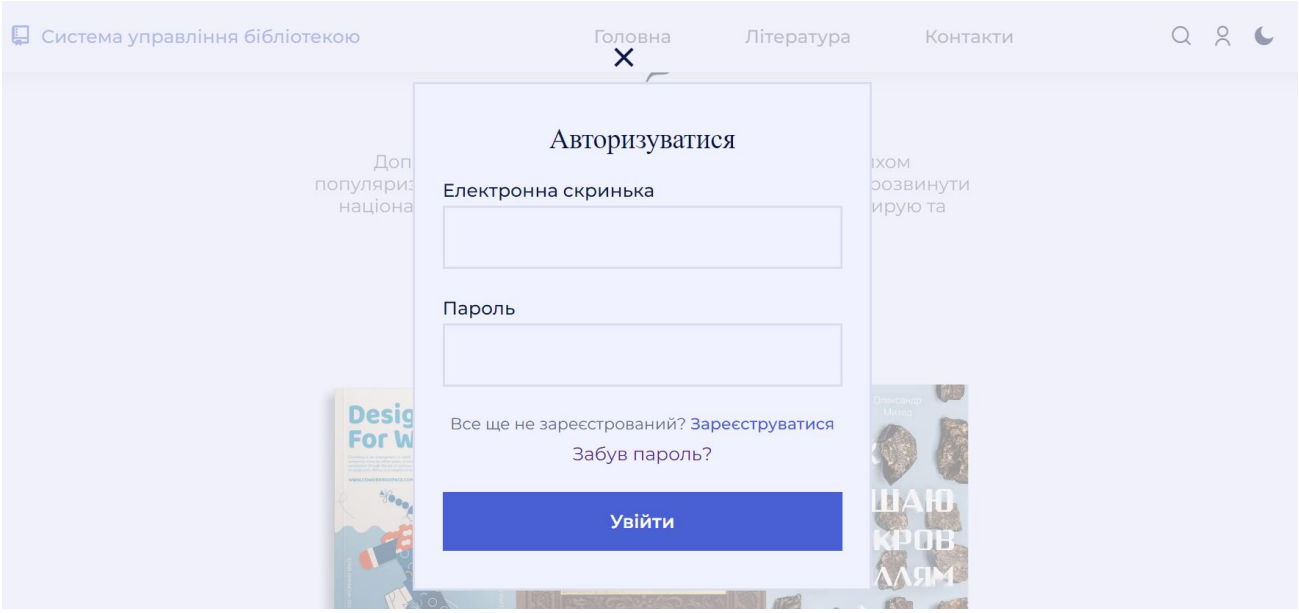

Рисунок 4.3 – Зображення вікна авторизації

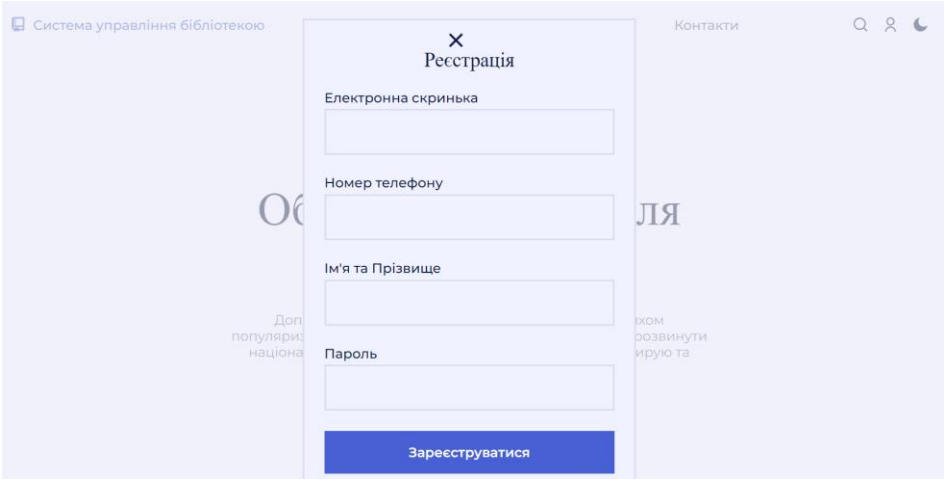

Рисунок 4.4 – Зображення вікна реєстрації

Для авторизації користувача в системі йому необхідно ввести дані уже існуючого аккаунту, а саме дані про електронну скриньку користувача та його пароль в системі, після чого користувач отримує доступ до можливості

формування замовлень. У випадку, якщо користувач користується системою вперше йому необхідно натиснути на виділене синім кольором слово «Зареєструватися» (див. рис. 4.5).

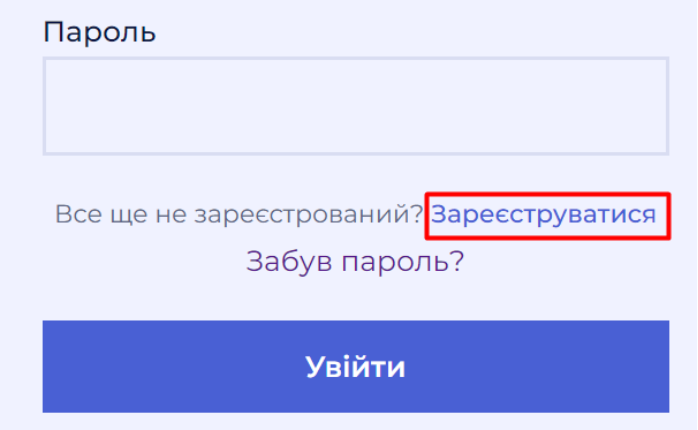

Рисунок 4.5 – Зображення вказівки для подальшої реєстрації користувача

Після успішної авторизації/реєстрації користувач матиме змогу для подальшого формування замовлення.

Для формування замовлення користувачеві необхідно перейти на сторінку списку літератури, та за допомого кнопки «Додати», вибрати бажану кількість книг (див. рис. 4.6).

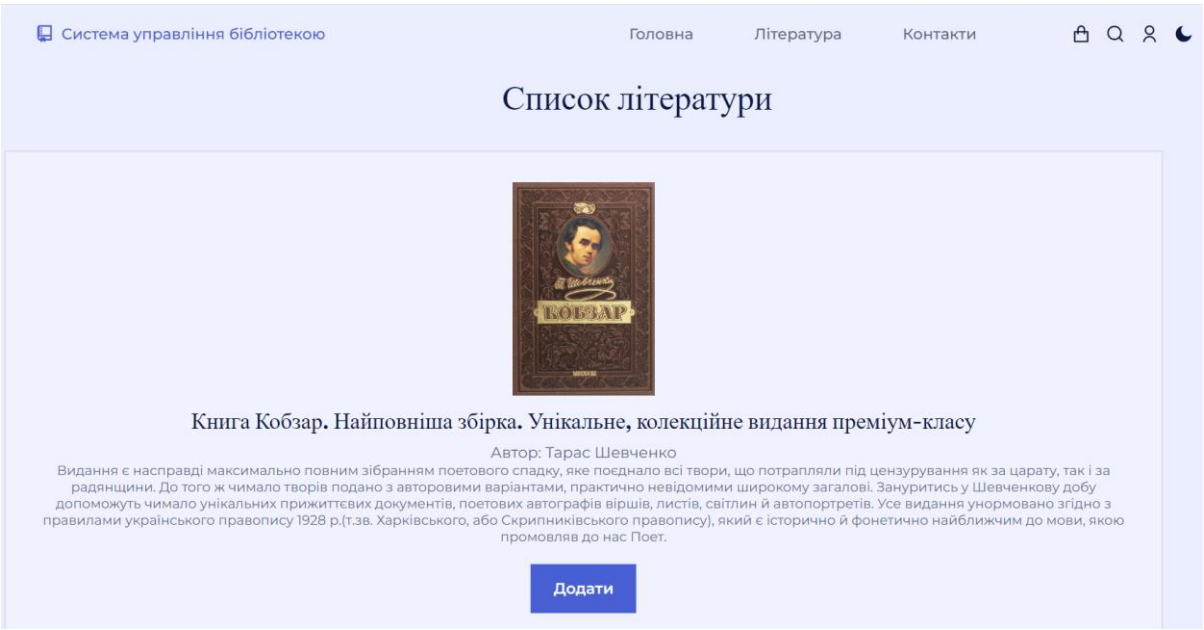

Рисунок 4.6 – Зображення прикладу сторінки «Література»

Після натиску на кнопку «Додати» у авторизованого користувача з’являється можливість підтвердити замовлення, натиснувши на іконку корзини (див. рис. 4.7).

Рисунок 4.7 – Зображення вказівки для підтвердження замовлення

У меню корзини користувач має змогу видалити елемент із неї або підтвердити свій вибір, натиснувши на кнопку «Підтвердити» (див. рис. 4.8).

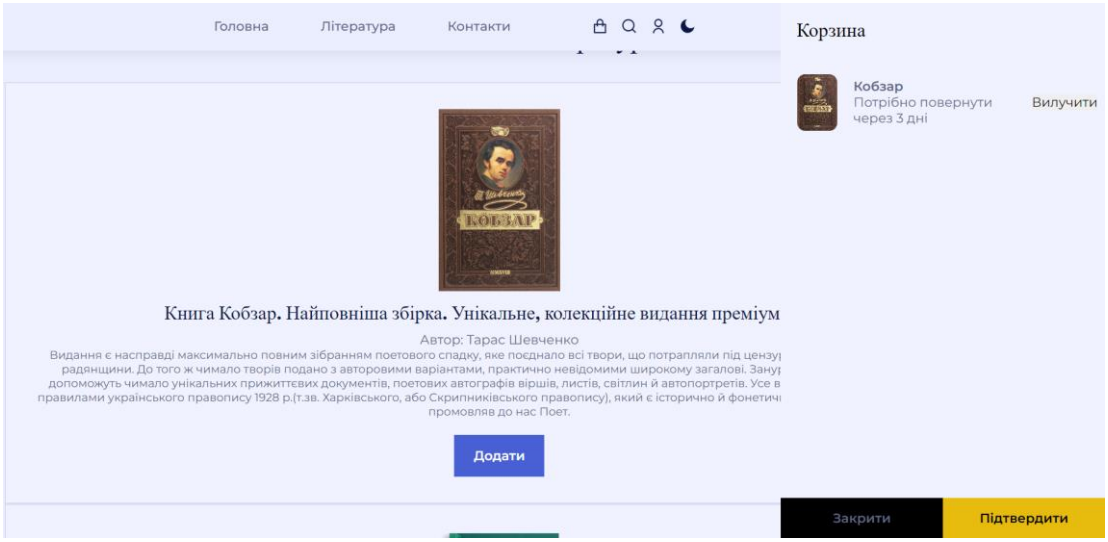

Рисунок 4.8 – Зображення графічного елементу корзини

Для подальшого отримання сповіщень, користувачу необхідно під’єднатися до Telgam боту: https://t.me/libraryNBot (див. рис. 4.9).

