**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**

**ім. Ю.М. Потебні**

**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ**

**КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# Кваліфікаційна робота

перший (бакалаврський)

(рівень вищої освіти)

на тему **Дослідження засобів фреймворку Spring для організації роботи з персистентними даними**

Виконав: студент 4 курсу, групи 6.1212-пзс

спеціальності 121 Інженерія програмного забезпечення

(код і назва спеціальності)

освітньої програми Програмне забезпечення систем

(код і назва освітньої програми)

О.К. Понятенко

(ініціали та прізвище )

Керівник к.ф.-м.н., доцент, Г.П. Коломоєць

(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Рецензент директор ТОВ «Діскус» Р.О. Лютий

(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя

2026

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**

**ім. Ю.М. Потебні**

**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ**

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

**ЗАТВЕРДЖУЮ**
Завідувач кафедри_____Тетяна КРИТСЬКА
“ 01 ” березня 2026 року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ**

Понятенку Олександру Костянтиновичу
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження засобів фрейморку Spring для організації роботи з персистентними даними

   керівник роботи Коломоєць Геннадій Павлович, к. ф.-м. н., доцент,
   ( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом ЗНУ від 23.12.2025 р. № 1830-с
2. Строк подання студентом кваліфікаційної роботи 02.06.2026
3. Вихідні дані бакалаврської роботи
   - комплект нормативних документів ;
   - технічне завдання до роботи.
4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)
   - Огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - Огляд та аналіз існуючих рішень та аналогів;
   - Аналіз сучасних технологій для створення Веб-застосунків;
   - Проєктування та розробка Веб-застосунку;
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) слайдів презентації

6. Консультанти розділів бакалаврської роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата |
|---|---|---|
| | | Завдання прийняв |
| | | |

7. Дата видачі завдання 02.03.2026

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів магістерської роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 14.03.2026 | виконано |
| 2 | Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником | 20.03.2026 | виконано |
| 3 | Аналіз існуючих методів рішення | 25.03.2026 | виконано |
| 4 | Огляд та збір літератури стосовно теми кваліфікаційної роботи | 05.04.2026 | виконано |
| 5 | Огляд та аналіз існуючих рішень та аналогів | 08.04.2026 | виконано |
| 6 | Вивчення засобів для розробки Веб-застосунку | 12.04.2026 | виконано |
| 7 | Визначення функціоналу застосунку, що розробляється | 20.04.2026 | виконано |
| 8 | Узгодження подальших дій з науковим керівником | 21.04.2026 | виконано |
| 9 | Проєктування бази даних та створення проєкту застосунку | 27.04.2026 | виконано |
| 10 | Проєктування та конструювання застосунку | 11.05.2026 | виконано |
| 11 | Програмна реалізація застосунку та його апробація | 25.05.2026 | виконано |
| 12 | Оформлення звіту | 05.06.2026 | виконано |
| 13 | Оформлення презентації. Отримання рецензій від опонентів | 23.06.2026 | виконано |

Студент _____ О.К. Понятенко

Керівник роботи _____ Г. П. Коломоєць

**Нормоконтроль пройдено**

Нормоконтролер _____ _____ І.А. Скрипни

# АНОТАЦІЯ

Сторінок – 76

Рисунків – 9

Джерел – 9

Понятенко О. К. Дослідження засобів фрейморку Spring для організації роботи з персистентними даними: кваліфікаційна робота бакалавра спеціальності 121 Інженерія програмного забезпечення / наук. керівник Г. П. Коломоєць. Запоріжжя : ЗНУ, 2026, 76 с.

Бакалаврська робота присвячена дослідженню засобів фреймворку Spring для організації роботи з персистентними даними. У роботі розглянуто сучасні підходи до розробки програмного забезпечення, що взаємодіє з базами даних, а також проведено аналіз можливостей Spring Framework, Spring Data JPA та Hibernate.

Практична частина роботи полягає у створенні веб-застосунку «Бібліотека», який демонструє основні механізми роботи з персистентними даними. Застосунок забезпечує виконання операцій створення, пошуку, перегляду та видалення інформації про книги та авторів. Для реалізації проєкту використано Spring Boot, Spring Data JPA, Hibernate, H2 Database, HTML, CSS та JavaScript.

У процесі виконання роботи було досліджено принципи об'єктно-реляційного відображення, механізми автоматичного формування SQL-запитів та особливості використання патерну Repository. Отримані результати підтверджують ефективність використання засобів Spring для швидкої розробки інформаційних систем.

Ключові слова: Spring Framework, Spring Boot, Spring Data JPA, Hibernate, ORM, Repository, персистентні дані, база даних, веб-застосунок.

# ABSTRACT

Pages- – 76

Drawings – 9

Sources – 9

Ponyatenko O.K. Further investigation of the features of the Spring framework for organizing work with persistent data: qualification of a bachelor's degree in 121 Software Engineering / Science. kerivnik G. P. Kolomoets. Zaporizhzhya: ZNU, 2026, 76 p.

The bachelor's work is dedicated to exploring the features of the Spring framework for organizing work with persistent data. The work examines current approaches to developing software that interacts with databases, and also analyzes the capabilities of the Spring Framework, Spring Data JPA and Hibernate.

The practical part of the work is based on the created web site “Library”, which demonstrates the main mechanisms of work with persistent data. Zastosunka will be ensured by the new operation of creating, searching, reviewing and looking at other information about books and authors. To implement the project, we used Spring Boot, Spring Data JPA, Hibernate, H2 Database, HTML, CSS and JavaScript.

In the process of research, the principles of object-relational mapping, mechanisms for automatic generation of SQL queries, and the peculiarities of the Repository pattern were studied. The obtained results confirm the effectiveness of the Spring features for the rapid development of information systems.

Keywords: Spring Framework, Spring Boot, Spring Data JPA, Hibernate, ORM, Repository, persistent data, database, web site.

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1: ТЕОРЕТИЧНІ ОСНОВИ ОРГАНІЗАЦІЇ РОБОТИ З ПЕРСИСТЕНТНИМИ ДАНИМИ 12
1.1 Поняття персистентних даних 12
1.2 Системи керування базами даних 13
1.2 Технологія JDBC 14
1.3 Концепція ORM 14
1.5 Hibernate як ORM-фреймворк 15
1.6 Фреймворк Spring 16
1.7 Spring Boot 17
1.8 Spring Data JPA 17
1.9 Переваги використання Spring для роботи з персистентними даними 18
1.10 Висновки до розділу 1 19
РОЗДІЛ 2: АНАЛІЗ ЗАСОБІВ SPRING ДЛЯ РОБОТИ З ПЕРСИСТЕНТНИМИ ДАНИМИ ТА ПОСТАНОВКА ЗАДАЧІ 20
2.1 Особливості роботи сучасних інформаційних систем із базами даних 20
2.2 Аналіз можливостей Hibernate 20
2.3 Порівняння JDBC, Hibernate та Spring Data JPA 21
2.4 Обґрунтування вибору Spring Boot 22
Постановка задачі 23
Функціональні вимоги 23
Нефункціональні вимоги 23
2.5 Висновки до розділу 2 24
РОЗДІЛ 3 : ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ «БІБЛІОТЕКА» 25
3.1 Загальна характеристика системи 25
3.2 Архітектура програмного застосунку 25
Рівень представлення (Presentation Layer) 25
Рівень контролерів (Controller Layer) 26
Сервісний рівень (Service Layer) 26

Рівень доступу до даних (Repository Layer) 26
Рівень бази даних (Database Layer) 27
3.3 Структура проєкту 27
3.4 Проєктування REST API 30
API авторів 30
API книг 31
3.5 Проєктування веб-інтерфейсу 31
3.6 Проєктування механізму пошуку 32
3.7 Проєктування механізму початкового завантаження даних 33
3.7 Висновки до розділу 3 34
РОЗДІЛ 4: РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ 35
4.1 Конфігурація проєкту 35
4.2 Реалізація сутності Author 36
4.3 Реалізація сутності Book 39
4.4 Реалізація репозиторію AuthorRepository 41
4.5 Реалізація репозиторію BookRepository 42
4.6 Реалізація сервісного рівня 44
4.7 Реалізація контролера AuthorController 47
4.8 Реалізація контролера BookController 48
4.9 Реалізація механізму автоматичного завантаження даних 48
4.10 Реалізація механізму пошуку 62
4.11 Реалізація операції додавання автора 62
4.12 Реалізація операції додавання книги 63
4.13 Реалізація операції видалення книги 63
4.14 Використання Git та GitHub 63
4.15 Висновки до розділу 4 64
РОЗДІЛ 5: ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 66
5.1 Мета тестування 66
5.2 Тестування отримання списку авторів 66
5.3 Тестування отримання списку книг 68

5.4 Тестування додавання автора 69
5.5 Тестування додавання книги 70
5.6 Тестування пошуку за назвою книги 71
5.7 Тестування пошуку за автором 72
5.8 Тестування видалення книги 72
5.9 Аналіз переваг використання Spring Data JPA 73
5.10 Аналіз отриманих результатів 74
5.11 Висновки до розділу 5 74
ВИСНОВКИ 75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 76

# ВСТУП

**Актуальність теми**

У сучасних інформаційних системах значна частина функціональності пов'язана із збереженням та обробкою даних. Практично будь-який програмний продукт використовує базу даних для зберігання інформації про користувачів, товари, документи, фінансові операції або інші сутності предметної області.

Традиційний підхід до роботи з базами даних передбачає використання SQL-запитів та технології JDBC. Незважаючи на високу продуктивність такого підходу, він характеризується значним обсягом шаблонного коду та складністю підтримки великих програмних проєктів.

Для спрощення розробки програмного забезпечення були створені ORM-фреймворки, які забезпечують автоматичне відображення об'єктів мови програмування на таблиці бази даних. Одним із найбільш популярних рішень є Hibernate, який інтегрується з екосистемою Spring.

Особливий інтерес становить модуль Spring Data JPA, що дозволяє суттєво скоротити кількість програмного коду та прискорити процес створення застосунків. Саме тому дослідження можливостей Spring Framework для організації роботи з персистентними даними є актуальним завданням.

**Мета роботи**

Метою роботи є дослідження засобів фреймворку Spring для організації роботи з персистентними даними та створення демонстраційного веб-застосунку, який реалізує взаємодію з базою даних за допомогою Spring Data JPA.

**Завдання роботи**

Для досягнення поставленої мети необхідно виконати такі завдання:

- дослідити поняття персистентності даних;

- проаналізувати архітектуру Spring Framework;
- дослідити можливості Spring Data JPA;
- проаналізувати принципи роботи Hibernate;
- розробити структуру програмного застосунку;
- реалізувати веб-застосунок «Бібліотека»;
- реалізувати операції додавання, пошуку та видалення даних;
- провести тестування створеного програмного забезпечення.

**Об'єкт дослідження**

Об'єктом дослідження є процес організації роботи з персистентними даними в сучасних Java-застосунках.

**Предмет дослідження**

Предметом дослідження є засоби фреймворку Spring Framework, призначені для взаємодії з базами даних.

**Методи дослідження**

Під час виконання роботи використовувалися методи аналізу наукових джерел, порівняння програмних технологій, проєктування інформаційних систем та практичної реалізації програмного забезпечення.

**Практичне значення роботи**

Практичне значення роботи полягає у створенні веб-застосунку «Бібліотека», який демонструє використання сучасних засобів Spring Framework для роботи з базами даних. Отримані результати можуть бути використані під час вивчення дисциплін, пов'язаних із розробкою серверних застосунків на Java.

