МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри
_____Світлана ВАЩЕНКО
____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»
на тему: «Web-система підтримки надання бібліотечних послуг»
_____
_____

Здобувача групи ІТ.мз-22с Сітало Максима Євгенійовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Максим СІТАЛО
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник к.т.н., доц. Юлія ПАРФЕНЕНКО _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра студентові

Сітало Максим Євгенійович
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи «Web-система підтримки надання бібліотечних послуг»

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «15» січня 2024 р.

3 Вхідні дані до кваліфікаційної роботи дані про каталог книг бібліотеки, дані про користувачів, дані про персонал бібліотеки

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, моделювання об'єкту дослідження, програмна реалізація web-системи підтримки надання бібліотечних послуг

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) тема, актуальність, постановка задачі, огляд програмних продуктів-аналогів, таблиця порівняння аналогів, функціональні вимоги до Web-орієнтованої системи, інструменти реалізації, структурно-функціональне моделювання, моделювання бази даних, моделювання варіантів використання, практична реалізація, демонстрація роботи, висновок

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області та постановка задачі дослідження		
2	Планування робіт		
3	Вибір методів дослідження		
4	Моделювання/проектування		
5	Практична реалізація результатів дослідження		
6	Написання розділів пояснювальної записки та надання керівнику на перевірку		
7	Підготовка заключної версії кваліфікаційної роботи, перевірка на плагіат		
8	Подача до ЕК комплекту документів до захисту		

Магістрант _____ Максим Сітало

Керівник роботи _____ к.т.н., доц. Юлія ПАРФЕНЕНКО

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Web-система підтримки надання бібліотечних послуг».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 35 найменувань, додатків. Загальний обсяг роботи − 84 сторінок, у тому числі 46 сторінок основного тексту, 4 сторінки списку використаних джерел, 34 сторінок додатків.

Актуальність роботи полягає в розробці та впровадженны Web-системи для підтримки надання бібліотечних послуг.

Мета роботи: створення Web системи на основі технології Java Servlets, що автоматизує процеси замовлення книг з каталогу користувачами послуг бібліотеки.

Обсяг анотації: 1 сторінка.

Ключові слова: Web-система, Java, Servlet, JSP, HTML, CSS, JavaScript, MySQL, MVC, Command.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології визначають прогрес, ефективність, доступність та зручність для людей різних сфер життя. Розвиток технологій це також важлива частина суспільного, економічного та культурного життя. Неможливо знайти галузь, яка під впливом технологічного розвитку не зазнала значних змін.

Об’єктом дослідження є інформаційне забезпечення підтримки надання бібліотечних послуг.

Предметом дослідження є веб-система для підтримки надання бібліотечних послуг.

Метою роботи є покращення обслуговування читачів і оптимізація рутинних завдань бібліотекарів шляхом автоматизації основних функцій за допомогою web-системи.

Для досягнення поставленої мети необхідно виконати наступні задачі:

- Виконати аналіз предметної області, а саме огляд літературних джерел, аналіз продуктів-аналогів;
- Провести аналіз технологій розробки веб-систем та визначити вимоги до до розроблювальної веб-системи;
- Виконати проектування бази даних;
- Змоделювати сценарії використання web-системи;
- Реалізувати веб-систему з допомогою визначених програмних засобів та провести її тестування.

Розроблена система підвищить задоволеність клієнтів і персоналу, а також покращить роботу бібліотеки. Бібліотека може використовувати її для підтримки, перегляду статистики та аналізу, полегшення взаємодії з читачами та точного обліку книг. Це дозволить закладу вдосконалювати свою роботу та відповідати вимогам сучасного інформаційного суспільства.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Застосування інформаційних технологій у бібліотечній сфері

У сучасному інформаційному суспільстві проблема оптимізації бібліотечних процесів стає все більш важливою [1]. Завдяки зростанню обсягу інформації та стрімкому розвитку технологій є необхідним впровадження ефективних інструментів для оптимізації ресурсів бібліотек і надійного обслуговування читачів [2].

Бібліотеки стають місцями, де збираються різні види інформації, від традиційних друкованих видань до електронних ресурсів, у сучасному світі [3]. Ця різноманітність потреб читачів і значні зміни у форматах зберігання інформації вимагають нового підходу до управління роботою бібліотек.

Як наслідок, вирішення проблеми оптимізації бібліотечних процесів має вирішальне значення для забезпечення того, щоб бібліотечна галузь відповідала вимогам сучасного суспільства та функціонувала з високою якістю в умовах інформаційного розвитку.

Огляд літератури та існуючих технологій є важливим кроком для розуміння сучасного стану проблеми та визначення напрямків подальших досліджень, що допоможуть у створенні веб-застосунку, який допоможе зробити функціонування бібліотеки більш зручним та сучасним.

У книзі [4] автор вказує, що використання інформаційних технологій в наданні бібліотечних послуг може значно полегшити ведення обліку книг, надання читачам послуг і статистичний аналіз роботи бібліотеки.

У роботі [5] автор зазначає, наскільки важливими є підвищення швидкості та точності обробки даних. Персонал бібліотеки потребує інструментів для ефективного управління ресурсами та розкладом роботи, оскільки читачі очікують високоякісного обслуговування.

У статті [6] автор аналізує вплив розвитку інформаційних технологій та, зокрема, мережі Інтернет на обслуговування читачів бібліотеки. Підбиваючи

підсумки роботи, автор зазначає, наскільки важливою є реорганізація бібліотечних функцій та зміна традиційних підходів до надання послуг.

У [7] автор детально розглядає основні ідеї та елементи сучасних систем автоматизації бібліотек. Його висновки такі: поліпшення обслуговування та ефективне управління ресурсами бібліотек залежить від ефективного використання інтегрованих технологій.

У статті [8] розглядаються основні проблеми та перспективи впровадження веб-технологій у бібліотечній сфері. Автор наголошує на тому, наскільки важливо пристосувати бібліотеки до цифрового середовища, і наголошує на ролі веб-застосунків у покращенні доступності та обслуговування клієнтів бібліотеки. Важливим елементом дослідження також є вплив веб-системи на роботу працівників бібліотек і взаємодію із читачами.

1.2 Огляд існуючих web-систем підтримки роботи бібліотекара

При підготовці до розробки web-системи підтримки надання бібліотечних послуг слід звернути особливу увагу на проблеми, які він має вирішувати. Одним із найважливіших питань є ефективне використання ресурсів бібліотеки, включаючи облік і взаємодію з книжковим фондом. Коли мова йде про якість обслуговування користувачів, точність і оперативність цих процесів є ключовими для функціонування закладу, тому при розробці системи слід максимально зосередити зусилля на покращенні взаємодії з читачами, щоб надати їм можливість користуватись послугами бібліотеки максимально зручно.

Зважаючи на це, розробка web-системи для оптимізації надання бібліотечних послуг стає необхідністю для підтримки функціонування закладу у сучасному світі. Для створення продукту, який зможе задовільнити потреби працівників та користувачів та вирішити всі згадані проблеми, потрібно провести подальше дослідження перед реалізацією програмного рішення. Одним із важливих етапів дослідження є огляд існуючих аналогів.

Бібліотека Сумського державного університету містить на своєму сайті велику кількість компонентів, велика частина яких носить суто інформативний характер, не виконуючи безпосередньо функції автоматизації певних процесів. Перелік сервісів та послуг, доступних на сайті, наведено на рисунку 1.1.

Рисунок 1.1 – Сервіси та послуги, доступні на сайті бібліотеки СумДУ

Джерело: побудовано автором (знімок з екрану)

Для більшості відвідувачів сторінки найбільш важливим та корисним є електронний каталог. Він дозволяє виконувати пошук по автору чи назві документу. Є також можливість фільтрації та сортування результатів по року видання, жанру, виду документу, тощо (рис. 1.2).