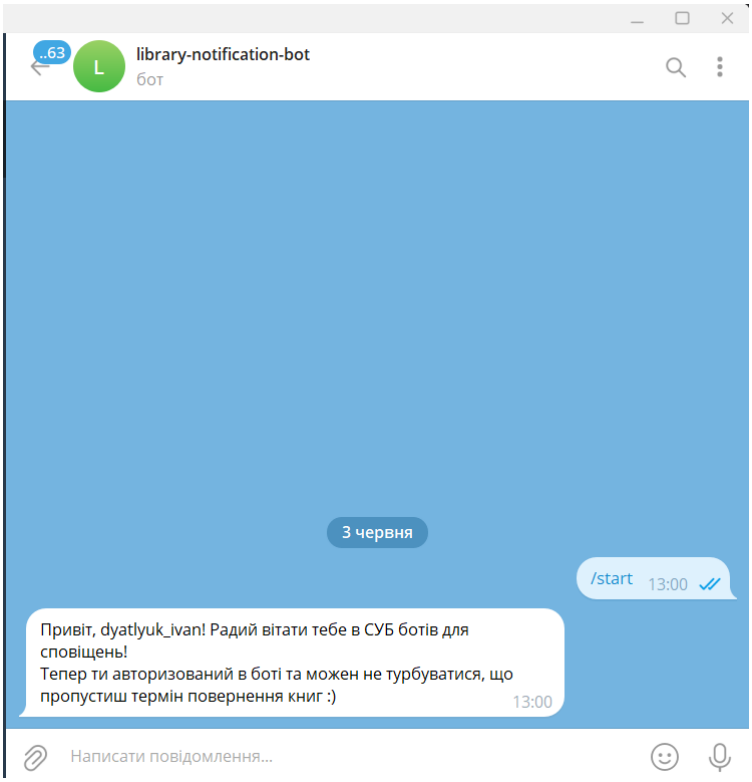

Рисунок 4.9 – Зображення запущеного Telegram боту

Далі, для остаточного налаштування сповіщень необхідно надіслати адресу електронної пошти боту, для автентифікації система (див. рис. 4.10).

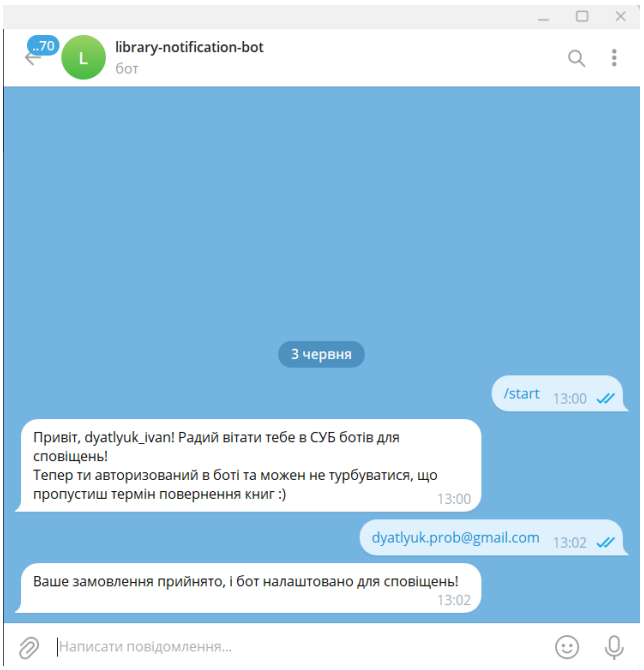

Рисунок 4.10 – Зображення налаштування боту

Загальне налаштування користувача у системі завершене.

Також, можлива авторизація користувача із привілейованими правами доступу, які передбачаються під час розробки системи. Привілейовані права надають користувачу можливість отримувати додаткові аналітичні дані з системи (див. рис. 4.11).

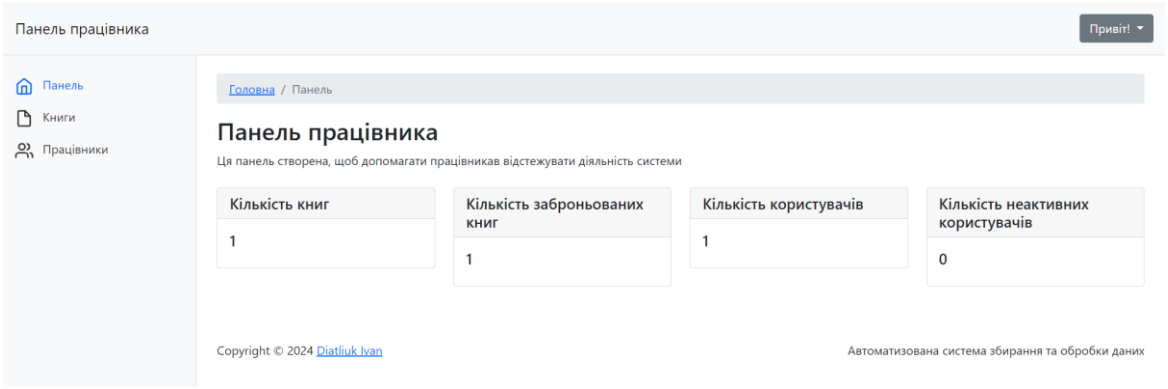

Рисунок 4.11 – Зображення панелі працівника

Уся необхідна інформація знаходиться на сторінці «Панель», тут наводяться такі дані:

- кількість книг – це загальна кількість книг, яких було заброньовано чи повернуто за весь час роботи системи;

- кількість заброньованих книг – це загальна кількість книг, які наразі заброньовані користувачами;

- кількість користувачів – це кількість активних користувачів, що бронюють книги впродовж заданого часу (1.5 роки);

- кількість неактивних користувачів – це кількість користувачів, які деактивували свої облікові записи або не проводили ніяких операцій у системі протягом заданого часу (1.5 роки).

Отже, у підрозділі бакалаврської дипломної роботи було розглянуто актуальність розробки інструкції користувача та було описану інструкція для розроблюваної системи.

4.4 Системні вимоги

Вказання мінімальних та рекомендованих системних вимог для

автоматизованої системи управління бібліотекою є важливим кроком для забезпечення її успішної роботи та задоволення потреб користувачів. Мінімальні вимоги визначають базову конфігурацію, необхідну для запуску системи, гарантуючи її коректне функціонування на пристроях користувачів. Це допомагає уникнути проблем з продуктивністю та помилок, які можуть виникнути при спробі запуску системи на недостатньо потужному обладнанні.

Рекомендовані вимоги, своєю чергою, вказують на конфігурацію, яка забезпечить оптимальну продуктивність та комфортну взаємодію з системою, особливо при великому навантаженні або обсязі даних. Це дозволяє користувачам насолоджуватися швидкою та плавною роботою системи, уникаючи затримок та інших незручностей.

Крім того, надання чіткої інформації про системні вимоги сприяє прозорості та допомагає користувачам прийняти обґрунтоване рішення щодо використання системи. Це також важливо для організацій, які планують впровадження системи, оскільки дозволяє їм правильно спланувати необхідні ресурси та бюджет.

Таким чином, вказання мінімальних та рекомендованих системних вимог є невід'ємною частиною процесу розробки та впровадження автоматизованої системи управління бібліотекою, що забезпечує її сумісність, продуктивність, зручність використання та задоволеність користувачів.

Для системи автоматизації управління бібліотек було виведено власні мінімальні та рекомендовані системні вимоги.

Мінімальні системні вимоги. Ці вимоги дозволять системі функціонувати, але продуктивність може бути обмежена при великому навантаженні або обсязі даних:

- Процесор: 2 ядра, 2.0 ГГц;

- Оперативна пам'ять (RAM): 4 ГБ;

- Місце на диску: 20 ГБ (для операційної системи, бази даних MySQL, Spring Boot додатку та клієнтської частини);

- Операційна система: Windows, Linux або macOS;

- База даних: MySQL 5.7 або вище;

- Java: Java 8 або вище;

- Веб-браузер: Сучасний браузер (Chrome, Firefox, Edge, Safari).

Рекомендовані системні вимоги. Ці вимоги забезпечать оптимальну продуктивність та комфортну роботу з системою:

- Процесор: 4 ядра, 3.0 ГГц або вище;

- Оперативна пам'ять (RAM): 8 ГБ або більше;

- Місце на диску: 50 ГБ або більше (залежно від обсягу даних у бібліотеці);

- Операційна система: Linux (рекомендується для серверного середовища);

- База даних: MySQL 8.0 або вище (для покращеної продуктивності та нових функцій);

- Java: Java 11 або вище (для підтримки новітніх можливостей Spring Boot);

- Веб-браузер: Останні версії Chrome, Firefox, Edge або Safari.

Додаткові рекомендації щодо покращення роботи системи:

- Сервер: якщо система буде використовуватися великою кількістю користувачів або матиме значний обсяг даних, рекомендується розгорнути її на окремому сервері.

- Моніторинг: для власного використання у комерційних цілях варто розглянути встановлення інструментів моніторингу для відстеження продуктивності та стану системи. Такими інструментами може слугувати New Relic або Splunk.

4.5 Висновки

У розділ було проаналізовано методи тестування програмного забезпечення, проведено модульне тестування валідації даних. Усі тести проведено успішно, критичних помилок та збоїв не виявлено. Також, було описано інструкцію користувача та технічні вимоги для забезпечення коректного виконання роботи системи.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмне забезпечення автоматизованої системи управління бібліотеками. Розроблене ПЗ має на меті підвищення ефективності управління роботою бібліотеки за рахунок впровадження таких сучасних технологій як штучний інтелект та телеграм боти, що сприятиме популяризації літератури та бібліотек.

Було розглянуто стан питання автоматизованих систем управління бібліотеками та методи розв’язання задачі, проведено порівняльний аналіз програмних продуктів аналогів, описано напрямки вдосконалення таких систем та підходи до створення.

Було спроєктовано модель даних та архітектуру програмного застосунку. Було розроблено загальний алгоритм роботи системи та методи формування рекомендацій для користувачів та взаємодії системи із Telegram ботом.

Було проведено порівняльний аналіз переваг та недоліків популярних мов програмування та середовищ розробки. На основі аналізу було обрано мову програмування Java, а саме Spring фреймворк та платформу Apache Kafka, для розробки автоматизованої системи та інтегроване середовище розробки IntelliJ IDEA. Описано процес розробки основних компонентів систем, таких як: інтерфейс користувача та сервіси серверної частиини.

Також, було розглянуто тему актуальності впровадження тестувань у програмні застосунки та види тестувань, на основі цього було обґрунтовано доцільність впровадження функціонального тестування для забезпечення працездатності та ефективності виконання роботи системи. Було створено тестові класи, які відповідають за перевірку основного функціоналу системи За результатами тестування було підтверджено працездатність та ефективність роботи системи. Також було описано інструкцію користувача та визначено системні вимоги для програмного застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дятлюк І. С., Романюк О. В. Інтеграція сучасних технологій із програмним забезпеченням для ефективного управління ресурсами бібліотеки. Матеріали LIII науково-технічної конференції підрозділів ВНТУ, Вінниця, 20-22 березня 2024 р. Електрон. текст. дані. 2024. URI: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20691.

2. Дятлюк І. С., Романюк О.В. Впровадження гібридної системи рекомендацій із використанням штучного інтелекту. / Молодь в науці: дослідження, проблеми, перспективи (МН-2024), Вінниця 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21655/17880

3. Koha [Електронний ресурс] – Режим доступу до ресурсу: https://koha-community.org/

4. Folio [Електронний ресурс] – Режим доступу до ресурсу: https://folio.org/

5. Alma [Електронний ресурс] – Режим доступу до ресурсу: https://almacam.com

6. The Principles of Designing Microservices [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/blog/implementing-designing-microservices

7. Exploring Monolith Architecture: Pros, Cons, and Modern Alternatives [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@sonalisahi/exploring-monolith-architecture-pros-cons-and-modern-alternatives-1cefefdfce96

8. Microservice Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://microservices.io/

9. Building Applications with Serverless Architectures [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/lambda/serverless-architectures-learn-more

10. Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor Recommender Systems Handbook – Нью-Йорк 2010.