**Глосарій**

*SQL (Structured Query Language)* - мова структурованих запитів до баз даних, що використовується для створення, модифікації та отримання даних.

*JDBC (Java Database Connectivity)* - стандартний програмний інтерфейс Java для взаємодії з реляційними базами даних.

*ORM (Object-Relational Mapping)* - технологія відображення об'єктів програмного коду на таблиці реляційної бази даних.

*Spring Framework* - фреймворк для розробки Java-застосунків, який надає інструменти для створення масштабованих та підтримуваних програмних систем.

*Spring Data JPA* - модуль Spring Framework, що спрощує роботу з базами даних через використання JPA.

*JPA (Java Persistence API) - специфікація* Java для керування персистентністю даних та взаємодії з реляційними базами даних.

*Hibernate - ORM-фреймворк, який реалізує специфікацію JPA та забезпечує автоматичне відображення* Java-об'єктів на структури бази даних.

*Repository - програмний компонент, що забезпечує доступ до даних та виконання операцій над ними.*

*Entity* - клас Java, який відповідає таблиці бази даних і описує її структуру.

*CRUD (Create, Read, Update, Delete)* - базові операції створення, читання, оновлення та видалення даних.

*REST API (Representational State Transfer Application Programming Interface)* - архітектурний стиль взаємодії між клієнтом і сервером через HTTP-запити.

*MySQL* - реляційна система керування базами даних з відкритим вихідним кодом.

*PostgreSQL - об'єктно-реляційна система керування базами даних, що підтримує розширені можливості обробки даних.*

*Dependency Injection (DI)* - механізм передачі залежностей об'єктам з метою зменшення зв'язності компонентів системи.

*Bean* - об'єкт, життєвим циклом якого керує контейнер Spring Framework.

*Persistence Layer - рівень програмного забезпечення, відповідальний за збереження та отримання даних із бази даних.*

# РОЗДІЛ 1: ТЕОРЕТИЧНІ ОСНОВИ ОРГАНІЗАЦІЇ РОБОТИ З ПЕРСИСТЕНТНИМИ ДАНИМИ

## 1.1 Поняття персистентних даних

У сучасних інформаційних системах важливу роль відіграє збереження інформації між сеансами роботи програмного забезпечення. Дані, які залишаються доступними після завершення виконання програми або перезавантаження комп'ютера, називаються персистентними даними [https://www.manning.com/books/java-persistence-with-hibernate-second-edition].

Персистентність забезпечує довготривале зберігання інформації та можливість її подальшого використання. Найчастіше для реалізації механізму персистентності використовуються системи керування базами даних. Вони дозволяють організувати структуру зберігання інформації, забезпечують швидкий пошук, оновлення та видалення даних.

У більшості сучасних програмних систем персистентні дані представлені у вигляді таблиць, які містять записи про користувачів, товари, документи, фінансові операції, книги, авторів та інші об'єкти предметної області.

Основними перевагами використання персистентних даних є:

- довготривале зберігання інформації;
- забезпечення цілісності даних;
- можливість багатокористувацького доступу;
- підтримка резервного копіювання;
- швидкий пошук необхідної інформації.

Без використання механізмів персистентності сучасні інформаційні системи були б нездатні забезпечити надійне зберігання та обробку великих обсягів інформації.

## 1.2 Системи керування базами даних

Система керування базами даних (СКБД) являє собою програмний комплекс, призначений для створення, зберігання та обробки структурованих даних.

Основними функціями СКБД є:

- створення баз даних;
- управління структурою таблиць;
- виконання запитів;
- забезпечення цілісності інформації;
- контроль доступу користувачів;
- резервне копіювання даних.

Серед найбільш поширених СКБД можна виділити:

- MySQL;
- PostgreSQL;
- Oracle Database;
- Microsoft SQL Server;
- MariaDB;
- H2 Database.

Для невеликих навчальних та демонстраційних проєктів часто використовується H2 Database. Вона є легкою вбудованою базою даних, яка не потребує окремого встановлення та конфігурації [https://www.h2database.com].

Основною перевагою H2 є простота інтеграції зі Spring Boot, що дозволяє швидко створювати навчальні застосунки для роботи з персистентними даними.

## 1.3 Технологія JDBC

Першим стандартним механізмом взаємодії Java-застосунків із базами даних стала технологія JDBC (Java Database Connectivity) [https://docs.oracle.com/en/database/oracle/oracle-database/26/jjdbc/jdbc-developers-guide.pdf].

JDBC являє собою набір інтерфейсів та класів, які дозволяють програмісту встановлювати з'єднання з базою даних та виконувати SQL-запити.

Типова послідовність роботи через JDBC включає:

1. Встановлення з'єднання з базою даних.
2. Створення SQL-запиту.
3. Виконання запиту.
4. Отримання результатів.
5. Закриття з'єднання.

Перевагами JDBC є:

- висока продуктивність;
- повний контроль над SQL-запитами;
- універсальність.

Недоліками JDBC є:

- значний обсяг шаблонного коду;
- складність підтримки великих проєктів;
- необхідність ручного перетворення даних між об'єктами Java та таблицями бази даних.

Саме ці недоліки стали причиною появи ORM-технологій.

## 1.4 Концепція ORM

**ORM (Object-Relational Mapping)** — це технологія об'єктно-реляційного відображення, яка забезпечує взаємодію між об'єктами програми

та реляційною базою даних. Основна ідея ORM полягає у представленні таблиць бази даних у вигляді класів мови програмування, рядків таблиць — у вигляді об'єктів, а стовпців — у вигляді властивостей цих об'єктів. Це дозволяє працювати з даними на рівні об'єктів без необхідності написання великої кількості SQL-запитів вручну.[https://www.manning.com/books/java-persistence-with-hibernate]

Наприклад, таблиця Author може бути представлена класом Author, а таблиця Book — класом Book.

Переваги ORM:

- зменшення кількості SQL-коду;
- підвищення продуктивності розробки;
- спрощення підтримки проєкту;
- автоматичне формування SQL-запитів.

Завдяки ORM розробник працює з об'єктами Java, а не з таблицями бази даних напряму.

### 1.5 Hibernate як ORM-фреймворк

Hibernate є одним із найпоширеніших ORM-фреймворків для Java [https://hibernate.org/orm/]. Основне призначення Hibernate полягає у перетворенні Java-об'єктів у записи таблиць бази даних та навпаки.

Основні можливості Hibernate:

- автоматичне створення SQL-запитів;
- підтримка зв'язків між таблицями;
- кешування даних;
- управління транзакціями;
- автоматична генерація структури бази даних.

Для опису сутностей використовуються спеціальні анотації:

- @Entity;
- @Table;

- @Id;
- @GeneratedValue;
- @OneToMany;
- @ManyToOne.

Hibernate значно спрощує процес створення програмних систем, які працюють із базами даних.

### 1.6 Фреймворк Spring

Spring Framework є одним із найпопулярніших фреймворків для розробки корпоративних Java-застосунків [https://spring.io/projects/spring-framework]. Основною перевагою Spring є модульна структура, що дозволяє використовувати лише необхідні компоненти.

До складу Spring входять:

- Spring Core - забезпечує базові механізми фреймворку, зокрема керування залежностями (Dependency Injection) та контейнером компонентів.
- Spring MVC - забезпечує реалізацію архітектурного шаблону Model-View-Controller для розробки вебзастосунків.
- Spring Security - забезпечує автентифікацію, авторизацію та захист застосунків від поширених загроз безпеки.
- Spring Data - забезпечує спрощений доступ до баз даних і роботу з різними технологіями зберігання даних.
- Spring Boot - забезпечує швидке створення та налаштування Spring-застосунків завдяки автоматичній конфігурації та вбудованим серверам.
- Spring Cloud - забезпечує інструменти для розробки та керування розподіленими й мікросервісними системами.

Spring забезпечує:

- впровадження залежностей (Dependency Injection);

- керування життєвим циклом об'єктів;
- інтеграцію з базами даних;
- підтримку REST API;
- масштабованість застосунків.

Завдяки цим можливостям Spring став стандартом де-факто для розробки серверних Java-застосунків.

### 1.7 Spring Boot

Spring Boot є надбудовою над Spring Framework, яка спрощує конфігурацію та запуск застосунків [https://spring.io/projects/spring-boot].

Основні переваги Spring Boot:

- автоматична конфігурація;
- вбудований вебсервер;
- простота розгортання;
- інтеграція з Maven та Gradle;
- швидкий старт проєкту.

Розробнику достатньо додати необхідні залежності до файлу pom.xml, після чого більшість налаштувань буде виконана автоматично.

Це значно прискорює процес розробки програмного забезпечення.

### 1.8 Spring Data JPA

Spring Data JPA є спеціалізованим модулем Spring Framework для роботи з базами даних [https://spring.io/projects/spring-data-jpa]. Його основною метою є спрощення реалізації доступу до даних. Spring Data JPA використовує патерн Repository — шаблон проєктування, який відокремлює логіку доступу до даних від бізнес-логіки застосунку. Repository виступає посередником між програмою та базою даних, надаючи стандартні методи для створення, читання, оновлення та видалення даних (CRUD-операцій). Використання

цього патерну спрощує підтримку коду, підвищує його модульність і дозволяє зменшити обсяг коду, необхідного для роботи з базою даних.

Наприклад:

public interface BookRepository extends JpaRepository<Book, Long>

Після створення інтерфейсу репозиторію, який успадковує JpaRepository, автоматично стають доступними готові методи для роботи з даними без необхідності їх самостійної реалізації:

- save() - зберігає новий об'єкт або оновлює існуючий запис у базі даних;
- findAll() - повертає список усіх записів відповідної сутності;
- findById() - здійснює пошук запису за його ідентифікатором;
- deleteById() - видаляє запис із бази даних за його ідентифікатором;
- count() - повертає загальну кількість записів у таблиці.

Використання цих методів дозволяє значно скоротити обсяг програмного коду, спростити процес розробки та зменшити ймовірність виникнення помилок під час реалізації доступу до даних.

## 1.9 Переваги використання Spring для роботи з персистентними даними

Основними перевагами використання Spring Framework є:

- висока швидкість розробки;
- автоматизація рутинних операцій;
- інтеграція з Hibernate;
- підтримка транзакцій;
- масштабованість;
- висока продуктивність;
- підтримка сучасних архітектурних підходів.

Саме тому Spring Framework широко використовується для створення інформаційних систем різного рівня складності.

### 1.10 Висновки до розділу 1

У першому розділі було розглянуто теоретичні основи організації роботи з персистентними даними в сучасних інформаційних системах. Проаналізовано поняття персистентних даних, їх значення для довготривалого зберігання інформації та роль систем керування базами даних у забезпеченні надійності й цілісності даних.

Досліджено технологію JDBC як базовий механізм взаємодії Java-застосунків із базами даних, а також визначено її переваги та недоліки. Встановлено, що складність роботи з великим обсягом шаблонного коду стала передумовою розвитку технологій об'єктно-реляційного відображення (ORM).

Розглянуто принципи роботи ORM та особливості використання Hibernate як одного з найпоширеніших ORM-фреймворків для платформи Java. Також проаналізовано можливості Spring Framework, Spring Boot та Spring Data JPA, які забезпечують спрощення розробки програмного забезпечення, автоматизацію доступу до даних, підтримку транзакцій та інтеграцію з сучасними базами даних.

Проведений аналіз показав, що використання Spring Framework у поєднанні з Hibernate та Spring Data JPA дозволяє значно скоротити обсяг програмного коду, підвищити продуктивність розробки та спростити підтримку програмних систем. Отримані теоретичні відомості є основою для подальшого дослідження практичних аспектів організації роботи з персистентними даними засобами Spring Framework у наступних розділах роботи.