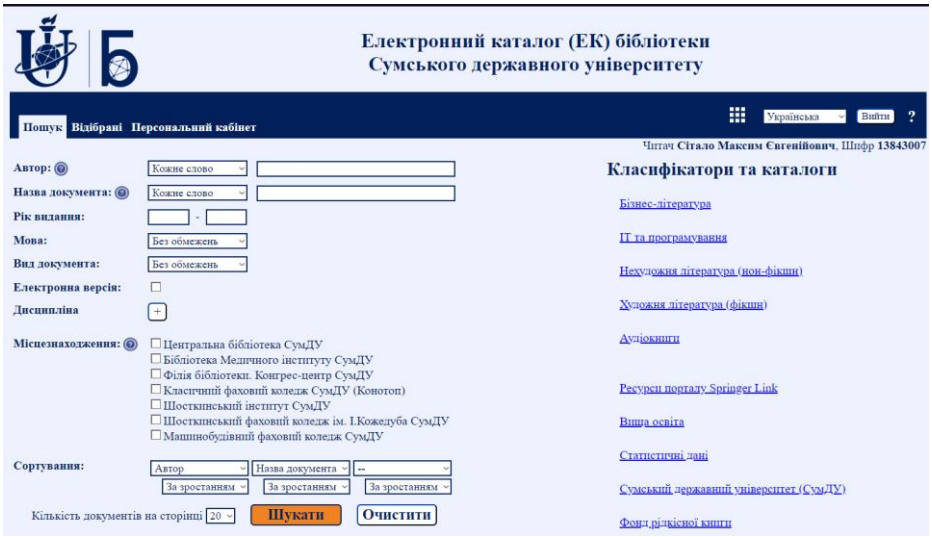


Рисунок 1.2 – Електронний каталог бібліотеки СумДУ

Джерело: побудовано автором (знімок з екрану)

Серед переваг електронного каталогу бібліотеки СумДУ можна зазначити можливість додавання літератури до списку «Відібрані», що дозволяє в подальшому стежити за їх наявністю, чи швидко знаходити певний екземпляр. Також є можливість зміни мови. Головний недолік – не дуже зручний та цікавий інтерфейс.

Сайт Сумської обласної наукової бібліотеки має значно менший функціонал у порівнянні з попереднім розглянутим продуктом (рис. 1.3-1.4).

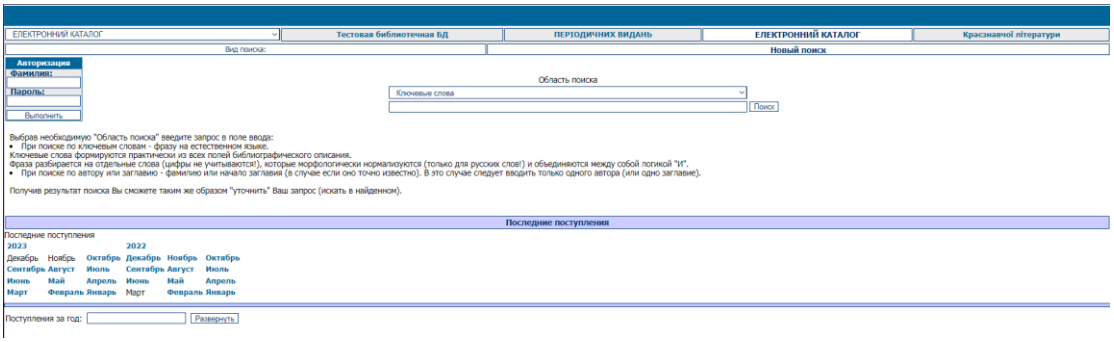


Рисунок 1.3 – Головна сторінка каталогу Сумської ОУНБ

Джерело: побудовано автором (знімок з екрану)

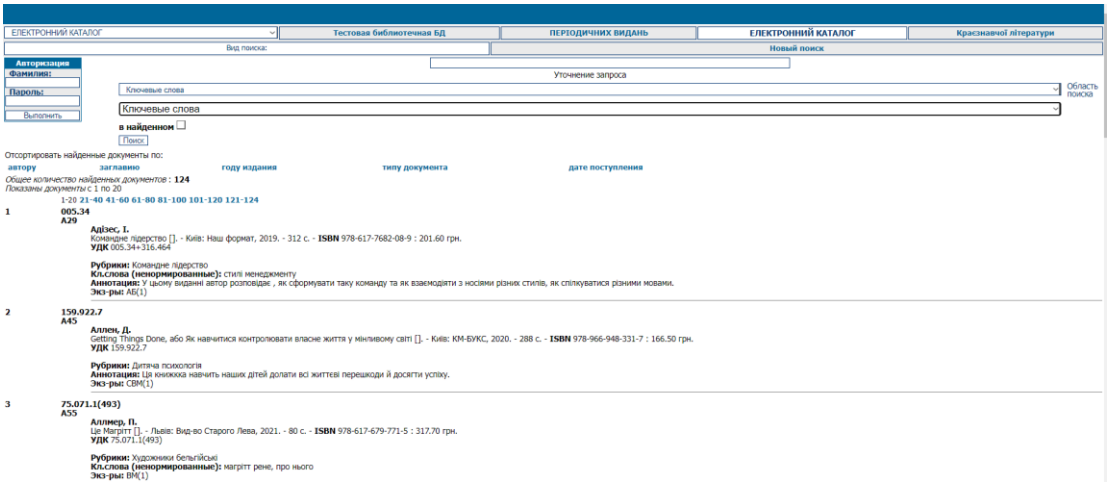


Рисунок 1.4 – Результат пошуку в каталозі Сумської ОУНБ

Джерело: побудовано автором (знімок з екрану)

Пошук літератури тут виконується тільки по назві, року видання, автору та ключовим словам. Можливість замовлення книги онлайн відсутня. Кількість екземплярів вказується тільки загальна, тобто перевірити чи доступна ця книга до замовлення можна лише відвідавши бібліотеку особисто. Можливість зміни мови відсутня, інтерфейс веб-системи незручний та незрозумілий користувачу

Після детального аналізу аналогів web-систем для підтримки надання бібліотечних послуг було визначено їх переваги та недоліки. Його результати представлені в таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця характеристик аналогів web-систем

Характеристика/ Система	**Електронний каталог бібліотеки СумДУ**	**Електронний каталог Сумської ОУНБ**	**Розроблювана веб-система**
Перегляд каталогу літератури	-	-	+
Можливість замовлення літератури	+	-	+
Можливість зміни мови	+	-	+
Особистий кабінет	+	+	+
Можливість сортування/фільтрації каталогу	+	+	+
Відстеження кількості екземплярів книг, доступних до замовлення	+	-	+

Джерело: побудовано автором

На основі даних, представлених в таблиці 1.1, можна сформувати вимоги до власної розробки, враховуючи переваги і виправляючи недоліки проаналізованих систем. Розроблювана web-система повинна реалізувати основну функціональність: перегляд каталогу літератури, інструменти для зручного пошуку необхідних видань, можливість замовлення книги.

1.3 Аналіз існуючих технологій розробки веб-систем

Для розробки web-системи необхідно обрати мову програмування, яка буде використана для розробки Backend-у системи. Головними кандидатами є Java, C# та Python.

Найпопулярніша на даний момент мова програмування Java відома перш за все своїм принципом кросплатформенності. Завдяки масштабованості, підтримкою багатопоточності, широкому вибору бібліотек та фреймворків, вбудованим засобам безпеки та активній спільноті розробників мова програмування набула популярності для розробки великих корпоративних проектів. JVM забезпечує високу швидкість виконання програм, а значна кількість вбудованих інструментів дає можливість ще більше оптимізувати роботу додатків.

Серед переваг C# можна зазначити більш зручний синтаксис, та інтеграцію з продуктами компанії Microsoft. Із недоліків варто виділити погану або взагалі відсутню інтеграцію з операційними системами, відмінними від Windows.

Python здобув велику популярність за рахунок своєї зручності та простоти використання. Але з точки зору швидкодії Python можу бути менш ефективним, ніж вказані раніше мови програмування. Великий обсяг програмного коду, написаного на цій мові буде важче підтримувати та розвивати.

Після аналізу існуючих технологій розробки web-систем було прийнято рішення про використання у розробці мови програмування Java з використанням сервлетів як основи системи, що будуть забезпечувати клієнт-серверну архітектуру. Такий вибір дозволить забезпечити продуктивність та витривалість системи, що дуже важливо для великих та комплексних систем [9]. Дотримання принципів об’єктно орієнтованого програмування допоможе у побудові чіткого та організованого коду, що полегшить розширення та підтримку коду в подальших розробках [10]. Механізми безпеки Java запобігають різноманітним типам атак, таким як SQL-ін’єкції та внедрення коду [11]. Java Servlets найкраще підходять для складних проектів, де висока продуктивність, масштабованість і об’єктно-орієнтоване програмування є важливими [12], порівняно з іншими технологіями, такими як PHP чи фреймворки типу WordPress. Використання Java Servlets дозволяє розробникам вирішувати складніші функціональні запитання та гарантувати високу безпеку та стабільність [13], хоча PHP та готові фреймворки можуть бути корисними для швидкого створення простих сайтів. WordPress, зокрема, чудово підходить для швидкого створення блогів і сайтів-візитівок, але він може бути менш гнучким для складніших завдань.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою проекту є створення web-системи, яка може запропонувати інструменти для спрощення та оптимізації бібліотечних процесів. Для цього спершу необхідно визначити, які саме функції буде виконувати система – відображення каталогу, оформлення замовлень та інші, продумати шляхи взаємодії користувачів додатку з програмними модулями.