11. Monolithic Approach vs. Microservices Approach: Which is Right for Your Application? [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/monolithic-approach-vs-microservices-which-right-your-majid-sheikh.

12. Service Discovery [Електронний ресурс] – Режим доступу до ресурсу: https://avinetworks.com/glossary/service-discovery/.

13. Шилдт Г., Ковард Д. Java: Повний довідник, Тринадцяте видання. Нью-Йорк: McGraw Hill Education, 2024.

14. C# [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/dotnet/

15. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/about/

16. Eclipse [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Eclipse

17. What is Intellij IDEA - A Comprehensive Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/what-is-intellij-idea-article

18. VS Code [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Visual_Studio_Code

19. The Importance of User Interface Design [Електронний ресурс] – Режим доступу до ресурсу: https://www.plego.com/importance-user-interface-design/

20. Lemay L., Colburn R., Kyrnin J. HTML, CSS & JavaScript Web Publishing. Indianapolis: Sams Teach Yourself

21. The Importance of Testing And How to Do It Properly [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/importance-testing-how-do-properly-peter-tylee-lahjc

22. Типи тестування: Types of software testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

23. Інструкція користувача: User Manual [Електронний ресурс] – Режим доступу до ресурсу: https://qatestlab.com/resources/knowledge-center/sample-deliverables/user-manuals/

ДОДАТКИ

Додаток А. Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного забезпечення системи управління роботою бібліотеки з використанням фреймворку Spring і платформи Apache Kafka» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
д.т.н., доцент Романюк О. В.
« 12 » березня 2024 р.
Виконав:
студент гр. 4ПІ-20б Дятлюк І. С.
« 12 » березня 2024 р.

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка автоматизованої системи управлінням роботою бібліотеки з використанням фреймворку Spring і платформи Apache Kafka».

Галузь застосування – галузь бібліотек, системи, що передбачають управління літературними фондами.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від 11.03.2024р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є підвищення ефективності управління роботою бібліотеки за рахунок впровадження таких сучасних технологій як штучний інтелект та телеграм боти, що сприятиме популяризації літератури та бібліотек.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. The Principles of Designing Microservices [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/blog/implementing-designing-microservices

2. Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor Recommender Systems Handbook – Нью-Йорк 2010.

3. Шилдт Г., Ковард Д. Java: Повний довідник, Тринадцяте видання. Нью-Йорк: McGraw Hill Education, 2024.

4. Lemay L., Colburn R., Kyrnin J. HTML, CSS & JavaScript Web Publishing. Indianapolis: Sams Teach Yourself

5. Технічні вимоги

Тип програми – веб система; модель розробки – ітеративна; СУБД для керування базою даних – MySQL; засоби керування чергами повідомлень – Apache Kafka; вхідні дані: інформація про книги, особисті дані користувачів, метадані запитів; вихідні дані: відображення списку книг та категорій, сповіщення, аналітичні дані та рекомендації; середовище розробки – IntelliJ Idea; мова програмування – Java; використанні додаткові бібліотеки та фреймворки – Spring (Boot, Data JPA, Web, Kafka).

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та постановка задач дослідження	14.03.24 – 19.03.24
2	Розробка архітектури та моделі системи	19.03.24 – 26.03.24
3	Розробка загального алгоритму роботи системи	26.03.24 – 03.04.24
4	Розробка методу формування рекомендацій	03.04.24 – 9.04.24
5	Розробка методу взаємодії з Telegram ботом	09.04.24 – 17.04.24
5	Аналіз та вибір засобів для реалізації програмної системи	17.04.24 – 20.04.24
6	Розробка програмного забезпечення	20.04.24 – 19.05.24
7	Тестування системи	19.05.24 – 23.05.24
	Оформлення матеріалів до захисту БДР	23.05.24 – 30.05.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного забезпечення системи управління роботою бібліотеки з використанням фреймворку Spring і платформи Apache Kafka

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.В.

Оригінальність	96,7
Схожість	3,3

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**
☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховання недобросовісних запозичень.

Особа, відповідальна за перевірку — Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи — Дятлюк І.С.

Керівник роботи — Романюк О.В.