# РОЗДІЛ 2: АНАЛІЗ ЗАСОБІВ SPRING ДЛЯ РОБОТИ З ПЕРСИСТЕНТНИМИ ДАНИМИ ТА ПОСТАНОВКА ЗАДАЧІ

## 2.1 Особливості роботи сучасних інформаційних систем із базами даних

У процесі функціонування програмного забезпечення виникає необхідність постійного виконання операцій створення, читання, оновлення та видалення даних. Такі операції прийнято називати CRUD-операціями (Create, Read, Update, Delete).

У великих програмних системах кількість таких операцій може досягати десятків тисяч на добу. Саме тому важливим завданням є забезпечення швидкої, надійної та безпечної взаємодії між програмним кодом і базою даних.

Традиційно для вирішення цього завдання використовувалась технологія JDBC, яка дозволяє безпосередньо працювати із SQL-запитами. Однак зі збільшенням складності програмних продуктів виникла необхідність у створенні більш високорівневих інструментів доступу до даних.

Одним із таких рішень стали ORM-фреймворки та екосистема Spring Framework.

## 2.2 Аналіз можливостей Hibernate

Hibernate являє собою ORM-фреймворк, який автоматизує процес взаємодії Java-застосунку з реляційною базою даних.

Основний принцип роботи Hibernate полягає у відображенні таблиць бази даних на Java-класи.

Наприклад, таблиця Author може бути представлена класом Author, а таблиця Book — класом Book.

До недоліків Hibernate можна віднести:

- складність конфігурації у великих проєктах;
- необхідність вивчення специфічних механізмів ORM;

- певне зниження продуктивності порівняно з «чистим» JDBC.

Незважаючи на це, Hibernate сьогодні є стандартом для більшості Java-проєктів.

Spring Data JPA також підтримує створення власних методів пошуку:

```
List<Book> findByTitleContainingIgnoreCase(String title);
```

У результаті SQL-запит формується автоматично без необхідності його ручного програмування. Це відбувається завдяки механізму генерації запитів на основі назв методів репозиторію. Наприклад, якщо в інтерфейсі репозиторію визначено метод findByTitle(String title), Spring Data JPA аналізує його назву, визначає критерій пошуку за полем title та автоматично створює відповідний SQL-запит. Під час виконання програми виклик методу репозиторію перетворюється на JPQL-запит, який далі Hibernate трансформує у SQL-запит, сумісний з обраною системою керування базами даних. Завдяки цьому розробнику не потрібно самостійно створювати SQL-запити для типових операцій пошуку, додавання, оновлення та видалення даних.

## 2.3 Порівняння JDBC, Hibernate та Spring Data JPA

Для оцінки ефективності різних підходів було виконано порівняльний аналіз наведений в таблиці 2.1..

Таблиця 2.1 – *Порівняння технологій доступу до даних*

| Характеристика | JDBC | Hibernate | Spring Data JPA |
|---|---|---|---|
| Робота із SQL | Повністю вручну | Автоматизована | Автоматизована |
| ORM | Ні | Так | Так |
| CRUD | Вручну | Частково оптимізований | Повністю автоматизований |
| Складність розробки | Висока | Середня | Низька |
| Швидкість створення проекту | Низька | Середня | Висока |
| Підтримка Spring | Обмежена | Так | Повна |

На підставі проведеного аналізу можна зробити висновок, що Spring Data JPA є найбільш ефективним рішенням для створення сучасних інформаційних систем.

### 2.4 Обґрунтування вибору Spring Boot

Для реалізації практичної частини роботи було обрано Spring Boot, оскільки цей фреймворк забезпечує швидке створення, налаштування та запуск Java-застосунків завдяки автоматичній конфігурації та вбудованим серверам [https://spring.io/projects/spring-boot].

Основними причинами вибору стали:

- простота налаштування;
- автоматична конфігурація;
- інтеграція зі Spring Data JPA;
- підтримка REST API;
- вбудований сервер Tomcat;
- велика кількість документації.

Використання Spring Boot дозволяє сконцентруватися на реалізації бізнес-логіки замість виконання складних конфігураційних налаштувань.

Постановка задачі

На основі проведеного аналізу було сформульовано задачу розробки демонстраційного програмного застосунку, який дозволить дослідити можливості Spring Framework щодо роботи з персистентними даними.

У якості прикладу обрано веб-застосунок «Бібліотека».

Функціональні вимоги

До системи висуваються такі функціональні вимоги:

1. Перегляд списку авторів із відображенням їхніх імен та країн походження.
2. Перегляд списку книг із відображенням назви, року видання, авторів та країн їх походження.
3. Додавання нового автора із зазначенням імені та країни.
4. Редагування інформації про автора.
5. Додавання нової книги із зазначенням назви, року видання та авторів.
6. Редагування інформації про книгу.
7. Пошук книги за частиною назви без урахування регістру символів.
8. Пошук книги за частиною імені або прізвища автора без урахування регістру символів.
9. Видалення книги із системи.
10. Автоматичне завантаження початкових даних під час запуску застосунку для демонстрації роботи системи.
11. Відображення результатів пошуку у вигляді відфільтрованого списку книг.
12. Збереження інформації про книги, авторів та країни у базі даних.

Нефункціональні вимоги

До системи висуваються такі нефункціональні вимоги:

1. Застосунок повинен запускатися локально за допомогою Spring Boot.
2. Веб-інтерфейс повинен бути доступний через браузер.
3. Дані повинні зберігатися у базі даних H2 Database.
4. Взаємодія між клієнтською та серверною частиною повинна виконуватися через REST API.
5. Застосунок повинен підтримувати виконання основних CRUD-операцій для роботи з даними.
6. Програмний код повинен бути структурований за шарами: controller, entity, repository, service.
7. Проєкт повинен бути сумісний із Java 21 та Spring Boot 3.

## 2.5 Висновки до розділу 2

У результаті аналізу було досліджено сучасні підходи до організації роботи з персистентними даними. Встановлено, що використання Spring Data JPA дозволяє значно спростити реалізацію доступу до баз даних порівняно з JDBC та класичним Hibernate. На підставі проведеного дослідження було обрано Spring Boot як основний інструмент реалізації демонстраційного застосунку «Бібліотека», який використовується для практичного дослідження можливостей Spring Framework щодо роботи з персистентними даними.

# РОЗДІЛ 3 : ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ «БІБЛІОТЕКА»

## 3.1 Загальна характеристика системи

Практична частина бакалаврської роботи полягає у розробці веб-застосунку «Бібліотека», який демонструє використання засобів Spring Framework для роботи з персистентними даними.

Основною метою створення застосунку є демонстрація можливостей Spring Boot, Spring Data JPA та Hibernate при реалізації операцій збереження, пошуку, оновлення та видалення інформації з бази даних.

У якості предметної області обрано бібліотечну систему, яка містить інформацію про книги та авторів.

Система реалізована за клієнт-серверною архітектурою та складається з:

- веб-інтерфейсу користувача;
- серверної частини на Spring Boot;
- бази даних H2.

Користувач взаємодіє із застосунком через браузер, надсилаючи HTTP-запити до серверної частини. Сервер обробляє запити та здійснює взаємодію з базою даних за допомогою Spring Data JPA.

## 3.2 Архітектура програмного застосунку

Для реалізації проєкту використано багатошарову архітектуру.

Основними рівнями системи є:

**Рівень представлення (Presentation Layer)**

Відповідає за взаємодію з користувачем.

До його складу входять:

- HTML;
- CSS;
- JavaScript.

Основними функціями рівня представлення є:

- відображення списку книг;
- відображення авторів;
- введення пошукових запитів;
- додавання нових записів;
- видалення записів.

**Рівень контролерів (Controller Layer)**

Використовується для обробки HTTP-запитів.

У проєкті реалізовано:

- BookController;
- AuthorController.

Контролери приймають запити від клієнта та передають їх на сервісний рівень.

**Сервісний рівень (Service Layer)**

Сервісний рівень містить бізнес-логіку застосунку.

У даному проєкті використовується:

- BookService.

Основними завданнями сервісного рівня є:

- обробка запитів;
- пошук книг;
- перевірка коректності даних;
- взаємодія з репозиторіями.

**Рівень доступу до даних (Repository Layer)**

Рівень доступу до даних реалізований за допомогою Spring Data JPA.

До його складу входять:

- BookRepository;
- AuthorRepository.

Репозиторії забезпечують виконання CRUD-операцій без необхідності написання SQL-коду.

Рівень бази даних (Database Layer)

Для зберігання інформації використовується H2 Database.

База даних містить таблиці:

- authors;
- books.

### 3.3 Структура проєкту

Структура розробленого проєкту має вигляд:

```
src
├── main
│   ├── java
│   │   ├── controller
│   │   ├── entity
│   │   ├── repository
│   │   ├── service
│   │
│   └── resources
│       ├── static
│       └── application.properties
│
└── test
```

Подібна структура відповідає рекомендаціям Spring Boot [https://docs.spring.io/spring-boot/] та забезпечує простоту підтримки програмного коду.

### 3.4 Проєктування сутностей

Для реалізації предметної області веб-застосунку «Бібліотека» було виділено три основні сутності:

- Country;
- Author;
- Book.

Сутність Country призначена для зберігання інформації про країни, до яких належать автори. Вона містить такі поля:

| Поле | Тип | Опис |
|---|---|---|
| id | Long | первинний ключ |
| name | String | назва країни |

Сутність Author призначена для зберігання інформації про авторів книг. Вона містить такі поля:

| Поле | Тип | Опис |
|---|---|---|
| id | long | первинний ключ |
| name | String | ім’я автора |
| country_id | Long | зовнішній ключ на таблицю countries |

Кожен автор належить одній країні, а одна країна може бути пов’язана з багатьма авторами. Для реалізації цього зв’язку використовується відношення One-To-Many між сутностями Country та Author.

Сутність Book призначена для зберігання інформації про книги. Вона містить такі поля:

| Поле- | Тип | Опис |
|---|---|---|
| id | Long | первинний ключ |
| title | String | назва книги |
| Publish_year | Integer | рік видання книги |

Одна книга може мати декількох авторів, а один автор може бути пов’язаний із декількома книгами. Для реалізації такого зв’язку використовується відношення Many-To-Many між сутностями Book та Author.

Для реалізації зв’язку Many-To-Many у базі даних використовується проміжна таблиця book_authors, яка містить зовнішні ключі book_id та author_id. Такий підхід дозволяє коректно зберігати інформацію про книги з декількома авторами та про авторів, які написали декілька книг.

Модель зв’язків між сутностями має такий вигляд:

Country 1 — N Author

Author N — M Book

Для реалізації цих зв’язків у програмному коді використовуються анотації JPA:

@OneToMany

@ManyToOne

@ManyToMany

@JoinTable

Таким чином, розроблена модель даних відповідає реальній предметній області бібліотечної системи.

На рисунку 3.1 наведено ER-діаграму бази даних веб-застосунку «Бібліотека».