Для розроблення web-системи можна виділити наступні задачі:

- Проектування системи, розробка діаграм для організації його структури та взаємозв’язків між компонентами та користувачами;
- Розробка інтуїтивно зрозумілого та зручного інтерфейсу;
- Реалізація бізнес-логіки системи;
- Інтеграція взаємодії з базою даних у реальному часі;
- Наповнення системи необхідним контентом;
- Проведення комплексного тестування;
- Впровадження у використання та підтримка;

Проектування системи включає в себе розробку діаграм, зокрема IDEF0, UML та ERD. Ці діаграми дозволять розробникам візуалізувати та проаналізувати архітектурні та функціональні аспекти системи. Вони сприяють розумінню вимог проекту, виявленню потенційних проблем на етапі моделювання.

На етапі розробки всі отримані раніше знання втілюються у програмний код, що при виконанні на гідному рівні впливає на ефективність реалізації системи. На цьому етапі необхідно реалізувати основний функціонал web-системи. Незареєстрований користувач повинен мати доступ до перегляду каталогу, авторизації і реєстрації. Авторизований в системі користувач, читач має можливість перегляду каталогу, замовлення книги та повернення вже виданого замовлення. Для модулю каталогу доступний пошук за автором та назвою літератури, а також сортування за назвою, автором, видавництвом та датою видання. Система відображає

для кожної книги поточну кількість примірників, готових до замовлення, та їх загальну кількість на балансі бібліотеки. Якщо користувач не повертає книгу у встановлений термін, то по замовлення нараховується пеня, що унеможливлює створення подальших замовлень. В особистому кабінеті читача відображається реєстраційна інформація, а також список його замовлень. Особистий кабінет бібліотекара містить інформацію про всіх користувачів та їх замовлення. Адміністратор системи має доступ до додавання, видалення та редагування книг, створення облікових записів бібліотекарів, блокування чи розблокування користувачів.

Неможливо недооцінити і процес тестування системи, який дозволяє виявити та виправити можливі проблеми перед введенням в експлуатацію. Це, в свою чергу, гарантує надійність та стабільність продукту, а також покращення користувацького досвіду, адже мало кому подобається користуватись продуктом, який працює не так як хотілось би.

Правильне впровадження гарантує ефективне використання системи, а ефективна підтримка допомагає вирішувати поточні проблеми та забезпечує надійність продукту в довгостроковій перспективі.

Розроблювана web-система дозволить зробити процес надання бібліотечних послуг більш зручним як для читачів, так і для працівників бібліотеки.

2.2 Функціональні вимоги

На основі потреб користувачів web-системи підтримки надання бібліотечних послуг було визначено наступні функціональні вимоги до розроблюваної системи:

– Реєстрація та авторизація користувачів;

– Можливість перегляду каталогу книг з можливістю сортування за різними критеріями, такими як автор, назва, жанр та інші;

– Модуль управління каталогом для додавання та видалення книг, оновлення інформації про книгу;

– Система замовлення та видачі книг з урахування поточної кількості наявних книг, включаючи можливість резервації та встановлення термінів повернення;

– Система нарахування штрафів у випадку неповернення книги до бібліотеки в термін, передбачений при видачі книги читачу;

Система має наступні ролі користувачів:

– Гість не має облікового запису або не виконав до нього вхід. Такий користувач матиме доступ тільки до перегляду каталогу та реєстрації і авторизації;

– Читач зможе оформлювати замовлення на книги з каталогу, отримає доступ до особистого кабінету;

– Бібліотекар не можу оформлювати замовлення на книги, але має доступ до списку користувачів та їх замовлень, мають можливість видавати замовлені книги користувачам;

– Адміністратор має можливість редагування каталогу книг.

3 МОДЕЛЮВАННЯ ОБ'ЄКТУ ДОСЛІДЖЕННЯ

3.1 Структурно-функціональне моделювання

Моделювання складається з етапів, які пов'язані між собою. Процес моделювання починається з абстрактної концептуальної схеми, після неї створюються логічна та фізична моделі.

Нотація IDEF0 використовується для наглядної демонстрації бізнес-логіки проекту [14]. При створенні діаграми в центрі створюється прямокутник, який представляє собою проект. В прямокутник з усіх сторін направлено стрілки, що представляють собою типи вхідних та вихідних даних.

Діаграма будується з точки зору користувачів – читача бібліотеки та бібліотекара. Для контекстної діаграми оформлення замовлення користувачем були визначено такі дані:

- Вхідні дані: логін і пароль користувача, потреба користувача у оформленні замовлення на книгу.
- Вихідні дані: оформлене замовлення на видачу книги.
- Управління: функціонал додатку, каталог доступної літератури.
- Механізми: веб-додаток, апаратне та технічне забезпечення, база даних MySQL.

Функціональне моделювання процесу замовлення книги користувачем з використанням веб-системи підтримки надання бібліотечних послуг в IDEF0 представлене на рисунку 3.1.

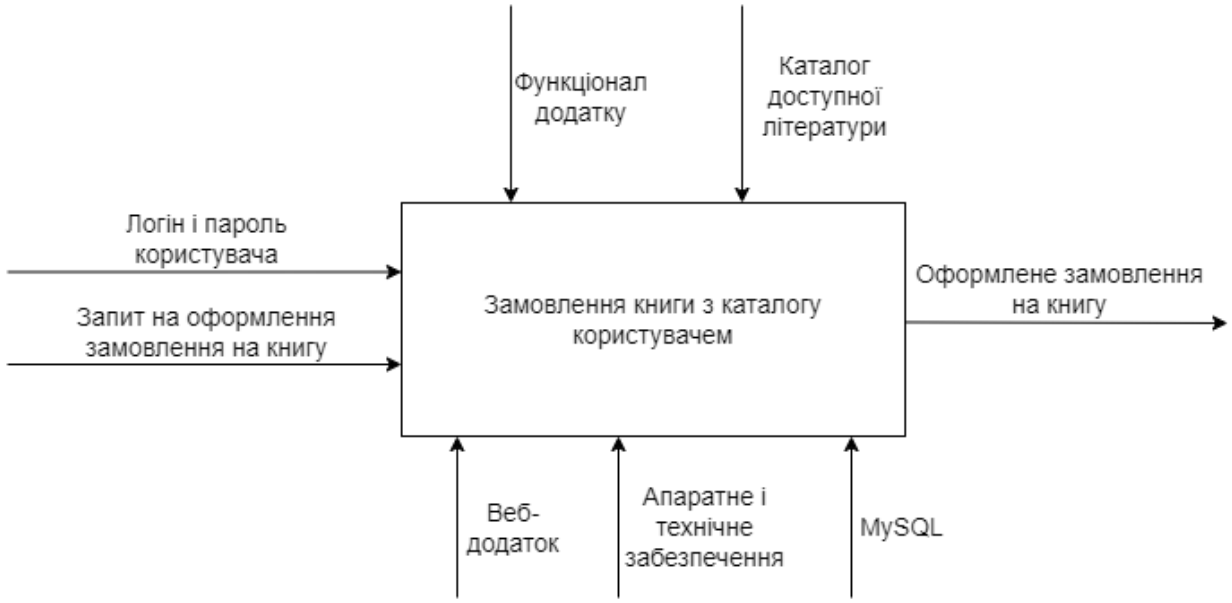


Рисунок 3.1 – Функціональна діаграма замовлення книги читачем

Джерело: побудовано автором

Незареєстрований читач має змогу переглядати каталог книг, використовуючи функції пошуку та сортування. Для доступу до оформлення замовлення користувачу потрібно зареєструватися або виконати вхід у свій обліковий запис.

Для деталізації внутрішніх процесів діаграми IDEF0 оформлення замовлення на книгу було виконано декомпозицію, що зображена на рисунку 3.2.

Декомпозиція першого рівня відображає деталізацію процесу використання web-системи підтримки надання бібліотечних послуг з точки зору користувача при оформленні замовлення. Структура блоків показує, як одні функції впливають на інші [15].

Після виконання декомпозиції було виділено 4 основні функції користування системою для оформлення замовлення:

- авторизація/реєстрація;
- перегляд каталогу літератури;
- перехід на сторінку замовлення;
- підтвердження замовлення.