Додаток В. Лістинг програми

```
//API-gateway project
//main.java
package org.vntu.apigateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ApiGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}

//application.properties
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
spring.application.name=api-gateway

logging.level.root=INFO
logging.level.org.spingframework.cloud.gateway.route.RouteDefinitionLocator=INFO
logging.level.org.spingframework.cloud.gateway=TRACE

server.port=8080

## Product Service Route
spring.cloud.gateway.routes[0].id=book-service
spring.cloud.gateway.routes[0].uri=lb://book-service:0
spring.cloud.gateway.routes[0].predicates[0]=Path=/api/books

## Order Service Route
spring.cloud.gateway.routes[1].id=order-service
spring.cloud.gateway.routes[1].uri=lb://order-service:0
spring.cloud.gateway.routes[1].predicates[0]=Path=/api/orders

## Discover Server Route
spring.cloud.gateway.routes[2].id=discover-route
spring.cloud.gateway.routes[2].uri=http://localhost:8761
spring.cloud.gateway.routes[2].predicates[0]=Path=/eureka/web
spring.cloud.gateway.routes[2].filters[0]=SetPath=/

//pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.vntu</groupId>
        <artifactId>architecture-project</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <artifactId>api-gateway</artifactId>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
    </dependencies>

</project>

//Book service project
//main.java
package org.vntu.bookservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class BookServiceApplication {

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

}

//MapperConfig.java
package org.vntu.bookservice.config;

import org.mapstruct.InjectionStrategy;
import org.mapstruct.NullValueCheckStrategy;

@org.mapstruct.MapperConfig(
        componentModel = "spring",
        injectionStrategy = InjectionStrategy.CONSTRUCTOR,
        nullValueCheckStrategy = NullValueCheckStrategy.ALWAYS,
        implementationPackage = "<PACKAGE_NAME>.iml"
)
public class MapperConfig {
}

//WebClientConfig.java
package org.vntu.bookservice.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;

@Configuration
public class WebClientConfig {
    @Bean
    @LoadBalanced
    public WebClient.Builder webClientBuilder() {
        return WebClient.builder();
    }
}

//BookController.java
package org.vntu.bookservice.controller;


import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.vntu.bookservice.dto.BookRequest;
import org.vntu.bookservice.dto.BookResponse;
import org.vntu.bookservice.service.BookService;
```

```
import java.util.List;

@RestController
@RequestMapping
@RequiredArgsConstructor
public class BookController {
    private final BookService bookService;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void createBook(@RequestBody BookRequest bookRequest) {
        bookService.createBook(bookRequest);
    }

    @GetMapping
    @ResponseStatus(HttpStatus.OK)
    public List<BookResponse> getAll() {
        return bookService.getAll();
    }

    @GetMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public BookResponse getById(@PathVariable Long id) {
        return bookService.getById(id);
    }

    @GetMapping("/category")
    @ResponseStatus(HttpStatus.OK)
    public List<BookResponse> getAllByCategory(@RequestParam String category) {
        return bookService.getAllByCategory(category);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public void deleteById(@PathVariable Long id) {
        bookService.deleteById(id);
    }
}

//BookRequest.java
package org.vntu.bookservice.dto;

import jakarta.persistence.Column;
import lombok.Data;

@Data
public class BookRequest {
    private String title;
    private String author;
    private String isbn;
    private String category;
    private String description;
    private String coverImage;
    private Integer quantity;
}

//BookResponse.java
package org.vntu.bookservice.dto;

import lombok.Data;

@Data
public class BookResponse {
    private Long id;
    private String title;
    private String author;
    private String isbn;
    private String category;
    private String description;
```

```
    private String coverImage;
}

//InventoryRequest.java
package org.vntu.bookservice.dto;

import lombok.Builder;
import lombok.Data;

@Data
@Builder
public class InventoryRequest {
    private String skuCode;
    private Integer quantity;
}

//EntityNotFoundException.class
package org.vntu.bookservice.exception;

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

//BookMapper.java
package org.vntu.bookservice.mapper;

import org.mapstruct.Mapper;
import org.vntu.bookservice.config.MapperConfig;
import org.vntu.bookservice.dto.BookRequest;
import org.vntu.bookservice.dto.BookResponse;
import org.vntu.bookservice.model.Book;

@Mapper(config = MapperConfig.class)
public interface BookMapper {
    Book toModel(BookRequest bookRequest);

    BookRequest toRequestDto(Book book);

    BookResponse toResponseDto(Book book);
}

//Book.class
package org.vntu.bookservice.model;


import jakarta.persistence.*;
import lombok.Data;

@Entity
@Table(name = "books")
@Data
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String author;

    @Column(nullable = false, unique = true)
    private String isbn;

    @Column(nullable = false)
    private String category;
```

```
    private String description;

    private String coverImage;
}

//BookRepository.java
package org.vntu.bookservice.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.vntu.bookservice.model.Book;

import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findAllByCategory(String category);
}

//BookService.java
package org.vntu.bookservice.service.impl;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import org.vntu.bookservice.dto.BookRequest;
import org.vntu.bookservice.dto.BookResponse;
import org.vntu.bookservice.dto.InventoryRequest;
import org.vntu.bookservice.exception.EntityNotFoundException;
import org.vntu.bookservice.mapper.BookMapper;
import org.vntu.bookservice.model.Book;
import org.vntu.bookservice.repository.BookRepository;
import org.vntu.bookservice.service.BookService;
import reactor.core.publisher.Mono;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class BookServiceImpl implements BookService {
    private static final String INVENTORY_SERVICE_URL = "http://inventory-
service/api/inventory/create";

    private final BookRepository bookRepository;
    private final BookMapper bookMapper;
    private final WebClient.Builder webClientBuilder;

    @Override
    public void createBook(BookRequest bookRequest) {
        Book book = bookMapper.toModel(bookRequest);

        bookRepository.save(book);
        InventoryRequest inventoryRequest = InventoryRequest.builder()
                        .skuCode(String.valueOf(book.getId()))
                        .quantity(bookRequest.getQuantity())
                        .build();

        webClientBuilder.build()
                .post()
                .uri(INVENTORY_SERVICE_URL)
                .accept(MediaType.APPLICATION_JSON)
                .contentType(MediaType.APPLICATION_JSON)
                .body(Mono.just(inventoryRequest), InventoryRequest.class)
                .retrieve()
                .toBodilessEntity()
```

```
                .block();

        log.info("Book {} is created!", book.getId());
    }

    @Override
    public List<BookResponse> getAll() {
        return bookRepository.findAll().stream()
                .map(bookMapper::toResponseDto)
                .toList();
    }

    @Override