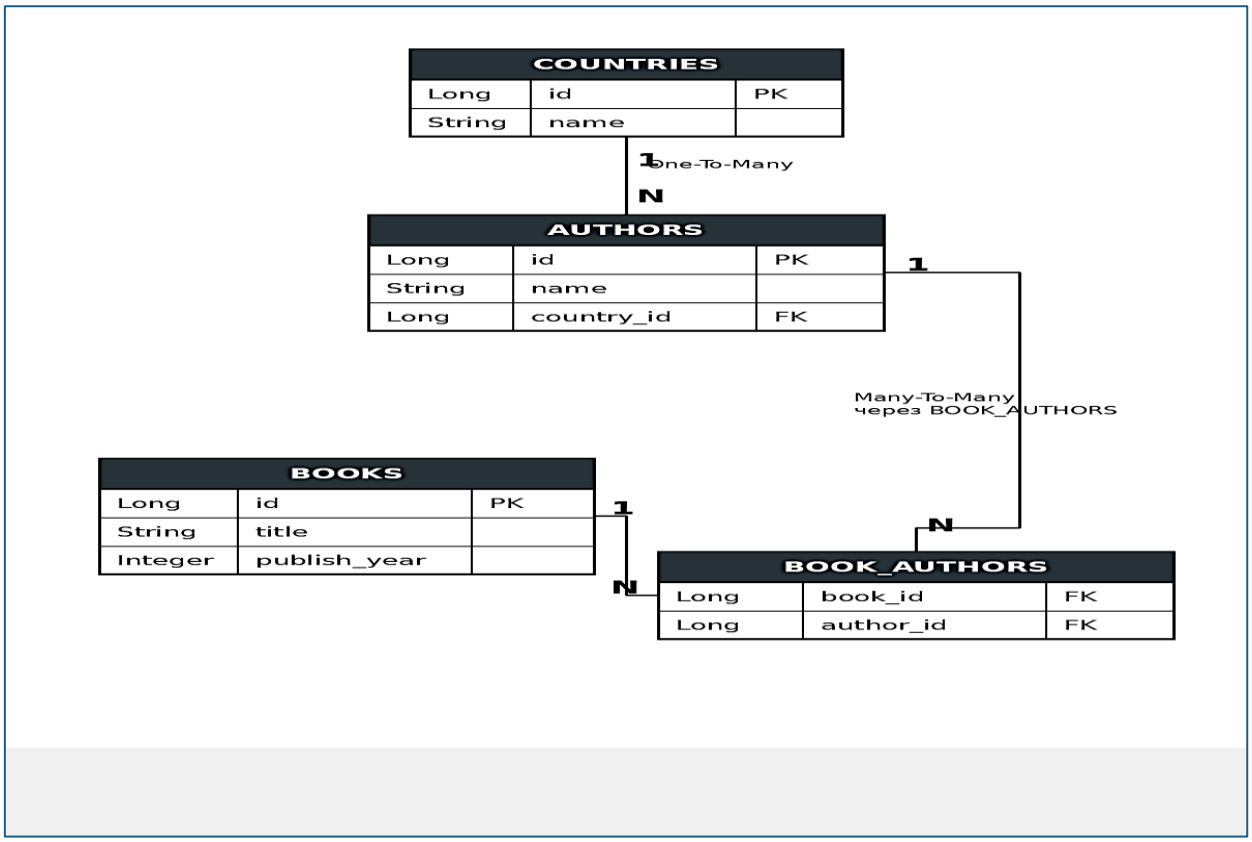



Рисунок *3.1 – ER-діаграма бази даних веб-застосунку «Бібліотека»*

Отже, спроєктована база даних забезпечує зберігання інформації про країни, авторів і книги, а також підтримує зв’язки One-To-Many та Many-To-Many між відповідними сутностями.

### 3.4 Проєктування REST API

Для взаємодії клієнтської частини із сервером використовується REST API.

API авторів

Отримати список авторів:

GET /authors

Додати автора:

POST /authors

Видалити автора:

DELETE /authors/{id}

API книг

Отримати список книг:

GET /books

Додати книгу:

POST /books

Видалити книгу:

DELETE /books/{id}

Пошук книги:

GET /books/search

### 3.5 Проєктування веб-інтерфейсу

Інтерфейс користувача розроблений із використанням HTML, CSS та JavaScript.

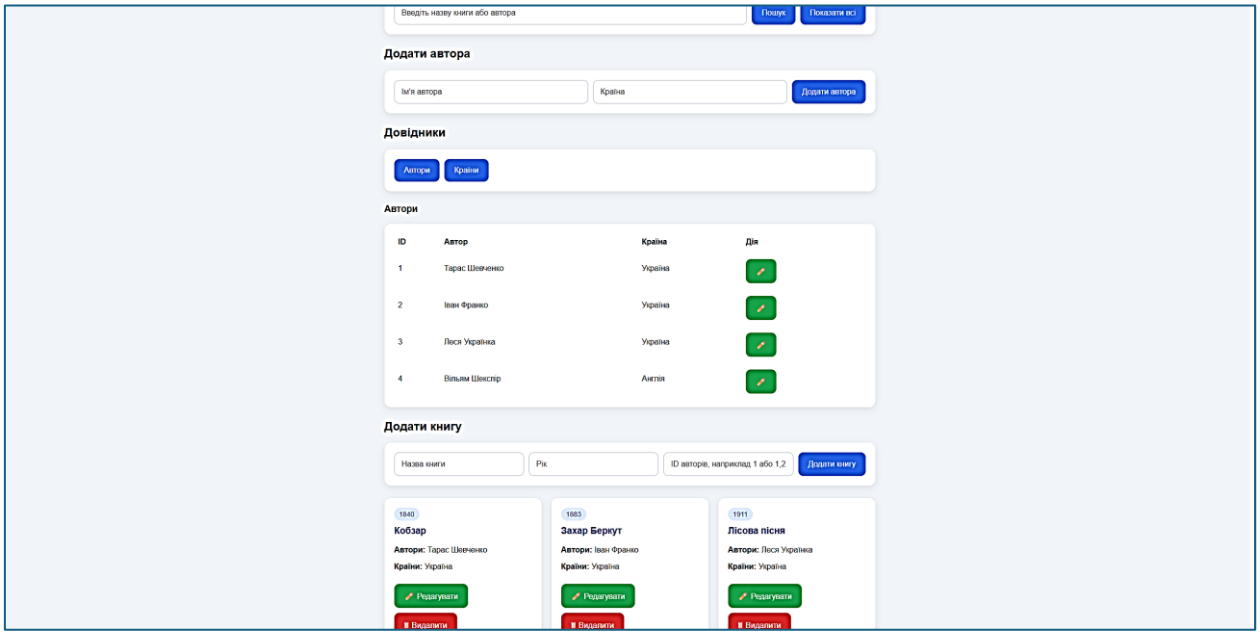



Рисунок 3.2 *Головна сторінка проєкту*

На головній сторінці розміщено (Рис. 3.2 ):

- поле введення для пошуку книг та авторів;
- кнопки **«Пошук»** та **«Показати всі»** для керування відображенням даних;
- форму додавання нового автора, що містить поля для введення імені автора та країни;
- таблицю авторів із відображенням ідентифікатора, імені автора та країни;
- кнопку редагування інформації про автора;
- форму додавання нової книги із зазначенням назви, року видання та авторів;
- картки книг із відображенням основної інформації про книгу;
- кнопку **«Редагувати»** для внесення змін до даних книги;
- кнопку **«Видалити»** для видалення книги з бази даних.

Основною вимогою до інтерфейсу була простота використання.

Користувач може виконувати всі операції без використання додаткових програм.

### 3.6 Проєктування механізму пошуку

Для пошуку використано Spring Data JPA та власний JPQL-запит.

Пошук виконується одночасно за повною або частиною назви книги та повним або частковим ім'ям автора. Реалізація пошуку використовує порівняння рядків без урахування регістру символів, що дозволяє знаходити книги навіть за фрагментом назви або імені автора (Лістинг 3.1:

Лістинг 3.1 – *Приклад JPQL - запиту*

```
    List<Book>
findByTitleContainingIgnoreCaseOrAuthorsNameContainingIgnor
eCase(
            String title,
            String authorName
    );
```

Таким чином користувач може вводити в одне поле пошуку як повну назву книги або її частину, так і повне ім'я автора або його фрагмент. У результаті система відобразить усі книги, що відповідають заданому критерію пошуку.

### 3.7 Проєктування механізму початкового завантаження даних

Для автоматичного наповнення бази даних використано компонент DataLoader. Даний клас імплементує інтерфейс CommandLineRunner, який входить до складу фреймворку Spring Boot. Після запуску застосунку Spring автоматично викликає перевизначений метод run(), у якому виконуються операції початкового заповнення бази даних. Для створення та збереження початкових записів використовуються методи репозиторіїв CountryRepository, AuthorRepository та BookRepository. Такий підхід дозволяє автоматично завантажувати тестові дані під час старту застосунку без необхідності ручного введення інформації користувачем.

Після запуску системи автоматично створюються:

- Тарас Шевченко;
- Іван Франко;
- Леся Українка;
- Вільям Шекспір.

Також додаються відповідні книги.

Це дозволяє демонструвати роботу застосунку без необхідності ручного заповнення бази даних.

### 3.7 Висновки до розділу 3

У даному розділі було виконано проєктування веб-застосунку «Бібліотека». Визначено архітектуру системи, структуру програмного проєкту, модель бази даних та REST API. Було спроєктовано сутності Country, Author та Book, визначено зв’язки One-To-Many і Many-To-Many між ними та розроблено ER-діаграму бази даних.

Також було виконано проєктування REST API шляхом визначення контролерів та кінцевих точок для роботи з авторами, книгами та країнами. Розроблені контролери забезпечують виконання основних CRUD-операцій, пошук книг за назвою та автором, а також взаємодію між клієнтською та серверною частинами застосунку.

Отримана архітектура забезпечує зручну організацію програмного коду, простоту підтримки та можливість подальшого розширення функціональних можливостей системи.

# РОЗДІЛ 4: РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ

## 4.1 Конфігурація проєкту

Для керування залежностями та автоматизації процесу складання проєкту використовується система керування проєктами Apache Maven. Maven забезпечує централізоване керування бібліотеками, автоматичне завантаження необхідних залежностей із віддалених репозиторіїв, компіляцію програмного коду та запуск застосунку.

Основним конфігураційним файлом Maven є **pom.xml** (Project Object Model), у якому визначаються параметри проєкту, версія Java, залежності та плагіни, необхідні для роботи застосунку.

У файлі pom.xml для проєкту було налаштовано використання Java 17 та підключено основні залежності Spring Boot:

- spring-boot-starter-web - для реалізації веб-застосунку та REST API;
- spring-boot-starter-data-jpa - для роботи з персистентними даними через JPA;
- h2 - вбудована реляційна база даних;
- spring-boot-starter-test - для тестування застосунку.

Також pom.xml містить батьківський проєкт Spring Boot Starter Parent, який забезпечує автоматичне налаштування сумісних версій бібліотек та плагінів. Використання Maven дозволяє спростити розробку, складання та підтримку програмного забезпечення.

Для реалізації програмного застосунку було обрано сучасний стек технологій, який забезпечує швидку розробку, зручність підтримки та можливість подальшого масштабування системи.

Під час розробки використовувалися:

- Java 21 (LTS-версія);
- Spring Boot 3;

- Spring Data JPA;
- Hibernate ORM;
- Maven;
- H2 Database;
- HTML5;
- CSS3;
- JavaScript;
- Git;
- GitHub;
- Visual Studio Code.

Мова програмування Java була обрана через її повну підтримку фреймворку Spring Boot, який використовується для розробки веб-застосунку. Java забезпечує платформонезалежність, надійність, розвинену екосистему бібліотек та засобів розробки, а також підтримує сучасні технології роботи з персистентними даними через Spring Data JPA та Hibernate ORM.

Spring Boot дозволив значно скоротити час розробки завдяки автоматичній конфігурації застосунку.

Для роботи з базою даних використовувався Spring Data JPA у поєднанні з Hibernate.