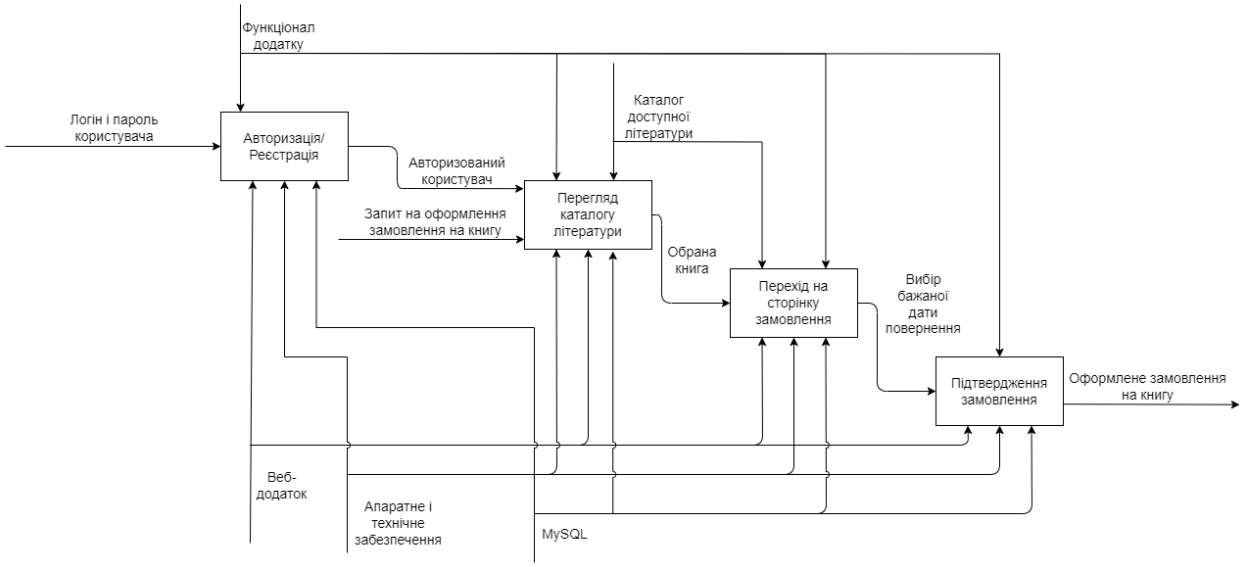


Рисунок 3.2 – Діаграма декомпозиції процесу функціонування

Джерело: побудовано автором

Під час процесу авторизації відбувається введення логіну та паролю користувача в форму авторизації. В результаті користувач отримує доступ до того функціоналу, який передбачено його обліковим записом, і може переглянути каталог доступних у наявності книг. Коли користувач знаходить потрібну книгу, він обирає її і переходить на сторінку замовлення. Там він встановлює бажану дату повернення та підтверджує замовлення. В результаті створюється нове замовлення, яке далі має підтвердити та видати користувачу бібліотекар.

Функціональне моделювання процесу видачі замовлення користувачу бібліотекарем з використанням веб-системи підтримки надання бібліотечних послуг в IDEF0 представлене на рисунку 3.3.

Після виконання декомпозиції було виділено 3 основні функції користування системою для видачі замовлення користувачу бібліотекарем:

- авторизація/реєстрація;
- перегляд списку нових замовлень;
- видача замовлення.

Результат декомпозиції процесу видачі замовлення з точки зору бібліотекара наведено на рисунку 3.4.

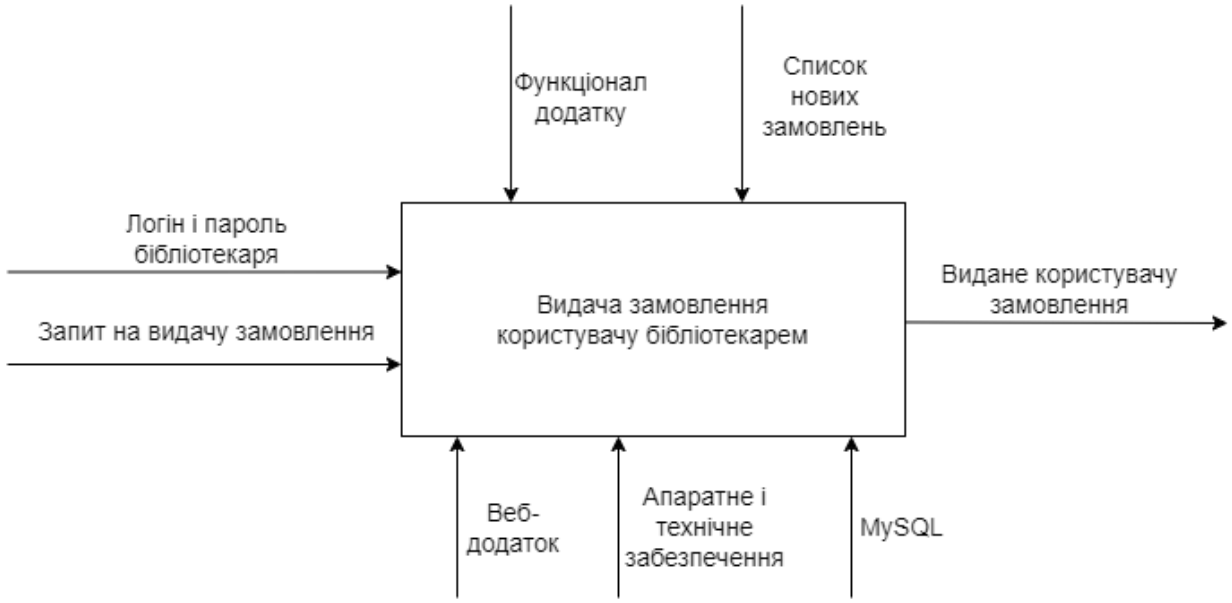


Рисунок 3.3 – Функціональна діаграма видачі замовлення бібліотекарем

Джерело: побудовано автором

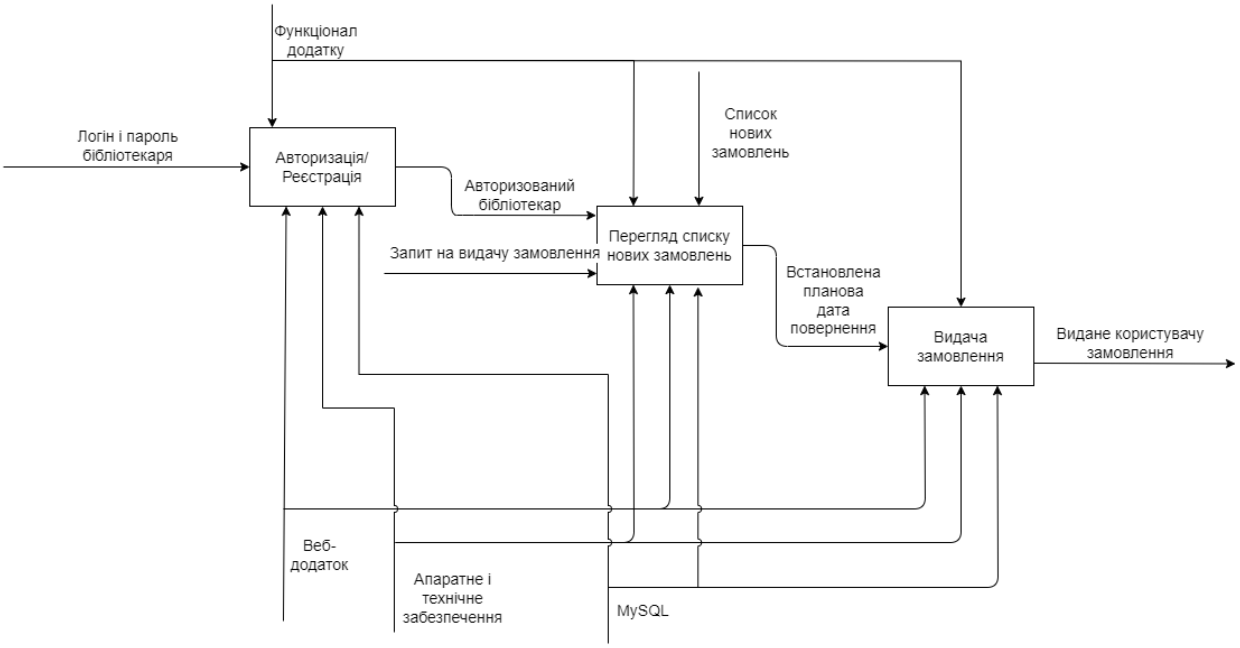


Рисунок 3.4 – Діаграма декомпозиції процесу видачі замовлення

Джерело: побудовано автором

3.2 Діаграма варіантів використання веб-системи

Після проведення структурно-функціонального моделювання web-системи необхідно розробити діаграму варіантів використання.

Діаграма варіантів використання описує функціональне призначення системи. Суть такої діаграми полягає у графічному відображенні взаємодії між користувачами і компонентами системи [16]. Під взаємодією мається на увазі те, що актор має можливість взаємодіяти з функціональним модулем. Кожен можливий варіант взаємодії визначається як варіант використання.

Акторами було визначено незареєстрованого користувача, читача, бібліотекара та адміністратора, базу даних.

Діаграма варіантів використання веб-системи підтримки надання бібліотечних послуг представлена на рисунку 3.5.