    public BookResponse getById(Long id) {
        Book bookById = bookRepository.findById(id).orElseThrow(
                () ->  new EntityNotFoundException("The book with id: " + id + " is not found!")
        );
        return bookMapper.toResponseDto(bookById);
    }

    @Override
    public List<BookResponse> getAllByCategory(String category) {
        List<Book> booksByCategory = bookRepository.findAllByCategory(category);
        return booksByCategory.stream()
                .map(bookMapper::toResponseDto)
                .toList();
    }

    @Override
    public void deleteById(Long id) {
        bookRepository.deleteById(id);
    }
}

//applicaton.properties
spring.application.name=book-service
spring.datasource.url=jdbc:mysql://localhost:3306/library_system_db
spring.datasource.username=root
spring.datasource.password=redblack0root

spring.jpa.open-in-view=false
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create-drop

server.port=0
server.servlet.context-path=/api/books
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

//DiscoveryService project
//main.java
package org.vntu.discoveryserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

import javax.swing.*;

@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServerApplication.class);
    }
}

//application.properties
eureka.instance.hostname=localhost
eureka.client.register-with-eureka=false
```

```
eureka.client.fetch-registry=false
server.port=8761

//Inventory service project
//main.java
package org.vntu.inventoryservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class InventoryServiceApplication {

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

}

//InventoryContoller.java
package org.vntu.inventoryservice.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.vntu.inventoryservice.dto.InventoryRequest;
import org.vntu.inventoryservice.dto.InventoryResponse;
import org.vntu.inventoryservice.service.InventoryService;

import java.util.List;

@RestController
@RequestMapping
@RequiredArgsConstructor
public class InventoryController {
    private final InventoryService inventoryService;

    @GetMapping("/get")
    @ResponseStatus(HttpStatus.OK)
    public List<InventoryResponse> isInStock(@RequestParam List<String> skuCodes) {
        return inventoryService.isInStock(skuCodes);
    }

    @PostMapping("/create")
    @ResponseStatus(HttpStatus.OK)
    public void createInventory(@RequestBody InventoryRequest inventoryRequest) {
        inventoryService.create(inventoryRequest);
    }
}
//InventoryRequest.java
package org.vntu.inventoryservice.dto;

import lombok.Data;

@Data
public class InventoryRequest {
    private String skuCode;
    private Integer quantity;
}

//InventoryResponse.java
package org.vntu.inventoryservice.dto;

import lombok.Builder;
import lombok.Data;

@Data
```

```
@Builder
public class InventoryResponse {
    private String skuCode;
    private boolean isInStock;
}

//Inventory.java
package org.vntu.inventoryservice.model;

import jakarta.persistence.*;
import lombok.Data;

@Entity
@Table(name = "inventories")
@Data
public class Inventory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String skuCode;
    private Integer quantity;
}

//InventoryRepository.java
package org.vntu.inventoryservice.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.vntu.inventoryservice.model.Inventory;

import java.util.List;
import java.util.Optional;

public interface InventoryRepository extends JpaRepository<Inventory, Long> {
    List<Inventory> findBySkuCodeIn(List<String> skuCode);
}

//InventoryService.java
package org.vntu.inventoryservice.service.impl;

import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.vntu.inventoryservice.dto.InventoryRequest;
import org.vntu.inventoryservice.dto.InventoryResponse;
import org.vntu.inventoryservice.model.Inventory;
import org.vntu.inventoryservice.repository.InventoryRepository;
import org.vntu.inventoryservice.service.InventoryService;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class InventoryServiceImpl implements InventoryService {
    private final InventoryRepository inventoryRepository;

    @Override
    @Transactional
    public List<InventoryResponse> isInStock(List<String> skuCodes) {
        return inventoryRepository.findBySkuCodeIn(skuCodes).stream()
                .map(inventory ->
                    InventoryResponse.builder().skuCode(inventory.getSkuCode())
                            .isInStock(inventory.getQuantity() > 0)
                            .build())
                .toList();
    }
```

```
    @Override
    public void create(InventoryRequest inventoryRequest) {
        Inventory inventory = new Inventory();
        inventory.setQuantity(inventoryRequest.getQuantity());
        inventory.setSkuCode(inventoryRequest.getSkuCode());

        inventoryRepository.save(inventory);
        log.info("The Inventory is saved!");
    }
}

// KafkaConsumerConfig
package org.dtlk.analyticsandrecommendationservice.config;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaConsumerConfig {
    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        configProps.put(ConsumerConfig.GROUP_ID_CONFIG, "analytics-and-recommendations-group-
id");
        configProps.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        configProps.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        return new DefaultKafkaConsumerFactory<>(configProps);
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, String>
kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String,String> factory = new
ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }
}

// MessageConsumer
package org.dtlk.analyticsandrecommendationservice.config;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.dtlk.analyticsandrecommendationservice.model.BookEvent;
import org.dtlk.analyticsandrecommendationservice.service.ConsumeEventsService;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class MessageConsumer {
    private final ConsumeEventsService consumeEventsService;

    @KafkaListener(topics = "books-event", groupId = "analytics-and-recommendations-group-id")
    public void listen(String message) {
        try {
            BookEvent event = new ObjectMapper().readValue(message, BookEvent.class);
```

```
            consumeEventsService.saveEvent(event);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }
}

// AnalyticsController
package org.dtlk.analyticsandrecommendationservice.controller;

import lombok.RequiredArgsConstructor;
import org.dtlk.analyticsandrecommendationservice.dto.AnalyticsDto;
import org.dtlk.analyticsandrecommendationservice.service.AnalyticsService;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(path = "/analytics")
@RequiredArgsConstructor
public class AnalyticsController {
    private final AnalyticsService analyticsService;

    @GetMapping
    @CrossOrigin
    public AnalyticsDto getAnalytics() {
        return analyticsService.createAnalytics();
    }
}

// AnalyticsDto
package org.dtlk.analyticsandrecommendationservice.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class AnalyticsDto {
    private Long numberOfBooks;
    private Long numberOfBookedBooks;
    private Long numberOfActiveUsers;
    private Long numberOfDeactivatedUsers;
}

// Analytics
package org.dtlk.analyticsandrecommendationservice.model;

import jakarta.persistence.*;
import lombok.Data;

@Data
@Entity
@Table(name = "analytics_data")
public class Analytics {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Long numberOfBooks;
    private Long numberOfBookedBooks;