### 4.2 Реалізація сутності Author

Сутність Author призначена для зберігання інформації про авторів книг. Для опису сутності використано анотації JPA:

@Entity
@Table(name = "authors")

Анотація @Entity вказує, що клас є сутністю JPA та відповідає таблиці бази даних. Анотація @Table використовується для явного задання назви таблиці, з якою пов'язана сутність.

Скорочений фрагмент реалізації класу має такий вигляд (Лістинг 3.2):

Лістинг 3.2 – *Фрагмент реалізації класу Author*

```
@Entity
@Table(name = "authors")
public class Author {

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

    private String name;

    @ManyToOne
    @JoinColumn(name = "country_id")
    private Country country;

    @ManyToMany(mappedBy = "authors")
    @JsonIgnore
    private List<Book> books;
}
```

Поле id використовується як первинний ключ сутності. Для його позначення застосовано анотацію @Id:

@Id
private Long id;

Анотація @Id визначає поле, яке є первинним ключем таблиці. Для автоматичної генерації значення ідентифікатора використано анотацію:

@GeneratedValue(strategy = GenerationType.IDENTITY)

Завдяки цьому значення ключа створюється автоматично під час додавання нового автора до бази даних.

Поле name призначене для зберігання імені автора.

Для реалізації зв'язку між сутностями Author та Country використовується фрагмент коду:

```
    @ManyToOne
@JoinColumn(name = "country_id")
private Country country;
```

Анотація @ManyToOne визначає зв'язок типу «багато до одного», відповідно до якого декілька авторів можуть належати одній країні. Анотація @JoinColumn задає зовнішній ключ country_id, який використовується для зв'язку з таблицею countries.

Зворотний зв'язок між сутностями Country та Author реалізовано у класі Country таким чином:

```
    @OneToMany(mappedBy = "country")
@JsonIgnore
private List<Author> authors;
```

Анотація @OneToMany визначає зв'язок типу «один до багатьох», відповідно до якого одна країна може бути пов'язана з багатьма авторами. Параметр mappedBy = "country" вказує, що власником зв'язку є поле country у сутності Author.

Для реалізації зв'язку між авторами та книгами у класі Author використовується фрагмент:

```
    @ManyToMany(mappedBy = "authors")
@JsonIgnore
private List<Book> books;
```

Анотація @ManyToMany визначає зв'язок типу «багато до багатьох». Завдяки цьому один автор може бути пов'язаний із декількома книгами, а одна книга може мати декількох авторів.

Анотація @JsonIgnore використовується для виключення відповідного поля з процесу серіалізації JSON. Це необхідно для запобігання циклічним посиланням між пов'язаними сутностями під час формування відповідей REST API.

Отже, сутність Author не лише зберігає основну інформацію про автора, а й забезпечує зв'язок автора з країною та книгами в межах моделі даних веб-застосунку.

### 4.3 Реалізація сутності Book

Сутність Book призначена для представлення інформації про книги.

Для опису сутності використовуються анотації JPA:

```
    @Entity
@Table(name = "books")
```

Анотація @Entity вказує, що клас є сутністю JPA та відповідає таблиці бази даних. Анотація @Table використовується для задання назви таблиці books, з якою пов'язана сутність.

Скорочений фрагмент реалізації класу має такий вигляд (Лістинг 3.2):

Лістинг 3.2 – *Фрагмент реалізації класу Book*

```
    @Entity
@Table(name = "books")
public class Book {

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

    private String title;

    @Column(name = "publish_year")
    private Integer year;
```

```
    @ManyToMany
    @JoinTable(
        name = "book_authors",
        joinColumns = @JoinColumn(name = "book_id"),
        inverseJoinColumns = @JoinColumn(name =
"author_id")
    )
    @JsonIgnoreProperties("books")
    private List<Author> authors;
}
```

Поле id використовується як первинний ключ сутності.

Для позначення первинного ключа застосовується анотація:

@Id

private Long id;

Анотація @Id визначає поле, яке є первинним ключем таблиці.

Для автоматичної генерації значень ідентифікатора використовується анотація:

@GeneratedValue(strategy = GenerationType.IDENTITY)

Завдяки цьому значення ключа створюється автоматично під час додавання нового запису до бази даних.

Поле title зберігає назву книги. Поле year містить рік видання книги та відображається на стовпець publish_year таблиці бази даних за допомогою анотації:

```
    @Column(name = "publish_year")
```

Для реалізації зв'язку між книгами та авторами використовується анотація:

```
    @ManyToMany
@JoinTable(
    name = "book_authors",
    joinColumns = @JoinColumn(name = "book_id"),
```

```
    inverseJoinColumns = @JoinColumn(name = "author_id")
)
```

Анотація @ManyToMany визначає зв'язок типу «багато до багатьох», відповідно до якого одна книга може мати декількох авторів, а один автор може бути пов'язаний із декількома книгами.

Анотація @JoinTable використовується для створення проміжної таблиці book_authors, яка містить зовнішні ключі book_id та author_id і забезпечує реалізацію зв'язку Many-To-Many на рівні бази даних.

Для запобігання циклічній серіалізації JSON використовується анотація:

```
@JsonIgnoreProperties("books")
```

Вона дозволяє виключити поле books із серіалізації пов'язаних об'єктів Author під час формування відповідей REST API.

Таким чином, сутність Book забезпечує зберігання інформації про книги та реалізує зв'язок Many-To-Many між книгами й авторами через проміжну таблицю book_authors.

### 4.4 Реалізація репозиторію AuthorRepository

Для роботи з авторами створено репозиторій:

```
public interface AuthorRepository
extends JpaRepository<Author, Long>
```

Інтерфейс JpaRepository містить великий набір готових методів, а Spring Data надає для них універсальну реалізацію.

Серед них:

save()

findAll()

findById()

deleteById()

count()

Використання Spring Data JPA дозволяє уникнути написання SQL-запитів вручну.

Репозиторій автоматично створюється Spring Framework під час запуску застосунку.

### 4.5 Реалізація репозиторію BookRepository

Для роботи з книгами створено інтерфейс (Лістинг 3.3):

Лістинг 3.3 – *Створення інтерфейсу для роботи з книгами*

```
package com.example.demo.repository;
import com.example.demo.entity.Book;
import
org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
public interface BookRepository extends
JpaRepository<Book, Long> {
        List<Book> findByTitleContainingIgnoreCase(String
title);
        List<Book> findByYear(Integer year);
        List<Book>
findByAuthorsNameContainingIgnoreCase(String name);
        List<Book> findByAuthorsId(Long authorId);
        List<Book>
findByTitleContainingIgnoreCaseOrAuthorsNameContainingIgnor
eCase(
            String title,
            String authorName
    );
    }
```

Інтерфейс успадковує функціональність від:

JpaRepository<Book, Long>

де:

- Book — тип сутності, з якою працює репозиторій;
- Long — тип первинного ключа сутності.

Завдяки наслідуванню від JpaRepository автоматично стають доступними стандартні операції роботи з базою даних:

findAll();

findById();

save();

deleteById();

Для реалізації пошуку книг використовується метод:

```
List<Book>
findByTitleContainingIgnoreCaseOrAuthorsNameContainingIgnor
eCase(
        String title,
        String authorName
);
```

Spring Data JPA автоматично формує SQL-запит на основі назви методу. Пошук виконується одночасно за повною або частиною назви книги та повним або частковим ім'ям автора без урахування регістру символів.

Для отримання всіх книг конкретного автора використовується метод:

```
List<Book> findByAuthorsId(Long authorId);
```

Даний метод дозволяє знаходити книги за ідентифікатором автора через зв'язок Many-To-Many між сутностями Author та Book.

Таким чином, використання Spring Data JPA дозволило реалізувати доступ до даних без написання SQL-запитів вручну, що спрощує розробку та підтримку програмного забезпечення.

### 4.6 Реалізація сервісного рівня

Для відокремлення бізнес-логіки від контролерів використано сервісний шар. Основним компонентом є клас BookService, який відповідає за виконання операцій над книгами та взаємодію з репозиторіями.

Скорочений фрагмент реалізації класу має такий вигляд (Лістинг 3.4):

Лістинг 3.4 – *Фрагмент реалізації класу BookService*

```
    @Service
public class BookService {

    private final BookRepository bookRepository;
    private final AuthorRepository authorRepository;

    public BookService(BookRepository bookRepository,
                       AuthorRepository authorRepository) {
        this.bookRepository = bookRepository;
        this.authorRepository = authorRepository;
    }

    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    public Book getBook(Long id) {
        return bookRepository.findById(id)
                .orElseThrow(() -> new
RuntimeException("Book not found"));
    }

    public Book saveBook(Book book) {
```

```
        List<Author> authors = new ArrayList<>();

        for (Author authorFromRequest : book.getAuthors())
{
            Author author =
authorRepository.findById(authorFromRequest.getId())
                    .orElseThrow(() -> new
RuntimeException("Author not found"));
            authors.add(author);
        }

        book.setAuthors(authors);

        return bookRepository.save(book);
    }

    public Book updateBook(Long id, Book updatedBook) {
        Book book = getBook(id);

        book.setTitle(updatedBook.getTitle());
        book.setYear(updatedBook.getYear());

        if (updatedBook.getAuthors() != null) {
            List<Author> authors = new ArrayList<>();

            for (Author authorFromRequest :
updatedBook.getAuthors()) {
                Author author =
authorRepository.findById(authorFromRequest.getId())
                        .orElseThrow(() -> new
RuntimeException("Author not found"));
                authors.add(author);
            }
```

```
            book.setAuthors(authors);
        }

        return bookRepository.save(book);
    }

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

    public List<Book> search(String keyword) {
        return bookRepository

.findByTitleContainingIgnoreCaseOrAuthorsNameContainingIgno
reCase(
                        keyword,
                        keyword
                );
    }

    public List<Book> findByAuthor(Long authorId) {
        return bookRepository.findByAuthorsId(authorId);
    }
}
```

Клас BookService містить методи для отримання списку книг, пошуку книг, додавання нових книг, оновлення інформації про книги та видалення книг. Метод saveBook() використовується для створення нової книги та прив’язування до неї одного або декількох авторів. Для цього з отриманого запиту беруться ідентифікатори авторів, після чого відповідні записи завантажуються з бази даних через AuthorRepository.

Метод updateBook() реалізує оновлення даних книги. Він дозволяє змінити назву книги, рік видання та список авторів. Цей метод використовується в інтерфейсі користувача через кнопку редагування книги, після натискання якої користувач може змінити дані обраної книги.

Метод search() виконує пошук книг за повною або частковою назвою книги та за повним або частковим ім’ям автора. Для цього використовується метод репозиторію findByTitleContainingIgnoreCaseOrAuthorsNameContainingIgnoreCase, реалізація якого автоматично формується Spring Data JPA на основі назви методу.

Сервісний шар забезпечує відокремлення бізнес-логіки від контролерів, покращує підтримку програмного коду, спрощує повторне використання логіки та полегшує тестування застосунку.

### 4.7 Реалізація контролера AuthorController

Контролер AuthorController забезпечує роботу з авторами через REST API.

Основні маршрути:

Отримання всіх авторів:

GET /authors

Додавання автора:

POST /authors

Видалення автора:

DELETE /authors/{id}

Контролер приймає HTTP-запити та передає їх до рівня доступу до даних.

Усі результати повертаються у форматі JSON.

### 4.8 Реалізація контролера BookController

Контролер BookController відповідає за роботу з книгами.

Підтримуються такі операції:

GET /books

POST /books

DELETE /books/{id}

GET /books/search

Контролер реалізує взаємодію між клієнтською частиною та сервісним рівнем.