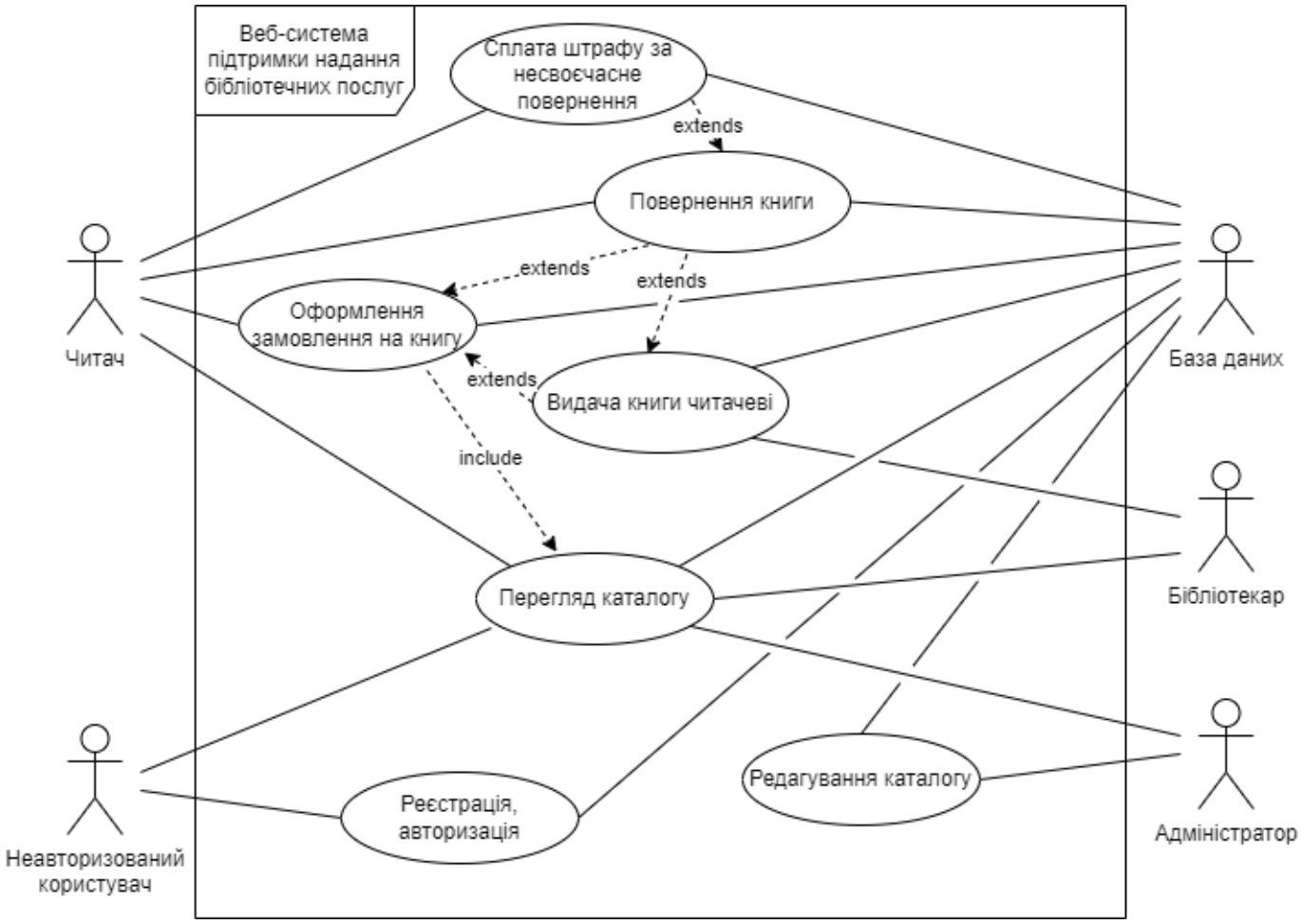


Рисунок 3.5 – Діаграма варіантів використання

Джерело: побудовано автором

Для web-системи підтримки надання бібліотечних послуг було визначено наступні варіанти використання:

- реєстрація, авторизація;
- перегляд каталогу;
- редагування каталогу;
- оформлення замовлення на книгу;
- видача замовлення читачу;
- повернення книги до бібліотеки;
- сплата штрафу за несвоєчасне повернення книги;

3.3 Моделювання бази даних веб-системи

При розробці будь-якого системи важливим етапом є моделювання бази даних для графічного відображення зв’язків між її сутностями. Побудова ERD перед розробкою проекту дозволяє візуалізувати структуру бази даних, визначити сутності та атрибути, опрацьовувати взаємозв’язки між ними та полегшувати планування розробки, допомагаючи ефективно визначити, чи потрібні зміни або розширення функціоналу.

Для бази даних веб-системи підтримки надання бібліотечних послуг було виділено наступні сутності:

- Користувач
- Книга
- Замовлення
- Штраф

Кожна з виділених сутностей має свій набір атрибутів, необхідний для функціонування системи. Користувач містить свою персональну інформацію, його роль. Сутність книги містить інформацію про саму книгу, таку як назва, автор, короткий опис, рік видання, видавництво та інші. Сутність замовлення пов’язує між собою сутності Користувача та Книги, так як обов’язково містить в собі користувача, який оформив замовлення, та книгу, яка і потрібна користувачу. Додатково в

замовленні вказані дата оформлення замовлення та планова дата повернення. Сутність штрафу містить посилання за замовлення за яким він був нарахований, суму штрафу та дату нарахування, яка буде фактичною датою повернення книги.

Взаємозв'язки між сутностями представлено в Entity Relationship діаграмі [17] на рис. 3.6.

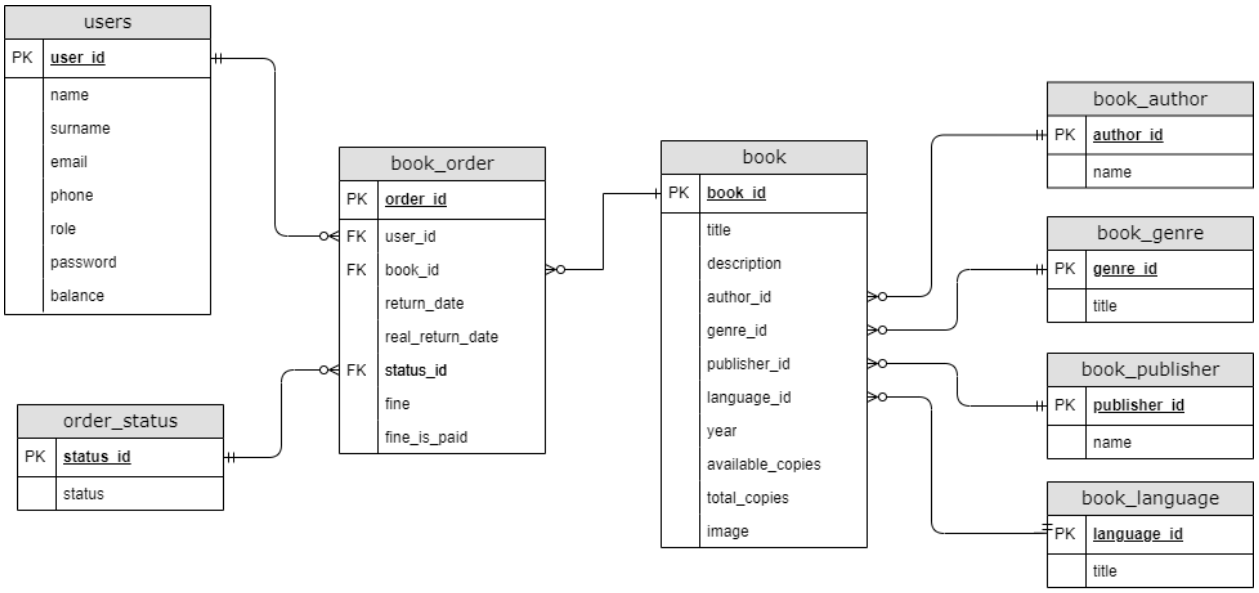


Рисунок 3.6 – Діаграма взаємозв'язків між сутностями БД

Джерело: побудовано автором

В таблицях 3.1-3.8 наведено опис атрибутів кожної сутності бази даних – їх тип даних, та призначення.