    private Long numberOfActiveUsers;
    private Long numberOfDeactivatedUsers;
}

// BookEvent
package org.dtlk.analyticsandrecommendationservice.model;
```

```
import jakarta.persistence.*;
import lombok.Data;

@Data
@Entity
@Table(name = "book_events")
public class BookEvent {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String userId;
    private String bookId;
    private Long timestamp;
    private Long returnDate;
    private String status;
}

/*
{
    userId: userId,
    bookId: bookId,
    timestamp: time,
    returnDate: returnDate,
    status: status
}
 */

//Role
package org.dtlk.analyticsandrecommendationservice.model;

import jakarta.persistence.*;
import lombok.Data;

@Data
@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Enumerated(EnumType.STRING)
    @Column(nullable = false, unique = true)
    private RoleName name;

    public enum RoleName {
        ROLE_ADMIN,
        ROLE_USER
    }
}

//User
package org.dtlk.analyticsandrecommendationservice.model;

import jakarta.persistence.*;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;
import org.hibernate.annotations.SQLDelete;
import org.hibernate.annotations.Where;

import java.util.Set;

@Data
@Entity
@SQLDelete(sql = "UPDATE users SET is_deleted = true WHERE id = ?")
@Where(clause = "is_deleted = false")
@Table(name = "users")
public class User {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String email;
    @Column(nullable = false)
    private String password;
    @Column(name = "first_name", nullable = false)
    private String firstName;
    @Column(name = "last_name", nullable = false)
    private String lastName;
    @ManyToMany
    @JoinTable(
            name = "users_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    private Set<Role> roles;

    @Column(name = "is_deleted", nullable = false)
    private boolean isDeleted;

}

// AnalyticsRepository
package org.dtlk.analyticsandrecommendationservice.repository;

import org.dtlk.analyticsandrecommendationservice.model.Analytics;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AnalyticsRepository extends JpaRepository<Analytics, Long> {
}

// BookEventsRepository
package org.dtlk.analyticsandrecommendationservice.repository;

import org.dtlk.analyticsandrecommendationservice.model.BookEvent;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookEventsRepository extends JpaRepository<BookEvent, Long> {
}

// AnalyticsServiceImpl
package org.dtlk.analyticsandrecommendationservice.service;

import lombok.RequiredArgsConstructor;
import org.dtlk.analyticsandrecommendationservice.dto.AnalyticsDto;
import org.dtlk.analyticsandrecommendationservice.model.Analytics;
import org.dtlk.analyticsandrecommendationservice.model.BookEvent;
import org.dtlk.analyticsandrecommendationservice.repository.AnalyticsRepository;
import org.dtlk.analyticsandrecommendationservice.repository.BookEventsRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class AnalyticsServiceImpl implements AnalyticsService {
    private final AnalyticsRepository analyticsRepository;
    private final BookEventsRepository bookEventsRepository;

    @Override
    public AnalyticsDto createAnalytics() {
        List<BookEvent> events = bookEventsRepository.findAll();
        Analytics analytics = new Analytics();
```

```
        analytics.setNumberOfBooks(events.stream().map(BookEvent::getBookId).distinct().count());
        analytics.setNumberOfBookedBooks((long) getCurrentBookedBooks(events));

analytics.setNumberOfActiveUsers(events.stream().map(BookEvent::getUserId).distinct().count());
        analytics.setNumberOfDeactivatedUsers(0L);
        analyticsRepository.save(analytics);

        return new AnalyticsDto(analytics.getNumberOfBooks(),
                analytics.getNumberOfBookedBooks(),
                analytics.getNumberOfActiveUsers(),
                analytics.getNumberOfDeactivatedUsers());
    }

    private int getCurrentBookedBooks(List<BookEvent> events) {
        Set<String> bookedBooks = events.stream().filter(bookEvent ->
bookEvent.getStatus().equals("BOOKED")).map(BookEvent::getBookId).collect(Collectors.toSet());
        Set<String> returnedBooks = events.stream().filter(bookEvent ->
bookEvent.getStatus().equals("RETURNED")).map(BookEvent::getBookId).collect(Collectors.toSet());
        bookedBooks.removeAll(returnedBooks);
        return bookedBooks.size();
    }
}

// ConsumeEventsServiceImpl
package org.dtlk.analyticsandrecommendationservice.service;

import lombok.RequiredArgsConstructor;
import org.dtlk.analyticsandrecommendationservice.dto.AnalyticsDto;
import org.dtlk.analyticsandrecommendationservice.model.Analytics;
import org.dtlk.analyticsandrecommendationservice.model.BookEvent;
import org.dtlk.analyticsandrecommendationservice.repository.AnalyticsRepository;
import org.dtlk.analyticsandrecommendationservice.repository.BookEventsRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class ConsumeEventsServiceImpl implements ConsumeEventsService {
    private final AnalyticsRepository analyticsRepository;
    private final BookEventsRepository bookEventsRepository;

    @Override
    public void saveEvent(BookEvent event) {
        bookEventsRepository.save(event);
    }
}

// AnalyticsAndRecommendationServiceApplication
package org.dtlk.analyticsandrecommendationservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AnalyticsAndRecommendationServiceApplication {

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

// create-analytics-table
databaseChangeLog:
  - changeSet:
      id: create-analytics-table
      author: ivan-diatliuk
      changes:
        - createTable:
```

```
            tableName: analytics_data
            columns:
              - column:
                  name: id
                  type: bigint
                  autoIncrement: true
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: number_of_books
                  type: bigint
                  constraints:
                    nullable: false
              - column:
                  name: number_of_booked_books
                  type: bigint
                  constraints:
                    nullable: false
              - column:
                  name: number_of_active_users
                  type: bigint
                  constraints:
                    nullable: false
              - column:
                  name: number_of_deactivated_users
                  type: bigint

// create-book-events-table
databaseChangeLog:
  - changeSet:
      id: create-book-events-table
      author: ivan-diatliuk
      changes:
        - createTable:
            tableName: book-events
            columns:
              - column:
                  name: id
                  type: bigint
                  autoIncrement: true
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: user_id
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: book_id
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: timestamp
                  type: bigint
                  constraints:
                    nullable: false
              - column:
                  name: return_date
                  type: bigint
                  constraints:
                    nullable: false
              - column:
                  name: status
                  type: varchar(255)
                  constraints:
                    nullable: false
```

```
// create-users-table
databaseChangeLog:
  - changeSet:
      id: create-users-table
      author: ivan-diatliuk
      changes:
        - createTable:
            tableName: users