Завдяки використанню Spring MVC маршрути формуються за допомогою анотацій.

### 4.9 Реалізація механізму автоматичного завантаження даних

Клієнтська частина веб-застосунку реалізована у файлі index.html, який розміщений у каталозі src/main/resources/static. Сторінка створена з використанням HTML5, CSS3 та JavaScript. HTML відповідає за структуру сторінки, CSS — за зовнішній вигляд, а JavaScript — за взаємодію з REST API серверної частини.

HTML-сторінка складається з таких основних блоків:

- заголовок сторінки з назвою веб-застосунку;
- блок пошуку книг;
- форма додавання автора;
- блок довідників для перегляду авторів і країн;
- форма додавання книги;

- область відображення списку книг.

Фрагмент HTML-коду форми пошуку має такий вигляд:

```
<div class="search-box">
<input type="text" id="searchInput"
placeholder="Введіть назву книги або автора">
<button onclick="searchBooks()">Пошук</button>
<button onclick="loadBooks()">Показати всі</button>
</div>
```

У цьому фрагменті поле searchInput використовується для введення назви книги або імені автора. Кнопка «Пошук» викликає функцію searchBooks(), а кнопка «Показати всі» викликає функцію loadBooks().

Для додавання книги використовується форма (Лістинг 3.5):

Лістинг 3.5 – *Реалізація форми для додавання книги*

```
<div class="search-box">
<input type="text" id="bookTitle" placeholder="Назва
книги">
<input type="number" id="bookYear" placeholder="Рік">
<input type="text" id="bookAuthorIds" placeholder="ID
авторів, наприклад 1 або 1,2,3">
<button onclick="addBook()">Додати книгу</button>
</div>
```

Особливістю цієї форми є можливість вказати декількох авторів через кому, наприклад 1,2,3. Це дозволяє перевірити роботу зв’язку Many-To-Many між книгами та авторами.

Для перемикання між списком авторів і списком країн використовується блок довідників (Лістинг 3.6):

Лістинг 3.6 – *Фрагмент реалізації класу search-box*

```
<div class="search-box">
<button onclick="showAuthorsBlock()">Автори</button>
<button onclick="showCountriesBlock()">Країни</button>
</div>
```

Функція loadBooks() завантажує список книг із серверної частини:

```
async function loadBooks() {
const response = await fetch('/books');
const books = await response.json();
showBooks(books);
}
```

У цій функції виконується HTTP-запит GET /books. Отримані у форматі JSON дані передаються до функції showBooks(), яка відповідає за відображення книг на сторінці.

Функція searchBooks() виконує пошук книг:

```
async function searchBooks() {
const keyword =
document.getElementById("searchInput").value;

const response = await
fetch('/books/search?keyword=' +
encodeURIComponent(keyword));
const books = await response.json();

showBooks(books);
}
```

Пошук виконується за повною або частковою назвою книги та за повним або частковим ім’ям автора. Для безпечної передачі введеного тексту в URL використовується функція encodeURIComponent().

Функція addAuthor() додає нового автора:

```
async function addAuthor() {
    const name =
document.getElementById("authorName").value;
    const countryName =
document.getElementById("authorCountry").value;

    const response = await fetch("/authors", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            name: name,
            country: {
                name: countryName
            }
        })
    });

    if (response.ok) {
        alert("Автор доданий");
        location.reload();
    } else {
        alert("Помилка при додаванні автора");
    }
}
```

У цій функції формується POST-запит до кінцевої точки /authors. Дані автора передаються у форматі JSON.

Функція addBook() додає нову книгу:

```
async function addBook() {
    const title =
document.getElementById("bookTitle").value;
```

```
    const year =
parseInt(document.getElementById("bookYear").value);
    const authorIdsText =
document.getElementById("bookAuthorIds").value;

    const authors = authorIdsText
        .split(",")
        .map(id => id.trim())
        .filter(id => id !== "")
        .map(id => ({ id: parseInt(id) }));

    const response = await fetch("/books", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            title: title,
            year: year,
            authors: authors
        })
    });

    if (response.ok) {
        alert("Книга додана");
        location.reload();
    } else {
        alert("Помилка при додаванні книги");
    }
}
```

У цій функції рядок з ідентифікаторами авторів розділяється за комами за допомогою методу split(","). Далі кожне значення очищується від зайвих пробілів, перетворюється на число і передається до серверної частини як

список авторів. Завдяки цьому користувач може додати книгу з одним або кількома авторами.

Функція editBook() використовується для редагування книги:

```
async function editBook(id, oldTitle, oldYear,
oldAuthorIds) {
    const newTitle = prompt("Нова назва книги:", oldTitle);
    if (newTitle === null) return;

    const newYear = prompt("Новий рік:", oldYear);
    if (newYear === null) return;

    const newAuthorIds = prompt("ID авторів через кому:",
oldAuthorIds);
    if (newAuthorIds === null) return;

    const authors = newAuthorIds
        .split(",")
        .map(id => id.trim())
        .filter(id => id !== "")
        .map(id => ({ id: parseInt(id) }));

    const response = await fetch(`/books/${id}`, {
        method: "PUT",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            title: newTitle,
            year: parseInt(newYear),
            authors: authors
        })
    });
```

```
    if (response.ok) {
        alert("Книгу оновлено");
        loadBooks();
    } else {
        alert("Помилка редагування");
    }
}
```

Ця функція виконує PUT-запит до кінцевої точки /books/{id} і дозволяє змінити назву книги, рік видання та список авторів.

Функція deleteBook() використовується для видалення книги:

```
async function deleteBook(id) {
if (!confirm("Видалити книгу?")) {
    return;
}

const response = await fetch(`/books/${id}`, {
    method: "DELETE"
});

if (response.ok) {
    alert("Книгу видалено");
    loadBooks();
} else {
    alert("Помилка при видаленні книги");
}
}
```

Перед видаленням користувач повинен підтвердити дію. Після успішного видалення список книг оновлюється.

Функція showBooks() відповідає за виведення списку книг на сторінку:

```
function showBooks(books) {
const result = document.getElementById('result');
```

```
    result.innerHTML = '';

    if (!books || books.length === 0) {
        result.innerHTML = '<div class="empty">Книги не
знайдено</div>';
        return;
    }

    books.forEach(book => {
        const authorsNames = book.authors
            ? book.authors.map(author =>
author.name).join(", ")
            : "Немає авторів";

        const countries = book.authors
            ? [...new Set(
                book.authors
                    .map(author => author.country ?
author.country.name : "")
                    .filter(country => country !== "")
            )].join(", ")
            : "";

        result.innerHTML += `
            <div class="book">
                <span class="tag">${book.year}</span>
                <h3>${book.title}</h3>
                <p><b>Автори:</b> ${authorsNames}</p>
                <p><b>Країни:</b> ${countries}</p>

                <button class="edit-btn"
                    onclick="editBook(${book.id},
'${book.title}', '${book.year}', '${book.authors ?
```

```
book.authors.map(a => a.id).join(",") : ""}')">
                     Редагувати
                </button>

                <button class="delete-btn"
onclick="deleteBook(${book.id})">
                     Видалити
                </button>
            </div>
        `;
    });
}
```

У цій функції для кожної книги формується окрема картка. Вона містить назву книги, рік видання, список авторів, країни авторів, а також кнопки редагування та видалення. Для уникнення дублювання назв країн використовується об’єкт Set.

Функція loadAuthors() завантажує список авторів:

```
async function loadAuthors() {
const response = await fetch("/authors");
const authors = await response.json();

const authorsList =
document.getElementById("authorsList");
authorsList.innerHTML = "";

authors.forEach(author => {
    const countryName = author.country ?
author.country.name : "Не вказана";

    authorsList.innerHTML += `
        <tr>
```

```
                <td
style="padding:8px;">${author.id}</td>
                <td
style="padding:8px;">${author.name}</td>
                <td
style="padding:8px;">${countryName}</td>
                <td style="padding:8px;">
                    <button class="edit-btn"

onclick="editAuthorByData(${author.id},
'${author.name}', '${countryName}')">
```



```
                    </button>
                </td>
            </tr>
        `;
    });
}
```

Функція виконує GET-запит до /authors та формує таблицю авторів із кнопкою редагування.

Функція editAuthorByData() використовується для редагування автора:

```
    async function editAuthorByData(id, oldName,
oldCountry) {
    const newName = prompt("Нове ім'я автора:", oldName);
    if (newName === null) return;

    const newCountry = prompt("Нова країна автора:",
oldCountry);
    if (newCountry === null) return;

    const response = await fetch(`/authors/${id}`, {
```

```
        method: "PUT",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            name: newName,
            country: {
                name: newCountry
            }
        })
    });

    if (response.ok) {
        alert("Автора оновлено");
        loadAuthors();
        loadBooks();
    } else {
        alert("Помилка редагування автора");
    }
}
```

Після редагування автора оновлюється як список авторів, так і список книг, оскільки книги можуть відображати дані зміненого автора.

Функції showAuthorsBlock() та showCountriesBlock() відповідають за перемикання між списком авторів і списком країн:

```
 function showAuthorsBlock() {
    document.getElementById("authorsBlock").style.display =
"block";
    document.getElementById("countriesBlock").style.display
= "none";
}

function showCountriesBlock() {
    document.getElementById("authorsBlock").style.display =
```

```
"none";
    document.getElementById("countriesBlock").style.display
= "block";
    loadCountries();
}
```

Такий підхід дозволяє зробити інтерфейс компактним і не перевантажувати сторінку зайвими блоками.

Функція loadCountries() завантажує список країн:

```
    async function loadCountries() {
    const response = await fetch("/countries");
    const countries = await response.json();

    const countriesList =
document.getElementById("countriesList");
    countriesList.innerHTML = "";

    countries.forEach(country => {
        countriesList.innerHTML += `
            <tr>
                <td
style="padding:8px;">${country.id}</td>
                <td
style="padding:8px;">${country.name}</td>
                <td style="padding:8px;">
                    <button class="edit-btn"
onclick="editCountry(${country.id},
'${country.name}')">
                        
                    </button>
                </td>
            </tr>
```

```
        `;
    });
}
```

Функція виконує GET-запит до /countries і формує таблицю країн.

Функція addCountry() додає нову країну:

```
async function addCountry() {
    const name =
document.getElementById("countryName").value;

    const response = await fetch("/countries", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            name: name
        })
    });

    if (response.ok) {
        alert("Країну додано");
        document.getElementById("countryName").value = "";
        loadCountries();
    } else {
        alert("Помилка при додаванні країни");
    }
}
```

Функція надсилає POST-запит до /countries і після успішного додавання оновлює список країн.

Функція editCountry() використовується для редагування країни:

```
async function editCountry(id, oldName) {
    const newName = prompt("Нова назва країни:", oldName);
```

```
    if (newName === null) return;

    const response = await fetch(`/countries/${id}`, {
        method: "PUT",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            name: newName
        })
    });

    if (response.ok) {
        alert("Країну оновлено");
        loadCountries();
        loadAuthors();
        loadBooks();
    } else {
        alert("Помилка редагування країни");
    }
}
```

Після редагування країни оновлюються списки країн, авторів і книг, оскільки зміна країни може впливати на інформацію, що відображається в інших частинах інтерфейсу.

Після завантаження сторінки автоматично викликаються функції:

loadBooks();
loadAuthors();

Це дозволяє одразу відобразити на сторінці список книг і список авторів без додаткових дій користувача.

Отже, клієнтська частина забезпечує перегляд, пошук, додавання, редагування та видалення даних, а також взаємодію з REST API серверної частини застосунку.