Таблиця 3.1 – Опис атрибутів сутності User

Атрибут	Тип даних	Призначення
user_id	INT	Унікальний ідентифікатор користувача
name	VARCHAR(20)	Ім'я користувача
surname	VARCHAR(20)	Прізвище користувача
email	VARCHAR(40)	Електронна пошта
phone	VARCHAR(13)	Мобільний телефон
role	INT	Роль – читач, бібліотекар чи адміністратор

password	TEXT	Зашифрований у SHA256 пароль користувача
balance	DOUBLE	Поточний баланс на рахунку користувача

Джерело: побудовано автором

Таблиця 3.2 – Опис атрибутів сутності Book

Атрибут	Тип даних	Призначення
book_id	INT	Унікальний ідентифікатор книги
title	VARCHAR	Назва книги
description	TEXT	Опис
author_id	INT	Посилання на автора книги
genre_id	INT	Посилання на жанр книги
language_id	INT	Посилання на мову видання
publisher_id	INT	Посилання на видавництво
available_copies	INT	Кількість доступних екземплярів
total_copies	INT	Загальна кількість екземплярів
year	INT	Рік видання книги
image	TEXT	Посилання на зображення обкладинки

Джерело: побудовано автором

Таблиця 3.3 – Опис атрибутів сутності Book_order

Атрибут	Тип даних	Призначення
order_id	INT	Унікальний ідентифікатор замовлення
user_id	INT	Користувач, що оформив замовлення
book_id	INT	Книга, на яку оформлено замовлення
return_date	DATETIME	Дата бажаного повернення (для нових замовлень) або дата узгодженого повернення для вже виданих замовлень
real_return_date	DATETIME	Фактична дата повернення книги
status_id	INT	Посилання на статус замовлення

fine	DOUBLE	Розмір пені
fine_is_paid	BOOLEAN	Індикатор, чи була сплачена пеня після повернення замовлення

Джерело: побудовано автором

Таблиця 3.4 – Опис атрибутів сутності Order_status

Атрибут	Тип даних	Призначення
status_id	INT	Унікальний ідентифікатор статусу замовлення
status	VARCHAR(10)	Текстове пояснення статусу замовлення

Джерело: побудовано автором

Таблиця 3.5 – Опис атрибутів сутності Book_author

Атрибут	Тип даних	Призначення
author_id	INT	Унікальний ідентифікатор автора
name	VARCHAR(40)	Ім’я чи псевдонім автора

Джерело: побудовано автором

Таблиця 3.6 – Опис атрибутів сутності Book_language

Атрибут	Тип даних	Призначення
language_id	INT	Унікальний ідентифікатор мови видання
title	VARCHAR(20)	Текстове представлення мови видання

Джерело: побудовано автором

Таблиця 3.7 – Опис атрибутів сутності Book_publisher

Атрибут	Тип даних	Призначення
publisher_id	INT	Унікальний ідентифікатор видавництва
name	VARCHAR(60)	Назва видавництва мовою оригіналу

Джерело: побудовано автором

Таблиця 3.8 – Опис атрибутів сутності Order_genre

Атрибут	Тип даних	Призначення
genre_id	INT	Унікальний ідентифікатор жанру книги
title	VARCHAR(60)	Текстове пояснення жанру книги

Джерело: побудовано автором

4 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-СИСТЕМИ ПІДТРИМКИ НАДАННЯ БІБЛІОТЕЧНИХ ПОСЛУГ

4.1 Архітектура системи

При розробці сучасних інформаційних систем велику увагу приділяють підтримці програмних рішень вже після їх впровадження у експлуатацію. Код з самого початку проектується не тільки для вирішення поточної задачі в існуючих умовах, а і для його легкого обслуговування та розширення у подальшому. Для досягнення цієї цілі розробники використовують різноманітні шаблони проектування та архітектурні підходи. Одним із найбільш відомих підходів до проектування інформаційних систем є патерн MVC, що визначає структуру програмного коду та взаємодію між собою його компонентів. Він розуміє під собою розбиття системи на три основні компоненти, кожен із яких виконує свою функцію [31]:

– Модель(Model) – Обєкти що використовуються для збереження даних та логіки їх обробки і використання;

– Вид (View) – Шар який відповідає за відображення інформації користувачу, та взаємодію з ним;

– Контролер (Controller) – Поєднує шари моделі та вигляду, виконує логіку обробки подій та визначає які дії потрібно виконати на двох попередніх шарах.

Архітектура web-системи підтримки надання бібліотечних послуг з використанням шаблону MVC зображена на рисунку 4.1.

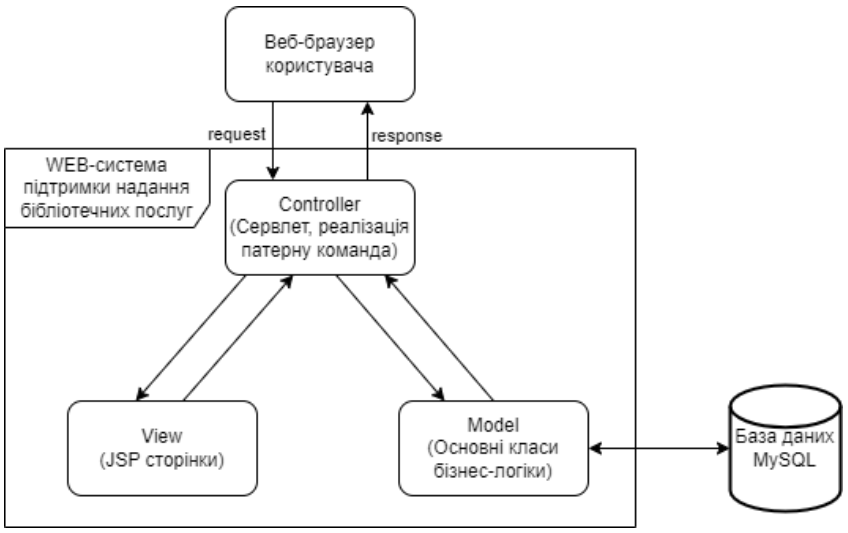


Рисунок 4.1 – Архітектура розроблюваної web-системи

Джерело: побудовано автором

Основою бізнес логіки проекту є використання поведінкового патерну Command, який дозволяє завернути запити в єдиний інтерфейс з єдиним методом – «виконати». Після цього ті ж самі відправники запитів зможуть працювати без прив'язки до конкретного класу команд [32]. Запит відправляється на шар бізнес-логіки, де його виконавець буде знайдений автоматично, без участі відправника.

Нижче наведено фрагмент коду що демонструє приклад відправки описаного вище запиту. Ця форма, як і будь-яка інша в програмі відправляє запит одному єдиному обробнику – сервлету. Приховане поле «action» фактично містить пояснення, що саме має бути виконано з цим запитом:

```
<div class="form-container">
    <h2><fmt:message bundle="${locale}" key="add_author.title"/></h2>
    <form id="authorForm" action="servlet" method="post">
        <input type="hidden" name="action" value="adminAddAuthorCommand">

        <label for="authorNameEn"><fmt:message bundle="${locale}"
key="add_author.name_en"/>:</label>
        <input type="text" id="authorNameEn" name="authorNameEn" required>

        <label for="authorNameUa"><fmt:message bundle="${locale}"
key="add_author.name_ua"/>:</label>
        <input type="text" id="authorNameUa" name="authorNameUa" required>

        <button type="submit" onclick="confirmAuthor()"><fmt:message
bundle="${locale}" key="add_author.confirm"/></button>
    </form>
</div>
```

У відповідності до типу запиту, сервлет загортає запит у єдиний інтерфейс, та передає відповідному обробнику. Нижче наведено фрагмент коду, який відповідає за обробку отриманих запитів.

```
private void handleRequest(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
    if (req.getSession().getAttribute("userRole") == null) {…}
    if (req.getSession().getAttribute("lang") == null)
req.getSession().setAttribute("lang", new Locale("ua"));
    String commandTitle = req.getParameter(JSPAttributes.COMMAND);
    if (commandTitle == null) commandTitle =
CommandsList.HOME_PAGE.getCommandName();
    Command command = CommandsList.getCommandByTitle(commandTitle);
    Router router = command.execute(req, resp);
    if (router.getRouteType().equals(Router.RouteType.FORWARD))
        req.getRequestDispatcher(router.getRouteLink()).forward(req,
resp);
    else
        resp.sendRedirect(router.getRouteLink());
}
```

В наведеному прикладі відповідно до типу запиту «adminAddAuthorCommand» буде задіяно відповідний обробник, який «дістає» із запиту необхідні у даному випадку дані – ім’я автора англійською та українською мовам, і передає отримані дані на шар бізнес логіки, де новий запис заноситься до бази даних. Нижче наведено фрагмент коду згаданого обробника.

```
public class AddAuthorAdminCommand implements Command {
    BookService bookService =
ServiceFactory.getInstance().getBookService();

    @Override
    public Router execute(HttpServletRequest req, HttpServletResponse
resp) {
        Router router = new Router();
        Object en  = req.getParameter("authorNameEn");
        Object ua = req.getParameter("authorNameUa");
        BookComponent author = new BookComponent(0, en.toString(),
ua.toString());
        bookService.addNewAuthor(author);

        req.setAttribute(JSPAttributes.COMMAND, "adminEditBook");
        req.setAttribute(JSPAttributes.ADMIN_PAGE_BOOK_LIST,
bookService.findAllBooks("0",
req.getSession().getAttribute("lang").toString()));
        router.setRouteLink(PageList.ADMIN_EDITOR_PAGE);
        router.setRouteType(Router.RouteType.FORWARD);
        return router;
    }
}
```