            columns:
              - column:
                  name: id
                  type: bigint
                  autoIncrement: true
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: email
                  type: varchar(255)
                  constraints:
                    nullable: false
                    unique: true
              - column:
                  name: password
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: first_name
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: last_name
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: is_deleted
                  type: boolean
                  defaultValueBoolean: false
                  constraints:
                    nullable: false
// create-roles-table
databaseChangeLog:
  - changeSet:
      id: create-roles-table
      author: ivan-diatliuk
      changes:
        - createTable:
            tableName: roles
            columns:
              - column:
                  name: id
                  type: bigint
                  autoIncrement: true
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: name
                  type: varchar(255)
                  constraints:
                    nullable: false
                    unique: true

// insert-roles-to-roles-table
databaseChangeLog:
```

```
  - changeSet:
      id: insert-roles-to-roles-table
      author: ivan-diatliuk
      changes:
        - insert:
            tableName: roles
            columns:
              - column:
                  name: name
                  value: "ROLE_ADMIN"
        - insert:
            tableName: roles
            columns:
              - column:
                  name: name
                  value: "ROLE_USER"

//db.changelog.file
databaseChangeLog:
  - include:
      file:  db/changelog/changes/00-create-analytics-table.yaml
  - include:
      file: db/changelog/changes/01-create-book-events-table.yaml
  - include:
      file: db/changelog/changes/02-create-users-table.yaml
  - include:
      file: db/changelog/changes/03-create-roles-table.yaml
  - include:
      file: db/changelog/changes/04-insert-roles-to-roles-table.yaml

//application
spring.application.name=analytics-and-recommendation-service
spring.datasource.url=jdbc:mysql://localhost:3306/analytics_db
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.datasource.username=root
spring.datasource.password=redblack0root
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.open-in-view=false
server.servlet.context-path=/api
spring.liquibase.enabled=false

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id=analytics-and-recommendations-group-id

//pom
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
       <modelVersion>4.0.0</modelVersion>
       <parent>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-starter-parent</artifactId>
              <version>3.2.4</version>
              <relativePath/> <!-- lookup parent from repository -->
       </parent>
       <groupId>org.dtlk</groupId>
       <artifactId>analytics-and-recommendation-service</artifactId>
       <version>0.0.1-SNAPSHOT</version>
       <name>analytics-and-recommendation-service</name>
       <description>analytics-and-recommendation-service</description>
       <properties>
              <java.version>17</java.version>
       </properties>
       <dependencies>
              <dependency>
                     <groupId>org.springframework.boot</groupId>
```

```
                <artifactId>spring-boot-starter-data-jpa</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
            </dependency>
            <dependency>
                <groupId>org.liquibase</groupId>
                <artifactId>liquibase-core</artifactId>
            </dependency>

            <dependency>
                <groupId>com.mysql</groupId>
                <artifactId>mysql-connector-j</artifactId>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
            <dependency>
                <groupId>org.springframework.kafka</groupId>
                <artifactId>spring-kafka</artifactId>
                <version>3.1.2</version>
            </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                    <configuration>
                        <excludes>
                            <exclude>
                                <groupId>org.projectlombok</groupId>
                                <artifactId>lombok</artifactId>
                            </exclude>
                        </excludes>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                    <configuration>

        <mainClass>com.baeldung.spring.kafka.KafkaApplication</mainClass>
                    </configuration>
                </plugin>
            </plugins>
        </build>
</project>


//Order service project
//WebClienConfig.java
package org.vntu.orderservice.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;
```

```
@Configuration
public class WebClientConfig {
    @Bean
    @LoadBalanced
    public WebClient.Builder webClientBuilder() {
        return WebClient.builder();
    }
}

//OrderController.java
package org.vntu.orderservice.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.vntu.orderservice.dto.OrderRequest;
import org.vntu.orderservice.service.OrderService;

@RestController
@RequestMapping("/orders")
@RequiredArgsConstructor
public class OrderController {
    private final OrderService orderService;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public String placeOrder(@RequestBody OrderRequest orderRequest) {
        orderService.placeOrder(orderRequest);
        return "Order Placed Successfully!";
    }

}

//InventoryResponseDTO.java
package org.vntu.orderservice.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class InventoryResponse {
    private String skuCode;
    private boolean isInStock;
}

//OrderLineItemDto.java
package org.vntu.orderservice.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderLineItemDto {
    private String skuCode;
}

//OrderRequest.java
package org.vntu.orderservice.dto;

import lombok.Data;
import org.vntu.orderservice.model.OrderLineItem;
```

```
import java.util.List;

@Data
public class OrderRequest {
    private List<OrderLineItemDto> orderLineItemDtos;
}

//Order.java
package org.vntu.orderservice.model;

import jakarta.persistence.*;
import lombok.Data;

import java.util.List;

@Entity
@Table(name = "orders")
@Data
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String orderNumber;
    private Long orderDateTimestamp;
    private Long returnDateTimestamp;

    @OneToMany(cascade = CascadeType.ALL)
    private List<OrderLineItem> orderLineItems;

}

//OrderLineItem.java
package org.vntu.orderservice.model;

import jakarta.persistence.*;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "order_line_items")
@Data
@NoArgsConstructor
public class OrderLineItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String skuCode;
}

//OrderRepository.java
package org.vntu.orderservice.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.vntu.orderservice.model.Order;

public interface OrderRepository extends JpaRepository<Order, Long> {
}

//OrderService.java
package org.vntu.orderservice.service.impl;

import com.zaxxer.hikari.util.ClockSource;
import lombok.RequiredArgsConstructor;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Service;
```

```
import org.springframework.web.reactive.function.client.WebClient;
import org.vntu.orderservice.dto.InventoryResponse;
import org.vntu.orderservice.dto.OrderLineItemDto;
import org.vntu.orderservice.dto.OrderRequest;