### 4.10 Реалізація механізму пошуку

Одним із головних елементів системи є пошук.

Було реалізовано єдине поле пошуку, яке дозволяє знаходити інформацію:

- за назвою книги;
- за ім'ям автора.

Алгоритм роботи:

1. Користувач вводить текст.
2. JavaScript надсилає HTTP-запит.
3. Контролер отримує параметр пошуку.
4. Репозиторій виконує пошук.
5. Результати повертаються на сторінку.

Подібний підхід значно покращує зручність використання системи.

### 4.11 Реалізація операції додавання автора

Для додавання автора створено форму:

```
<input id="authorName">
<input id="authorCountry">
```

Після натискання кнопки виконується JavaScript-функція:

addAuthor()

Функція формує POST-запит до сервера.

Після успішного виконання новий автор автоматично зберігається в базі даних.

### 4.12 Реалізація операції додавання книги

Для додавання книги реалізовано окрему форму.
Користувач вводить:

- назву;
- рік;
- ідентифікатор автора.

Після натискання кнопки виконується функція:
addBook()

Система створює новий запис у таблиці books.

### 4.13 Реалізація операції видалення книги

Для кожної книги відображається кнопка:
🗑 Видалити

Після натискання виконується функція:
deleteBook(id)

На сервер надсилається запит:
DELETE /books/{id}

Після успішного видалення список автоматично оновлюється.

### 4.14 Використання Git та GitHub

Для контролю версій під час розробки програмного забезпечення використовувалася система керування версіями Git. Вона дозволяє

відстежувати зміни у вихідному коді, зберігати історію модифікацій та забезпечує можливість повернення до попередніх версій проєкту.

Основними командами Git, які використовувалися під час розробки, є:

```
git add .
git commit -m "message"
git push
```

Для зберігання вихідного коду та резервного копіювання проєкту використовувався сервіс GitHub.

Віддалений репозиторій проєкту розміщений за адресою:

https://github.com/poniatenkooleksandr/library-project

Використання GitHub забезпечує:

- резервне зберігання програмного коду;
- контроль змін у проєкті;
- можливість спільної роботи над програмним забезпеченням;
- демонстрацію результатів розробки.

### 4.15 Висновки до розділу 4

У даному розділі було розглянуто практичну реалізацію веб-застосунку «Бібліотека» з використанням фреймворку Spring Boot та технологій роботи з персистентними даними. Було виконано налаштування проєкту за допомогою системи керування залежностями Maven, а також обрано сучасний стек технологій, що забезпечує ефективну розробку та подальший розвиток програмного забезпечення.

У процесі реалізації створено сутності Author, Book та Country, для яких налаштовано зв’язки типів One-To-Many і Many-To-Many за допомогою засобів JPA та Hibernate ORM. Для доступу до даних розроблено репозиторії на основі Spring Data JPA, що дозволило автоматизувати виконання основних операцій з базою даних без написання SQL-запитів вручну.

Також було реалізовано сервісний рівень, який забезпечує виконання бізнес-логіки застосунку, та REST-контролери для обробки HTTP-запитів клієнтської частини. Розроблений веб-інтерфейс на основі HTML, CSS і JavaScript забезпечує можливість перегляду, пошуку, додавання, редагування та видалення даних про книги, авторів і країни.

Окрему увагу приділено реалізації механізму пошуку, який дозволяє знаходити книги за назвою та авторами, що підвищує зручність роботи користувача із системою. Для контролю версій та зберігання вихідного коду було використано Git і GitHub.

Отримані результати підтвердили доцільність використання Spring Framework, Spring Data JPA та Hibernate ORM для організації роботи з персистентними даними, а також продемонстрували ефективність обраної архітектури під час створення сучасних веб-застосунків.

# РОЗДІЛ 5: ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

## 5.1 Мета тестування

Після завершення розробки програмного застосунку необхідно провести тестування для перевірки коректності його роботи.

Основною метою тестування є підтвердження працездатності всіх функцій системи та перевірка правильності взаємодії між користувацьким інтерфейсом, серверною частиною та базою даних.

У рамках даної роботи перевірялась робота таких функціональних можливостей:

- перегляд списку авторів;
- перегляд списку книг;
- додавання автора;
- додавання книги;
- пошук за назвою книги;
- пошук за автором;
- видалення книги;
- взаємодія із базою даних;
- робота REST API.

Тестування виконувалося за допомогою веббраузера та програмного засобу Postman.

## 5.2 Тестування отримання списку авторів

Для перевірки роботи сервісу отримання списку авторів використовувався HTTP-запит:

GET /authors

Очікуваний результат:

Система повинна повернути список усіх авторів у форматі JSON.

Фактичний результат:

Сервер успішно повернув список авторів.

Приклад відповіді:


```
[
 {
   "id":1,
   "name":"Тарас Шевченко",
   "country":"Україна"
 }
]
```


Результат тестування підтвердив коректність роботи API.

У звіт необхідно вставити скріншот відповіді Postman.

.




Рисунок *5.1 – Отримання списку авторів через Postman*

### 5.3 Тестування отримання списку книг

Для перевірки функціональності перегляду книг використовувався запит:

GET /books

Очікуваний результат:

Повернення повного списку книг.

Фактичний результат:

Система успішно повернула список книг із зазначенням авторів.

Приклад відповіді:

```
[
 {
   "title":"Кобзар",
   "year":1840,
   "author":{
      "name":"Тарас Шевченко"
   }
 }
]
```

Отримані результати підтверджують коректну роботу механізму доступу до бази даних.




Рисунок 5.2 – *Отримання списку книг*

### 5.4 Тестування додавання автора

Для перевірки функції створення автора використовувався запит:

POST /authors

Тіло запиту:

```
{
  "name":"Ліна Костенко",
  "country":"Україна"
}
```

Очікуваний результат:

Новий запис повинен бути доданий до таблиці authors.

Фактичний результат:

Автор успішно створений.

Після виконання запиту запис з'явився у списку авторів.

*Рисунок 5.3 – Додавання автора.*

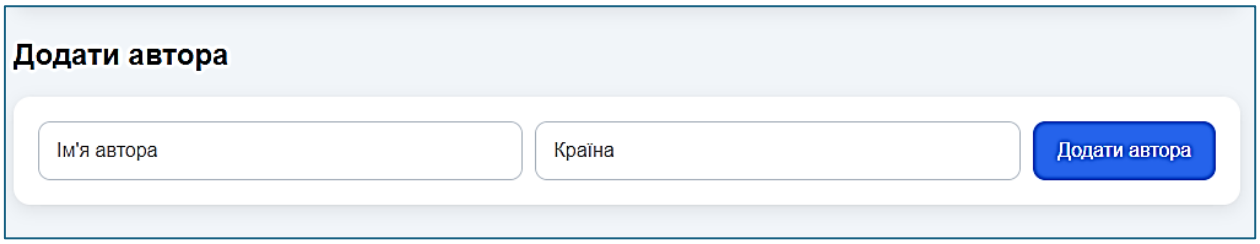



Рисунок 5.3 – *Додавання автора.*

### 5.5 Тестування додавання книги

Для перевірки функції додавання книги було використано запит:

POST /books

Приклад запиту:

```
{
  "title":"Маруся Чурай",
  "year":1979,
  "author":{
      "id":5
  }
}
```

Очікуваний результат:

Створення нового запису в таблиці books.

Фактичний результат:

Книга була успішно додана до бази даних.

Після оновлення сторінки новий запис відображався в інтерфейсі користувача.

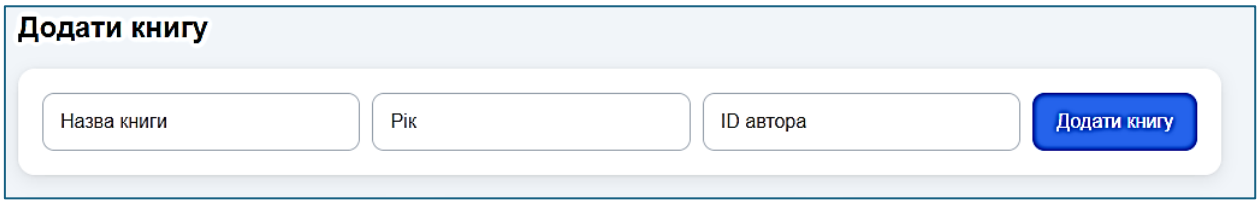



Рисунок *5.4 – Додавання книги через вебінтерфейс.*

## 5.6 Тестування пошуку за назвою книги

Для перевірки механізму пошуку було введено слово:

Кобзар

Очікуваний результат:

Система повинна знайти книгу «Кобзар».

Фактичний результат:

У результатах пошуку відображалась лише відповідна книга.

Помилок не виявлено.

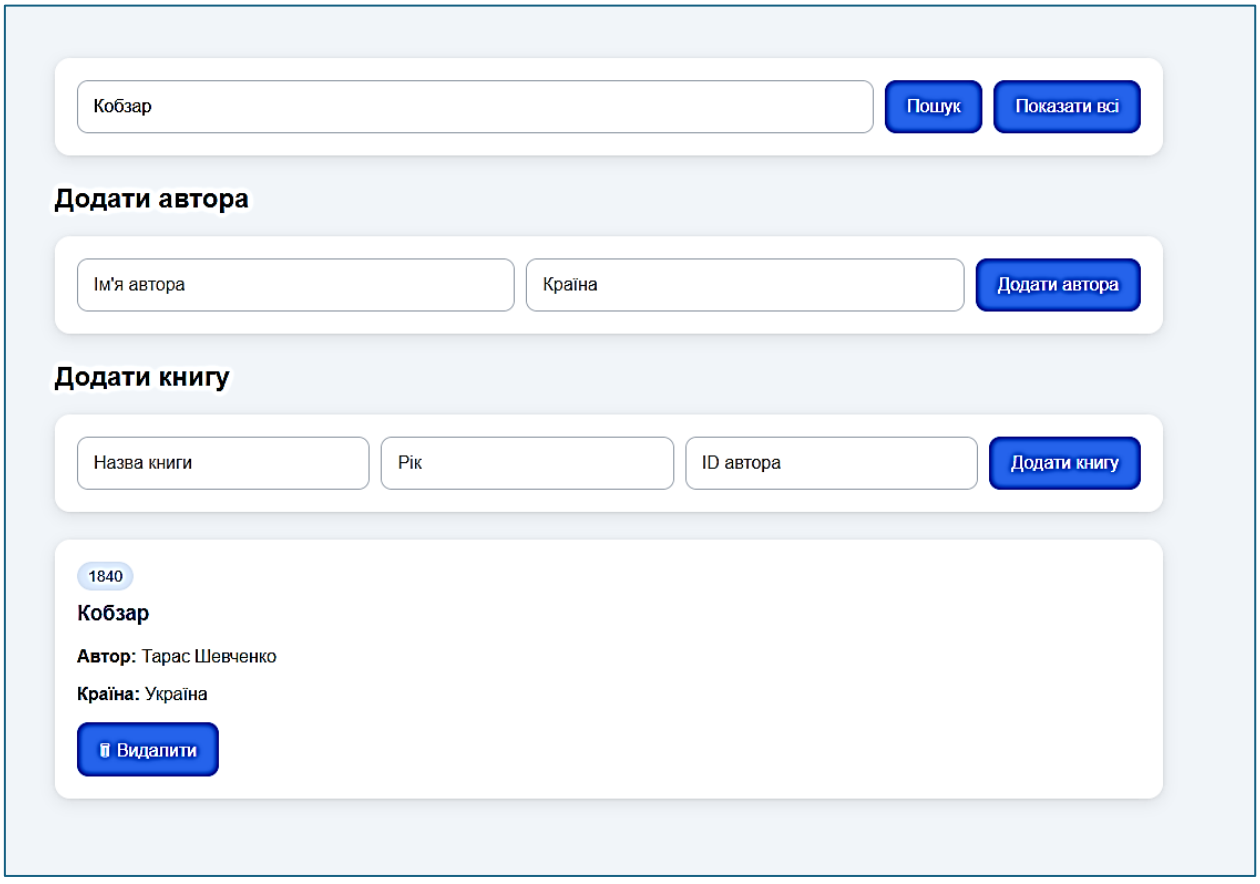



Рисунок *5.5 – Пошук за назвою книги*

### 5.7 Тестування пошуку за автором

Особливістю реалізованого застосунку є підтримка пошуку не лише за назвою книги, а й за автором.