Варто відзначити також використання породжуючого патерну Builder. Нижче наведено фрагмент коду, що демонструє його використання. Даний патерн дозволяє покроково створювати складні об’єкти. Наприклад, при створенні нової книги немає потреби присвоювати їй ідентифікатор – це буде зроблено автоматично базою даних. А от при отриманні списку книг з бази даних навпаки, кожна книга має мати свій унікальний числовий ідентифікатор для подальшого її використання. Описаний патерн якраз дозволяє наповнити об’єкт саме так, як необхідно в умовах поточної задачі [33].

```
protected Book constructBookFromResult(ResultSet resultSet) throws
SQLException {
    return new BookBuilder().setId(resultSet.getInt(BOOK_ID))
            .setAuthor(new BookComponent(resultSet.getInt(AUTHOR_ID),
                    resultSet.getString(AUTHOR_NAME_EN),
                    resultSet.getString(AUTHOR_NAME_UA)))
            .setGenre(new BookComponent(resultSet.getInt(GENRE_ID),
                    resultSet.getString(GENRE_VALUE_EN),
                    resultSet.getString(GENRE_VALUE_UA)))
            .setLanguage(new
BookComponent(resultSet.getInt(LANGUAGE_ID),
                    resultSet.getString(LANGUAGE_VALUE_EN),
                    resultSet.getString(LANGUAGE_VALUE_UA)))
            .setPublisher(new
BookComponent(resultSet.getInt(PUBLISHER_ID),
                    resultSet.getString(PUBLISHER_VALUE_EN),
                    resultSet.getString(PUBLISHER_VALUE_UA)))
            .setTitleEn(resultSet.getString(BOOK_TITLE_EN))
            .setTitleUa(resultSet.getString(BOOK_TITLE_UA))

.setDescriptionEn(resultSet.getString(BOOK_DESCRIPTION_EN))

.setDescriptionUa(resultSet.getString(BOOK_DESCRIPTION_UA))

.setYearOfPublication(resultSet.getInt(BOOK_PUBLICATION_DATE))
            .setPages(resultSet.getInt(BOOK_PAGES))
            .setTotalQuantity(resultSet.getInt(BOOK_TOTAL_QUANTITY))

.setAvailiableQuantity(resultSet.getInt(BOOK_FREE_QUANTITY))
            .setImage(resultSet.getString(BOOK_IMAGE))
            .getResult();
}
```

Серед інших шаблонів проектування варті згадки також шаблони Factory Method та Data Access Object. Нижче наведено фрагмент коду, що демонструє приклад використання шаблону Factory Method.

```
public class DAOFactory {
    private static DAOFactory INSTANCE;
    private final BookDAO bookDAO;
```

```
        private final UserDAO userDAO;

        private DAOFactory() {
            bookDAO = new BookDAOImpl();
            userDAO = new UserDAOImpl();
        }

        public static DAOFactory getInstance() {
            if (INSTANCE == null) INSTANCE = new DAOFactory();
            return INSTANCE;
        }

        public BookDAO getBookDAO() {
            return bookDAO;
        }

        public UserDAO getUserDAO() {
            return userDAO;
        }
    }
```

Нижче наведено фрагмент коду, що демонструє приклад використання шаблону DAO.

```
package com.maxim328.dao;

import com.maxim328.model.Book;
import com.maxim328.model.BookComponent;

import java.util.Date;
import java.util.List;

public interface BookDAO {
    Book findBookById(Integer bookId);
    List<Book> findAllBooks();
    void createOrder(Integer bookId, Integer userId, Date
preferableReturnDate);
    List<BookComponent> getAllAuthors();
    List<BookComponent> getAllPublishers();
    List<BookComponent> getAllLanguages();
    List<BookComponent> getAllGenres();
    void editBook(Book updatedBook);
    void addNewBook(Book book);
    void deleteBook(Integer bookId);
    void addNewAuthor(BookComponent author);
}
```

Для керування інформаційним наповненням системи створено відповідні моделі:

- Order – сутність замовлення;
- Book – сутність книги;

– User – сутність користувача;
– BookComponent – компонент книги. Фактично, це поля які можуть дублюватися у різних книг. З метою приведення бази даних до 3-ї нормальної форми такі поля було виокремлено в окремі сутності [34].

Графічне представлення реалізації бази даних у програмному середовищі наведено на рисунку 4.2.

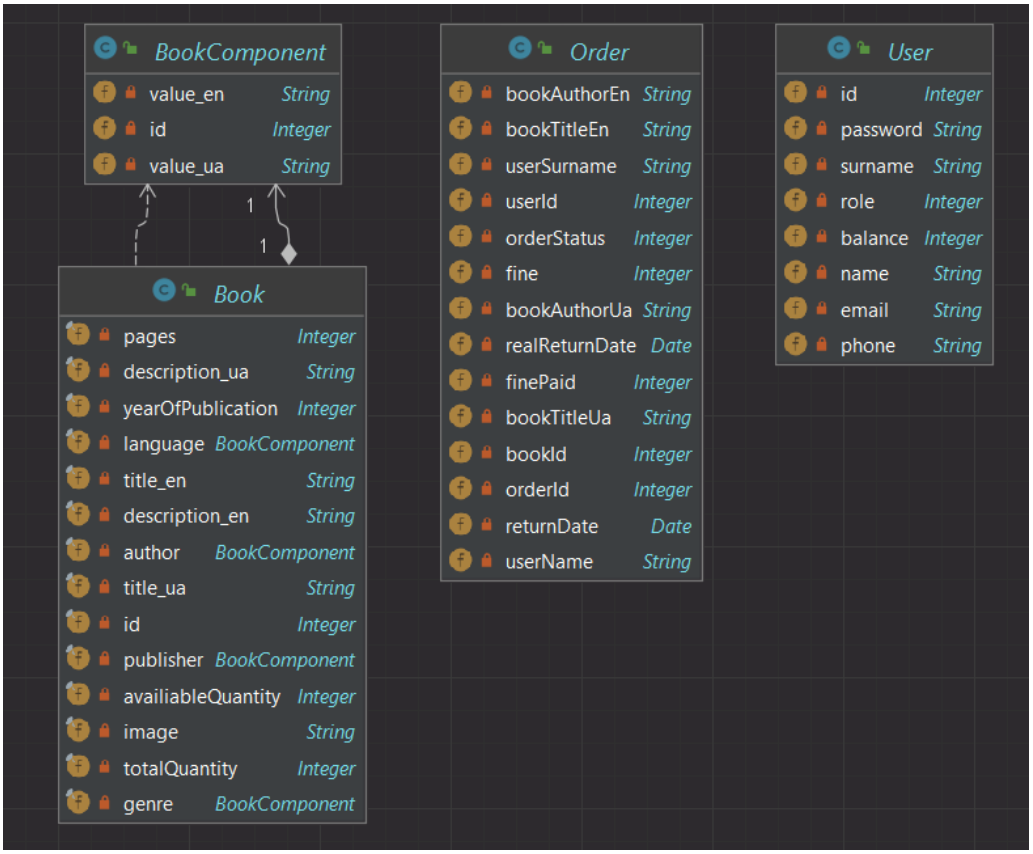


Рисунок 4.2 – Графічний вигляд моделі бази даних у програмному середовищі

Джерело: побудовано автором (знімок з екрану)

4.2 Розробка інтерфейсу

Для реалізації інтерфейсу додатку використовуються JSP або Java Server Pages сторінки. Дана технологія дозволяє об’єднати на сторінці статичний вміст, що описується мовами розмітки, як HTML та динамічний вміст, який генерується за допомогою мови програмування Java. Такий підхід дозволяє обмінюватися даними

між веб-сторінкою та серверною частиною системи в двосторонньому напрямку, а також виконувати код мовою Java всередині самої сторінки.

Всі основні сторінки сервісу побудовані з використанням технології JSP. Повний список використаних сторінок представлено на рисунку 4.3.