import org.vntu.orderservice.model.Order;
import org.vntu.orderservice.model.OrderLineItem;
import org.vntu.orderservice.repository.OrderRepository;
import org.vntu.orderservice.service.OrderService;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {
    private static final Long DEFAULT_ORDER_PERIOD = 172800000L;
    private static final String INVENTORY_SERVICE_URL = "http://inventory-
service/api/inventory/get";

    private final OrderRepository orderRepository;
    private final WebClient.Builder webClientBuilder;

    @Override
    public void placeOrder(OrderRequest orderRequest) {
        Order order = new Order();
        order.setOrderNumber(UUID.randomUUID().toString());
        order.setOrderDateTimestamp(ClockSource.MillisecondClockSource.CLOCK.currentTime0());
        order.setReturnDateTimestamp(order.getOrderDateTimestamp() + DEFAULT_ORDER_PERIOD);
        order.setOrderLineItems(orderRequest.getOrderLineItemDtos().stream()
                .map(this::toModel)
                .toList());

        List<String> skuCodes = order.getOrderLineItems().stream()
                .map(OrderLineItem::getSkuCode)
                .toList();

        //Call inventory service
        InventoryResponse[] inventoryResponses = webClientBuilder.build()
                .get()
                .uri(INVENTORY_SERVICE_URL, uriBuilder ->
                    uriBuilder.queryParam("skuCodes", skuCodes).build())
                .accept(MediaType.APPLICATION_JSON)
                .retrieve()
                .bodyToMono(InventoryResponse[].class)
                .block();

        boolean isInStock = false;
        if (inventoryResponses != null) {
            isInStock =
Arrays.stream(inventoryResponses).filter(InventoryResponse::isInStock).findAny().isPresent();
        }
        if (isInStock) {
            orderRepository.save(order);
        } else {
            throw new RuntimeException("Can't process the order since one of the books is not in
the Stock!");
        }
    }

    private OrderLineItem toModel(OrderLineItemDto orderLineItemDto) {
        return OrderLineItem.builder()
                .skuCode(orderLineItemDto.getSkuCode())
                .build();
    }
}
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ УПРАВЛІННЯ РОБОТОЮ БІБЛІОТЕКИ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ SPRING І ПЛАТФОРМИ APACHE KAFKA

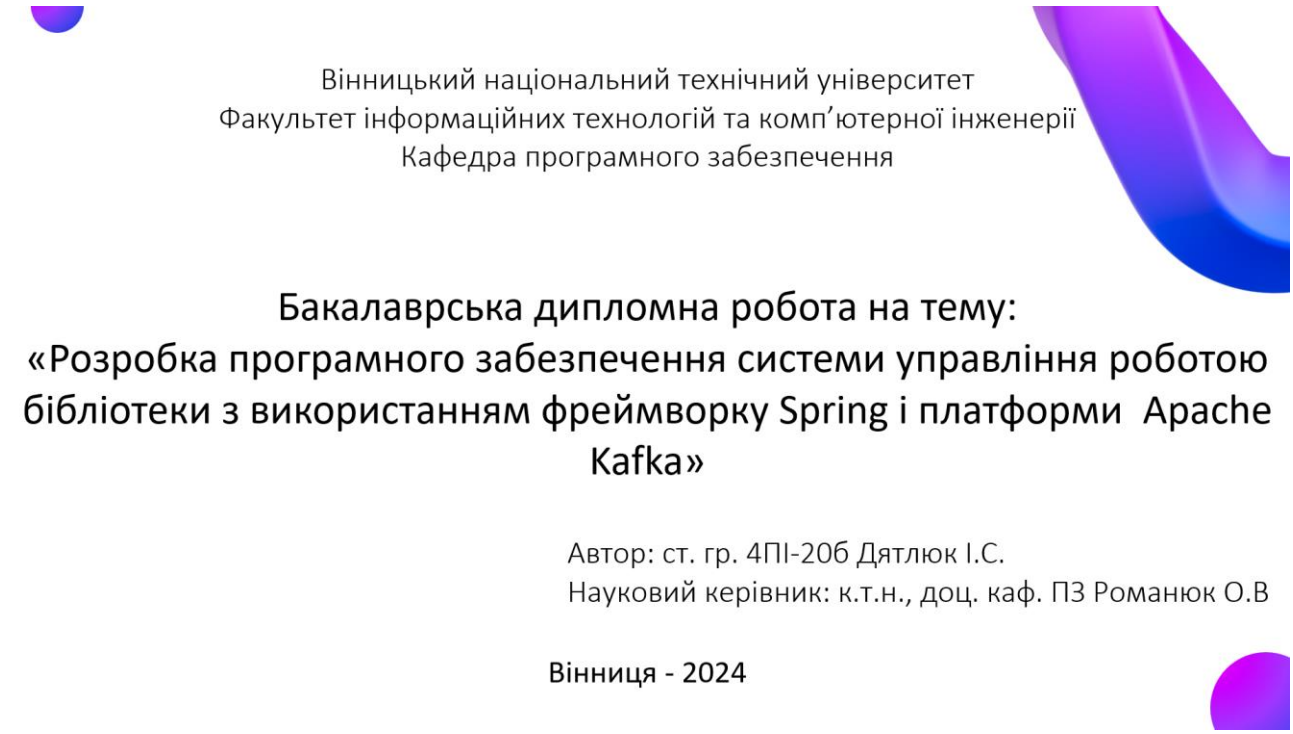

Рисунок Г.1 – Титульний слайд

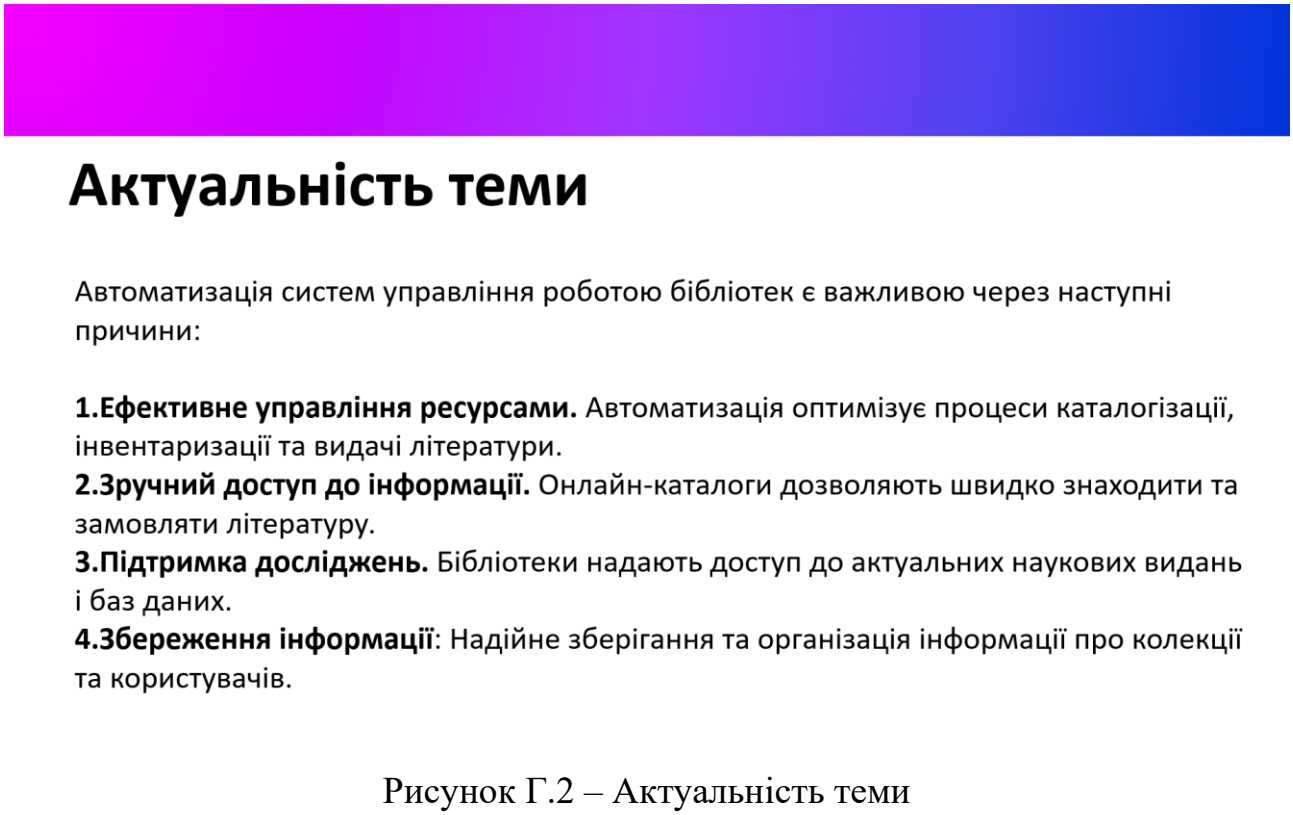

Рисунок Г.2 – Актуальність теми

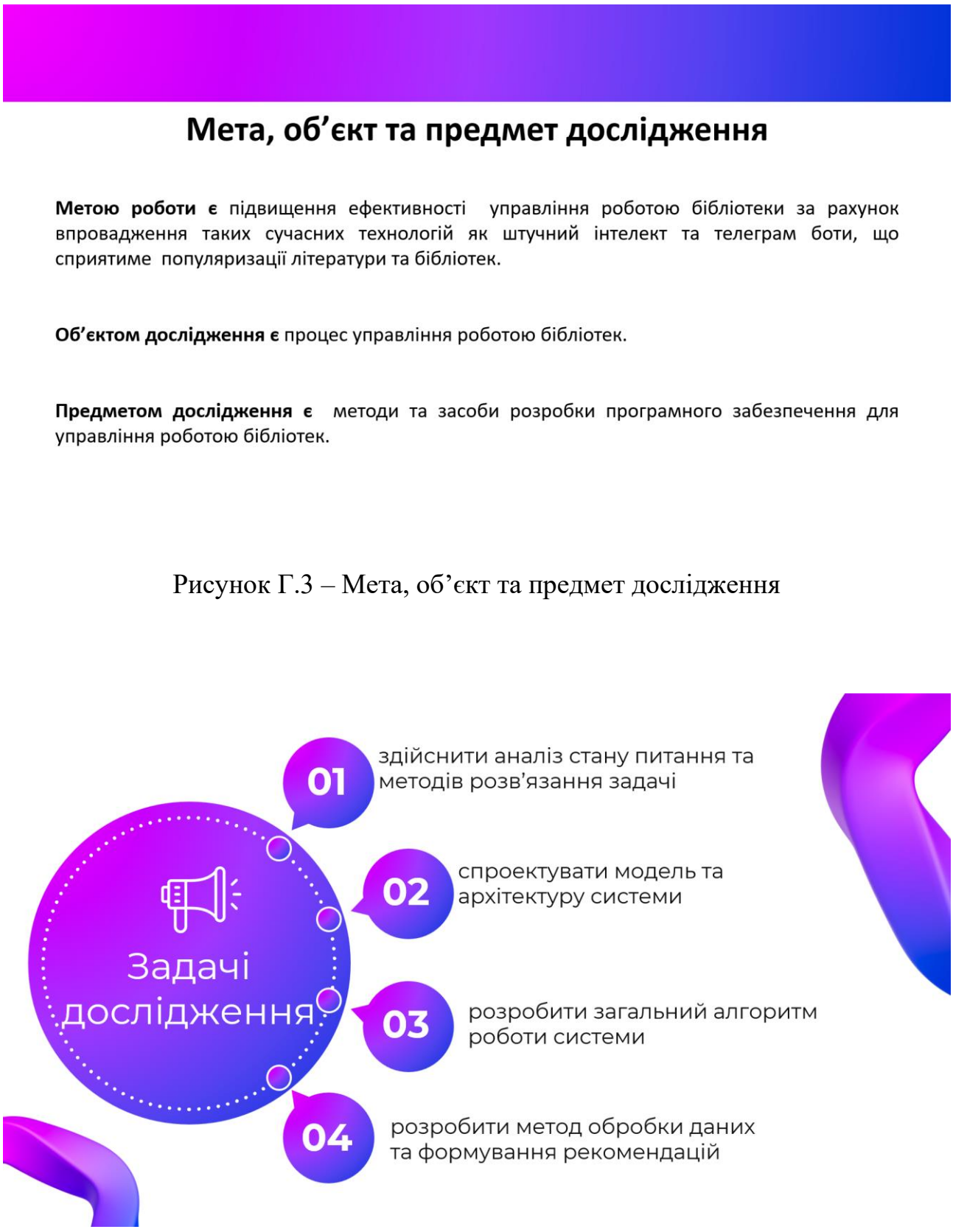

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Рисунок Г.4 – Задачі дослідження

Рисунок Г.5 – Задачі дослідження (продовження)

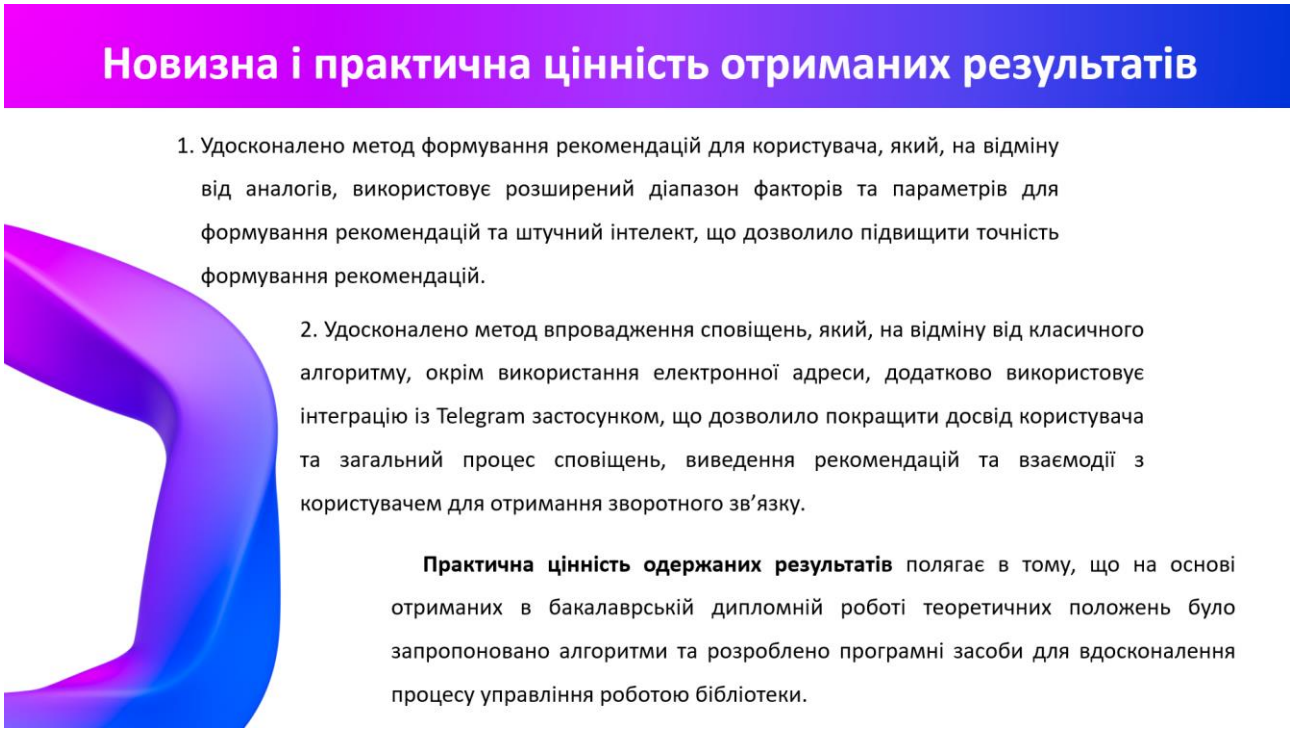

Рисунок Г.6 – Новизна і практична цінність отриманих результатів

Порівняльний аналіз аналогів

Критерії	Koha	FOLIO	Alma	Власна розробка
Наявність безкоштовної ліцензії	-	-	-	+
Розширена фільтрація книг	+/-	+	+	+
Управління користувачами	+	+	+	+
Підтримка цифрових ресурсів	+	+	+	-
Інтеграція сповіщень у чат ботах	-	-	-	+
Підсумок	1.5	3	3	4

Рисунок Г.7 – Порівняльний аналіз аналогів

Архітектура системи

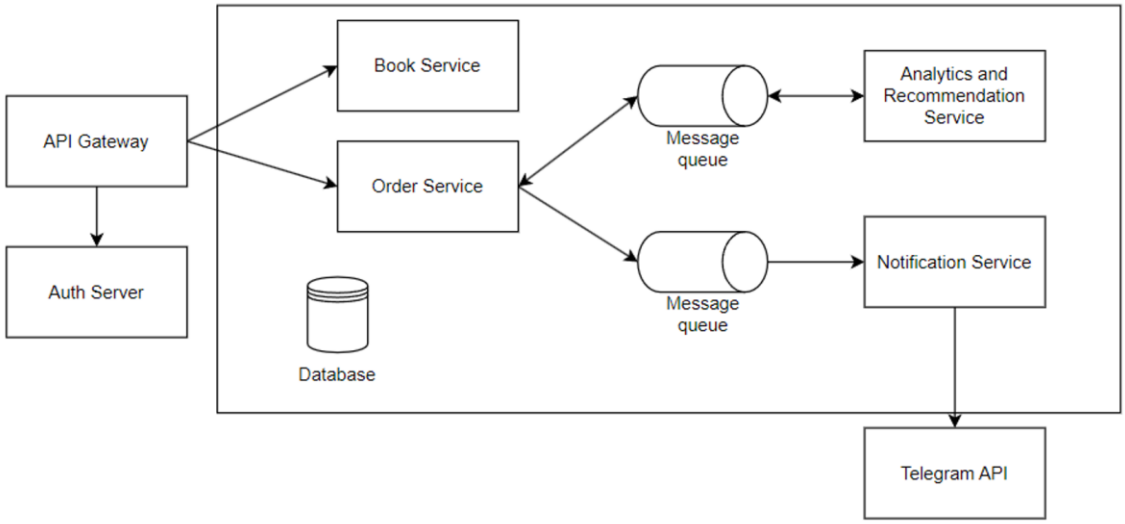

Рисунок 1 – Структурна діаграма автоматизованої системи

Рисунок Г.8 – Архітектура системи

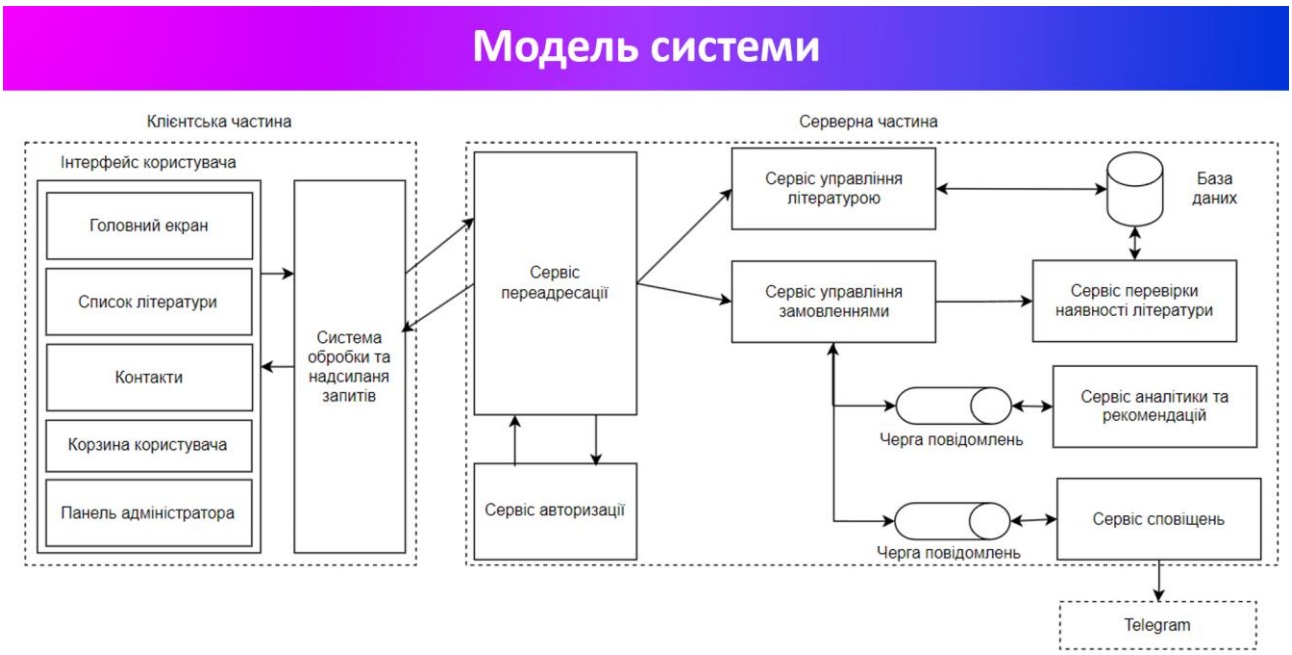

Рисунок 2 – Схема зображення моделі системи

Рисунок Г.9 – Модель системи

Рисунок Г.10 – Блок-схема загального алгоритму роботи системи

Розробка методу формування рекомендацій

1. Збір даних. Реалізація сховища даних – використовуємо Spring Data JPA для роботи з базою даних (MySQL) для зберігання явних та неявних даних про користувачів та книги. Парсинг даних про книги: Використовуємо бібліотеки для парсингу даних з описів книг.

2. Обробка та аналіз даних. Контентна фільтрація – використовуємо бібліотеки Java для обробки природної мови для виділення ключових слів та тем з описів книг, також використовуємо дані

3. Формування рекомендацій. Ранжування – сортуємо список рекомендацій за релевантністю, використовуючи різні метрики такі як: історія замовлень, жанри книг та автори.

5. Відображення рекомендацій. За допомогою черги повідомлень надсилаємо повідомлення до сервісу обробки замовлень, який в подальшому переадресує дані на сервіс сповіщень, який у свою чергу надішле повідомлення із рекомендаціями користувачу в Telegram.

Рисунок Г.11 – Розробка методу формування рекомендацій

Розробка методу взаємодії системи із Telegram ботом

Ініціалізація бота
Для забезпечення ефективної взаємодії з користувачами через Telegram бот було розроблено покроковий метод взаємодії. Перший крок включає ініціалізацію бота:

1.Реєстрація бота:
1. Бота було зареєстровано в Telegram через BotFather, що дозволило отримати унікальний токен доступу.
2. Це забезпечило автентифікацію бота та дозволило використовувати API Telegram.

2.Налаштування вебхуків:
1. Вебхуки були налаштовані для отримання оновлень від Telegram у режимі реального часу.
2. Це дозволило системі оперативно реагувати на дії користувачів.

3.Створення обробника оновлень:
1. На сервері було створено обробник оновлень, який реагує на отримані повідомлення від користувачів, забезпечуючи інтерактивну взаємодію.

Рисунок Г.12 – Розробка методу взаємодії системи із Telegram ботом

Розробка методу взаємодії системи із Telegram ботом

Авторизація користувача
Для безпечної та персоналізованої взаємодії з користувачами була впроваджена процедура авторизації:
1.Ініціація авторизації:
1. Користувач надсилає будь-яке повідомлення боту, що ініціює процес авторизації.
2. Бот отримує оновлення від Telegram про нове повідомлення.

2.Перевірка користувача:
1. Обробник оновлень на сервері перевіряє, чи є користувач зареєстрованим у системі.
2. Це дозволяє визначити, чи має користувач доступ до певних функцій та сповіщень.

Надсилання сповіщень
Для забезпечення своєчасного інформування користувачів про важливі події було розроблено систему сповіщень:

1.Генерація сповіщень:
1. Система управління бібліотекою генерує повідомлення про книги, такі як завершення терміну бронювання або нові рекомендації.

2.Обробка сповіщень:
1. Обробник сповіщень на сервері отримує ці повідомлення.
2. Система знаходить у базі даних Telegram ID користувачів, які повинні отримати сповіщення.

3.Надсилання повідомлень:
1. Бот надсилає сповіщення користувачам через Telegram API, забезпечуючи оперативне інформування.

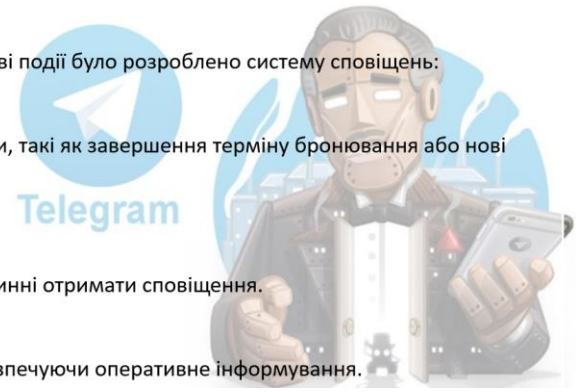

Рисунок Г.13 – Розробка методу взаємодії системи із Telegram ботом (продовження)

Розробка методу взаємодії системи із Telegram ботом

Обробка команд користувача
Для підвищення зручності користування ботом було впроваджено обробку команд:

1.Команди користувача:
1. Користувач може надсилати боту команди, такі як /start, /help, /settings.

2.Виконання команд:
1. Обробник оновлень розпізнає ці команди та виконує відповідні дії.
2. Це дозволяє користувачам налаштовувати сповіщення, отримувати інформацію про бота та інші функції, що підвищує зручність та функціональність системи.

Рисунок Г.14 – Розробка методу взаємодії системи із Telegram ботом (продовження)

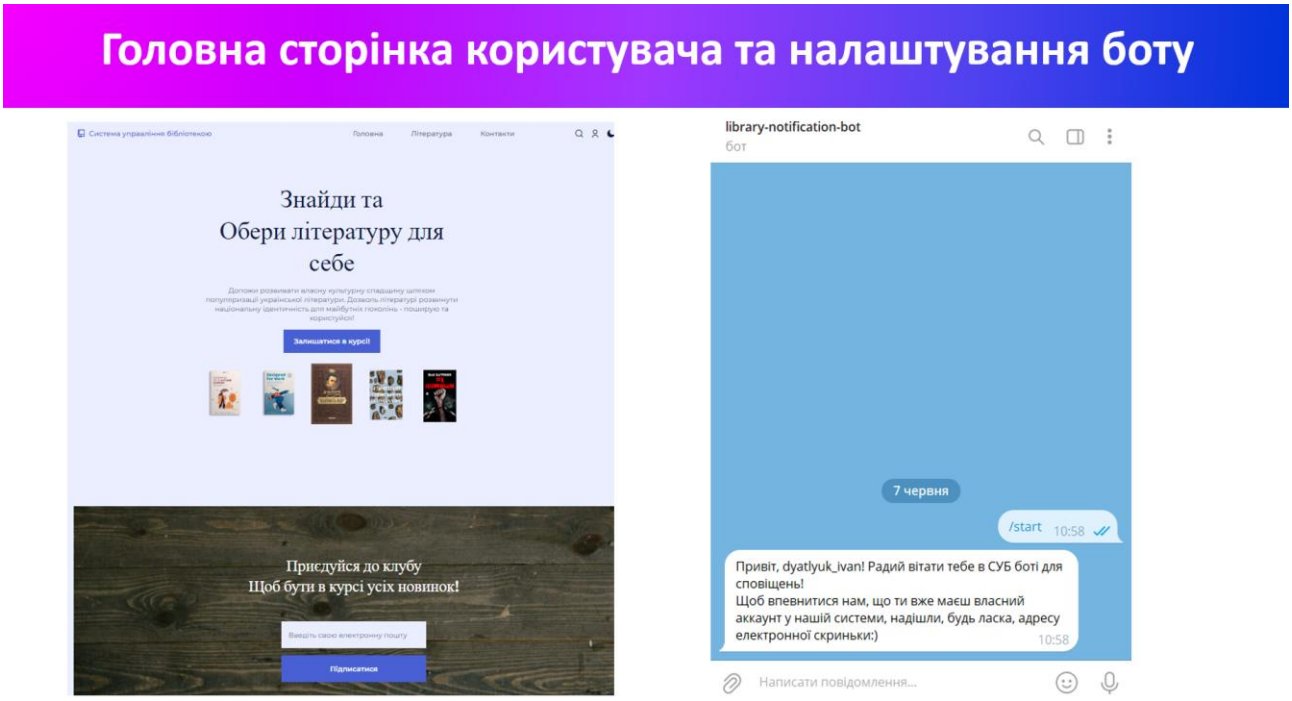

Рисунок Г.15 – Головна сторінка та налаштування боту

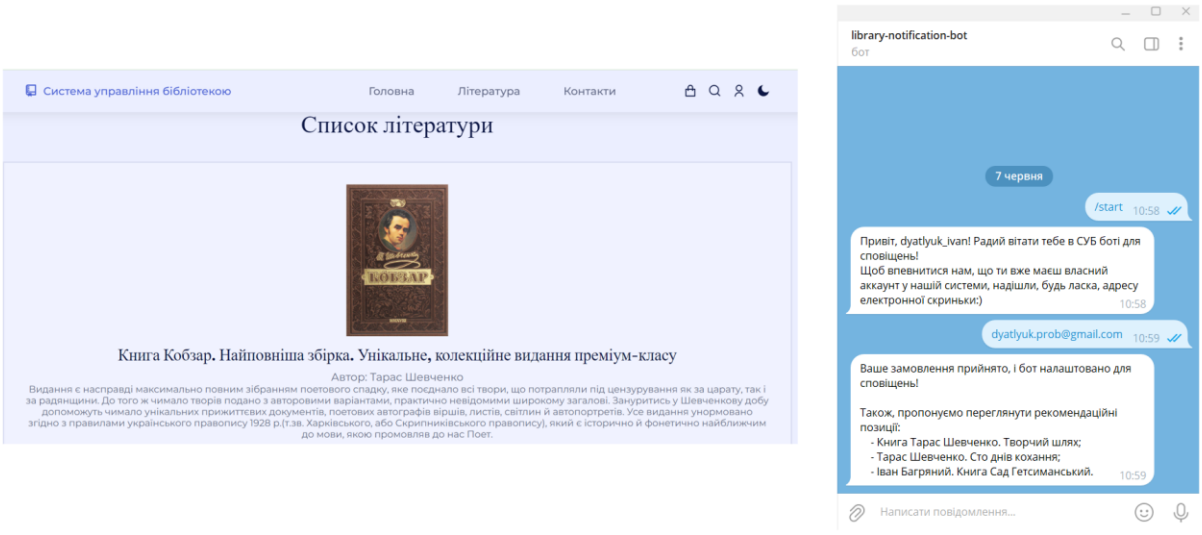

Рисунок Г.16 – Сторінка списку книг та отримання сповіщень

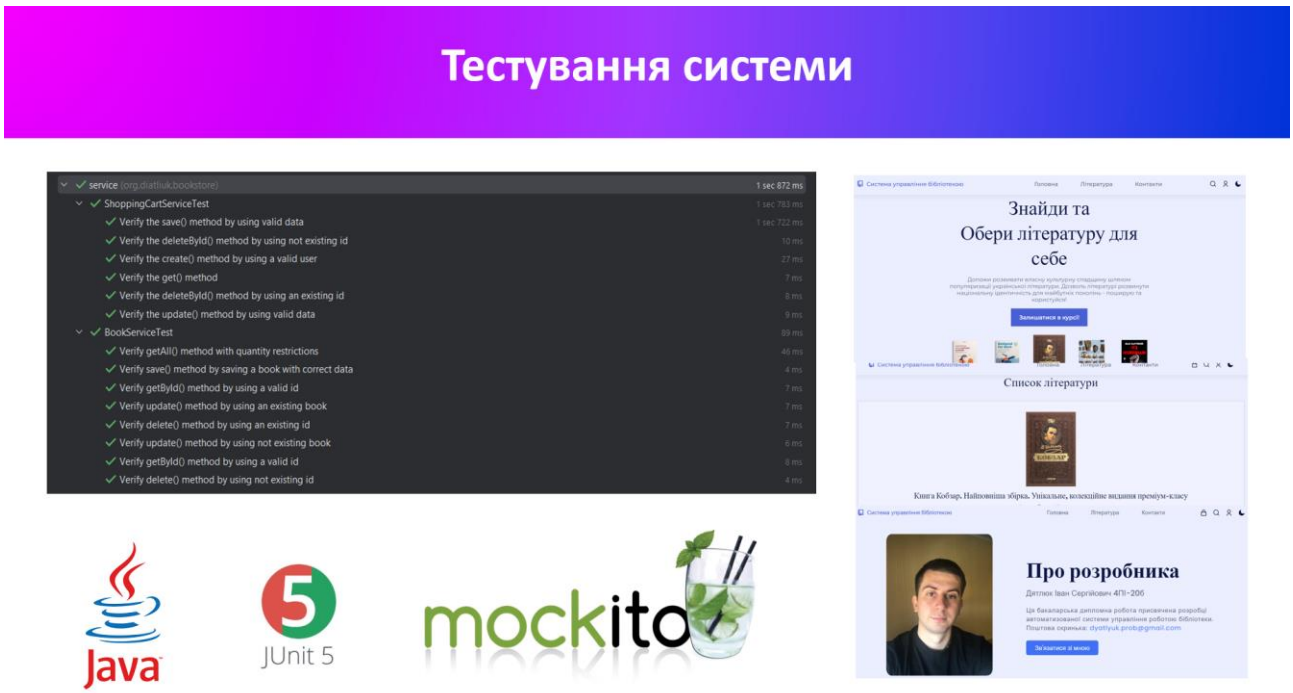

Рисунок Г.17 – Тестування системи

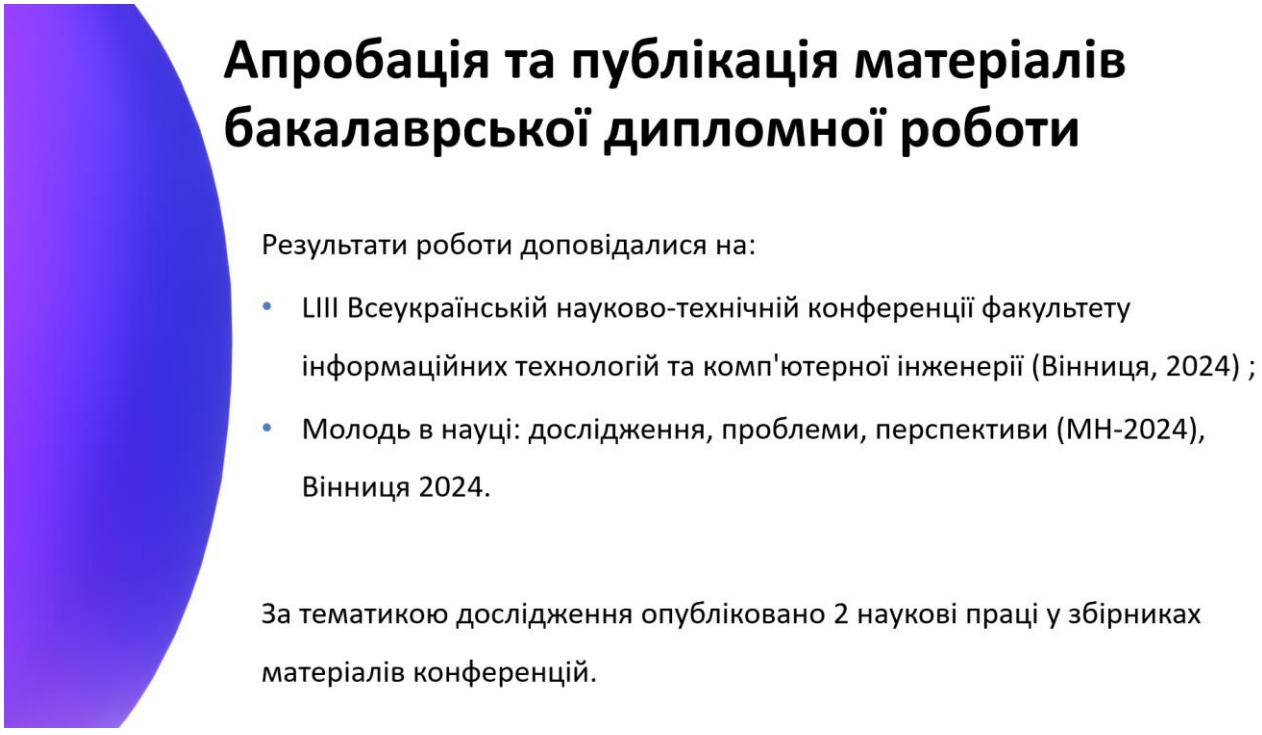

Рисунок Г.18 – Апробація та публікація матеріалів бакалаврської дипломної роботи

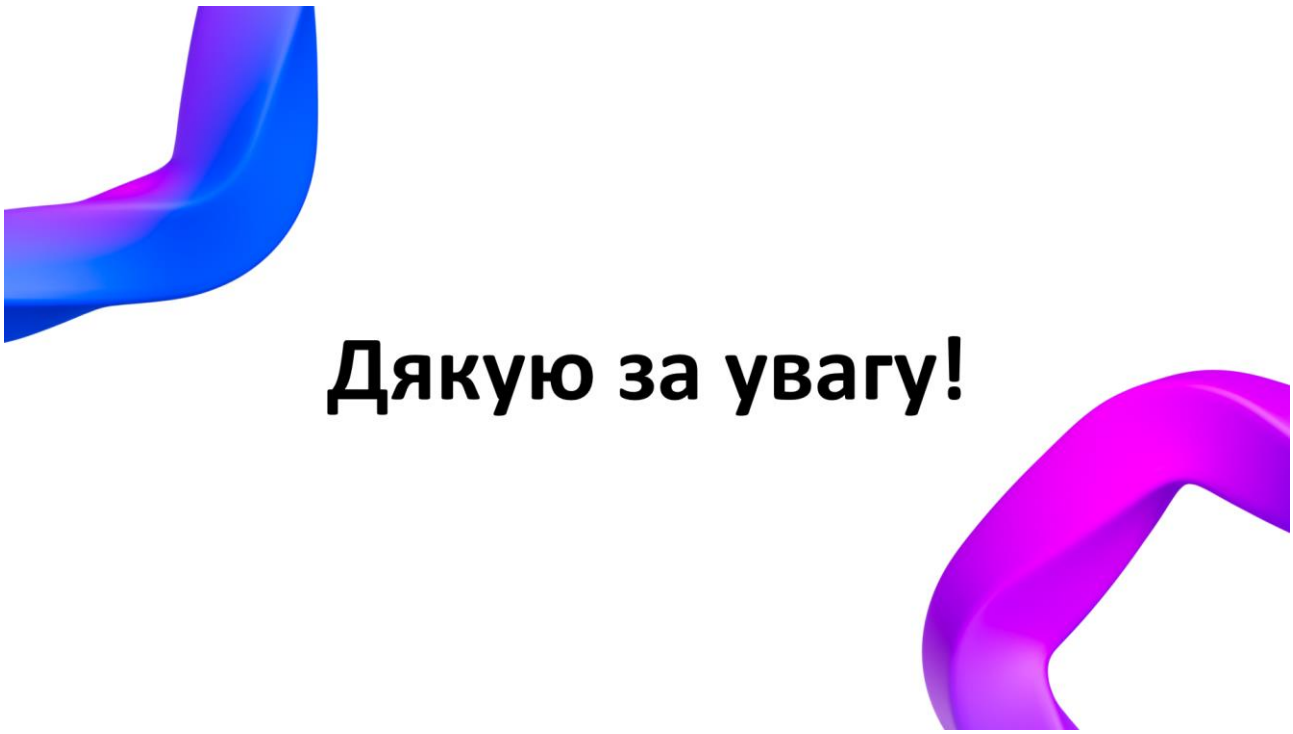

Рисунок Г.19 – Фінальний слайд