Для перевірки було використано запит:

Шевченко

Очікуваний результат:

Система повинна знайти всі книги Тараса Шевченка.

Фактичний результат:

У списку результатів відобразились відповідні записи.

Функція працює коректно.

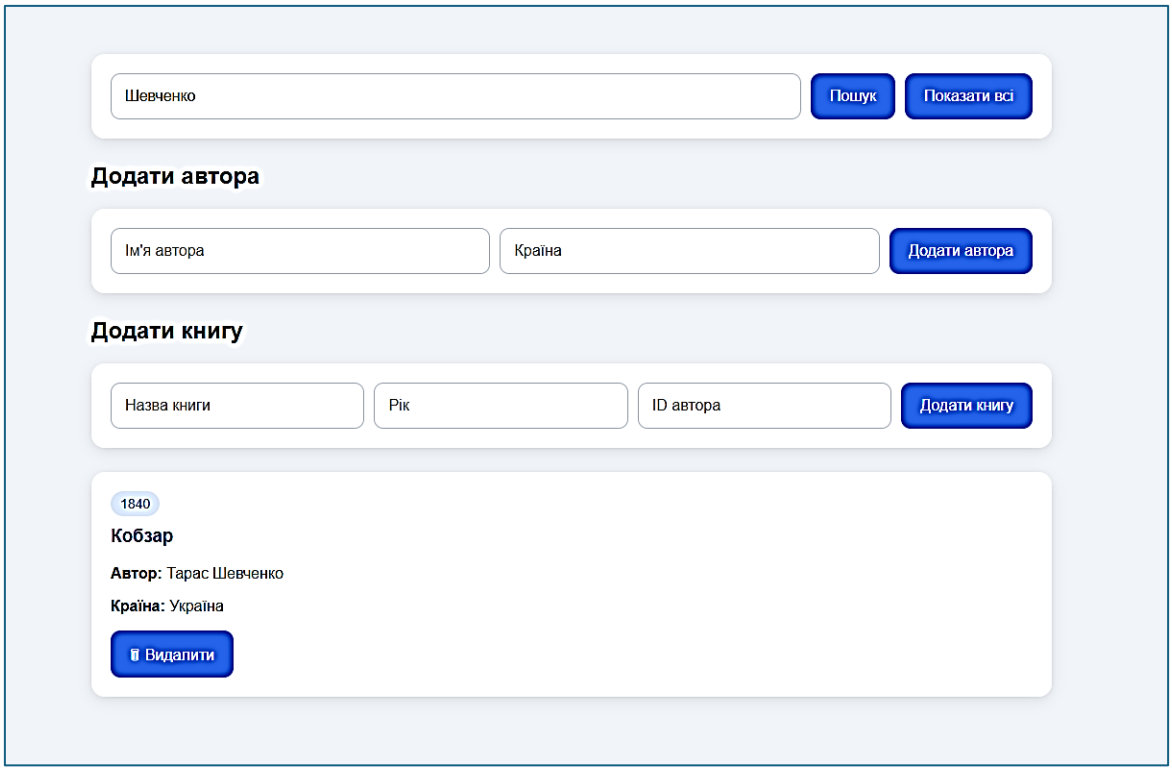



Рисунок *5.6 – Пошук за автором*

### 5.8 Тестування видалення книги

Для перевірки механізму видалення використовувалась кнопка:

🗑 Видалити

Очікуваний результат:

Після натискання книга повинна бути видалена з бази даних.

Фактичний результат:

Запис успішно видалявся.

Після оновлення сторінки книга більше не відображалась.

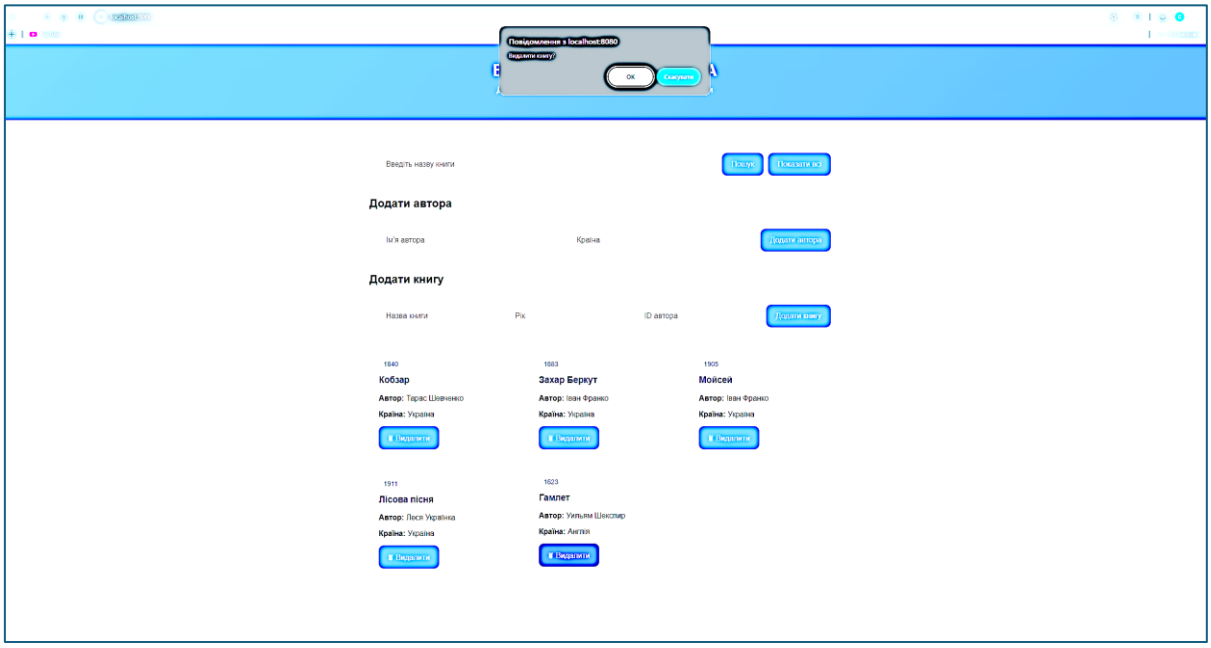



Рисунок *5.7 – Видалення книги.*

Під час тестування не було виявлено помітних затримок у роботі інтерфейсу. Усі операції додавання, редагування, пошуку та видалення даних виконувалися коректно та забезпечували комфортну взаємодію користувача із системою.

## 5.9 Аналіз переваг використання Spring Data JPA

У процесі виконання роботи було встановлено такі переваги Spring Data JPA:

- мінімальна кількість програмного коду;
- автоматичне створення SQL-запитів;
- швидка реалізація CRUD-функціоналу;
- підтримка Repository Pattern;
- інтеграція з Hibernate;

- підтримка транзакцій;
- висока масштабованість.

Використання Spring Data JPA дозволило суттєво скоротити час розробки застосунку.

### 5.10 Аналіз отриманих результатів

Розроблений веб-застосунок повністю виконав поставлені завдання.

Було успішно реалізовано:

- роботу з персистентними даними;
- використання Spring Boot;
- використання Hibernate;
- використання Spring Data JPA;
- створення REST API;
- реалізацію вебінтерфейсу;
- CRUD-операції;
- механізм пошуку.

Практична реалізація підтвердила ефективність використання Spring Framework для створення інформаційних систем.

### 5.11 Висновки до розділу 5

У ході тестування було перевірено всі основні функції розробленого застосунку. Результати тестування підтвердили коректність роботи програмного забезпечення та правильність використання засобів Spring Framework для організації роботи з персистентними даними. Система успішно виконує операції створення, перегляду, пошуку та видалення інформації, забезпечуючи стабільну взаємодію між користувацьким інтерфейсом та базою даних.

# ВИСНОВКИ

У результаті виконання бакалаврської роботи було досягнуто поставленої мети та розроблено веб-застосунок «Бібліотека» для організації роботи з персистентними даними.

Основні результати роботи:

1. Проведено аналіз засобів Spring Framework для організації роботи з персистентними даними та обґрунтовано використання Spring Data JPA і Hibernate ORM для реалізації доступу до бази даних.

2. Спроєктовано структуру веб-застосунку, модель бази даних та REST API. Реалізовано сутності Country, Author і Book, а також зв’язки One-To-Many та Many-To-Many між ними.

3. Розроблено серверну частину застосунку засобами Spring Boot. Реалізовано контролери, репозиторії та сервісний шар для виконання операцій створення, перегляду, редагування, пошуку та видалення даних.

4. Розроблено клієнтський веб-інтерфейс на основі HTML, CSS і JavaScript. Реалізовано перегляд книг, авторів і країн, пошук за назвою книги або ім’ям автора, а також механізми додавання та редагування даних.

5. Проведено тестування створеного програмного продукту, яке підтвердило коректність роботи основних функцій системи та взаємодії між клієнтською і серверною частинами застосунку.

Отримані результати можуть бути використані як основа для подальшого розвитку інформаційних систем обліку та керування бібліотечними ресурсами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Понятенко О. Дослідження засобів фрейморку Spring для організації роботи з персистентними даними. Збірник наукових праць студентів, здобувачів PhD, докторантів і молодих вчених, університетська конференція «МОЛОДА НАУКА-2026» Том V «Секції Інженерного навчально-наукового інституту ім. Ю.М. Потебні Запорізького національного університету» : м. Запоріжжя, 20–24 квітня 2026 р. Запоріжжя, 2026. С .

2. Long J., Detmer M. *Spring 6 Recipes: A Problem-Solution Approach*. Apress, 2023.

3. Sacco A. *Beginning Spring Data*. Apress, 2023.

4. Bauer C., King G. *Java Persistence with Hibernate*. 2nd ed. Manning Publications, 2015.

5. Oracle Corporation. *JDBC Developer's Guide*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/26/jjdbc/jdbc-developers-guide.pdf (дата звернення: 10.04.2026).

6. Spring Framework. URL: https://spring.io/projects/spring-framework (дата звернення: 10.04.2026).

7. Spring Boot. URL: https://spring.io/projects/spring-boot (дата звернення: 10.04.2026).

8. Spring Data JPA. URL: https://spring.io/projects/spring-data-jpa (дата звернення: 10.04.2026).

9. Hibernate ORM. URL: https://hibernate.org/orm/ (дата звернення: 10.04.2026).

**Декларація**

**академічної доброчесності**

**здобувача ступеня вищої освіти ЗНУ**

Я, Понятенко Олександр Костянтинович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми «Програмне забезпечення систем», адреса електронної пошти se121-13@std.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «Дослідження засобів фрейморку Spring для організації роботи з персистентними даними» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

-згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 07.06.2026 _____ Понятенко Олександр Костянтинович
(студент)

Дата 07.06.2026 _____ Коломоєць Геннадій Павлович
(керівник)