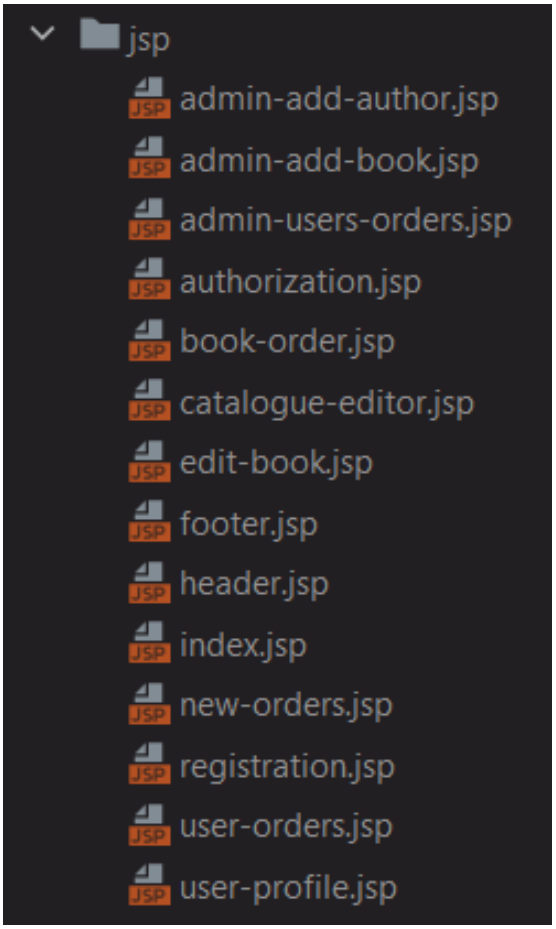


Рисунок 4.3 – Перелік використаних у проекті JSP сторінок

Джерело: побудовано автором (знімок з екрану)

Приклади відображення розроблених сторінок наведено на рисунках 4.4-4.5.

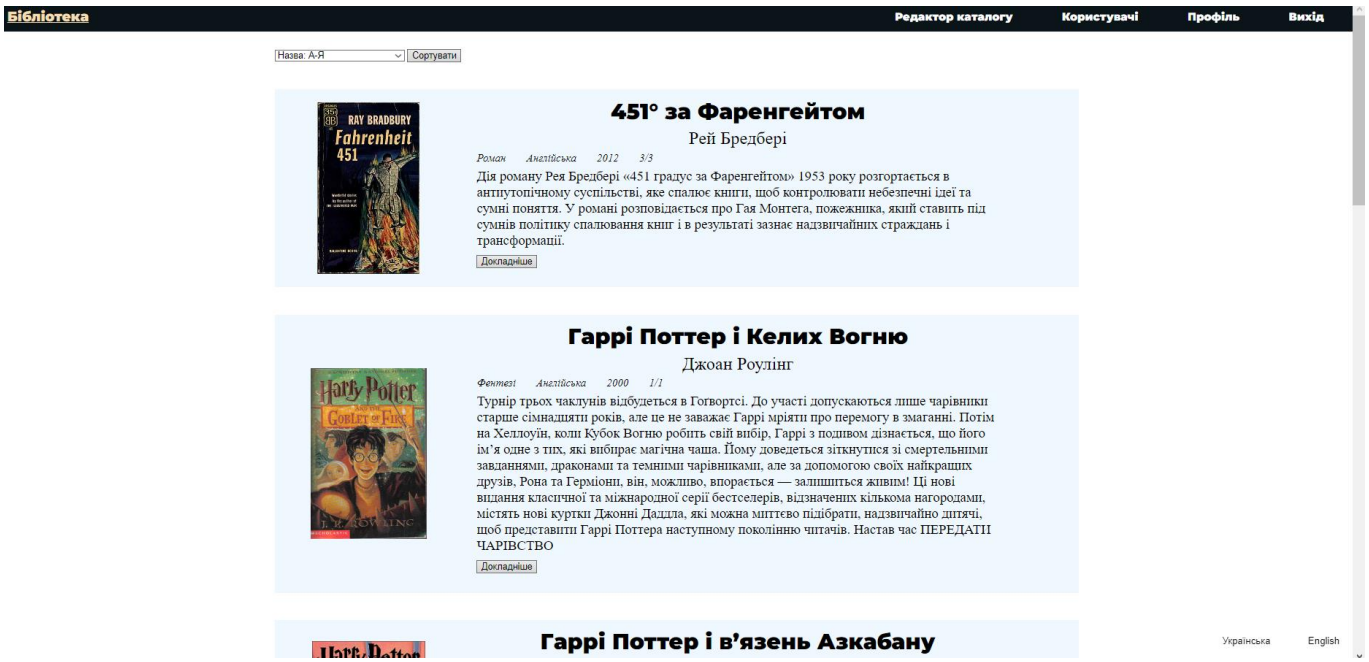


Рисунок 4.4 – Відображення сторінки каталогу книг

Джерело: побудовано автором (знімок з екрану)

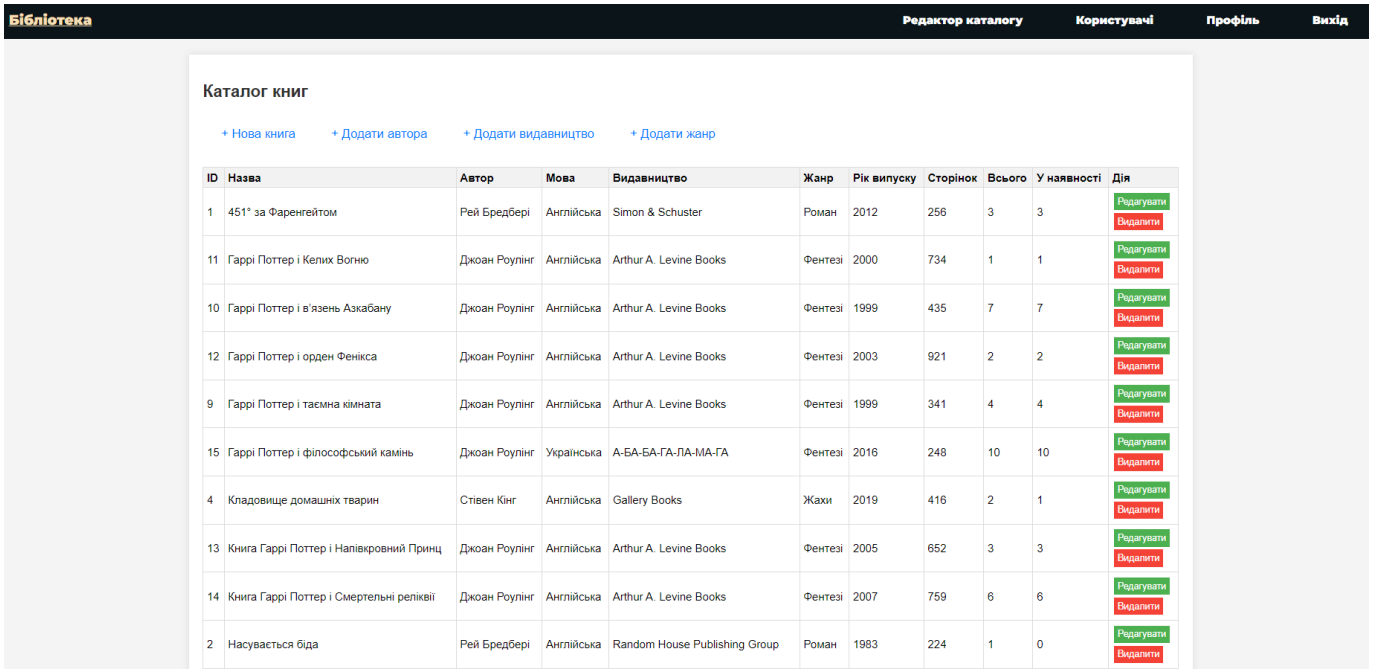


Рисунок 4.5 – Відображення редактору каталогу книг

Джерело: побудовано автором (знімок з екрану)

На обох прикладах інформаційне наповнення сторінок отримується з бази даних і генерує вміст сторінки безпосередньо у самому html коді.

4.3 Інтеграція з базою даних MySQL

При описі патернів проектування згадувалось, що для доступу до бази даних використовується поведінковий шаблон DAO та наводився приклад опису функціональності для класу, що буде безпосередньо відповідати за зв'язок з базою даних. Реалізацію представленого інтерфейсу наведено на рисунку 4.6.

```
public class BookDAOImpl extends BaseDAO implements BookDAO {
    @Override
    public Book findBookById(Integer bookId) {
        Book bookResult = null;
        ResultSet resultSet;
        try(Connection con = pool.getConnection();
                PreparedStatement ps = con.prepareStatement(
                        sql: DAOConstants.FIND_ALL_BOOKS + DAOConstants.BY_BOOK_ID)) {
            ps.setInt( parameterIndex: 1, bookId);
            resultSet = ps.executeQuery();
            resultSet.next();
            bookResult = constructBookFromResult(resultSet);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return bookResult;
    }

    @Override
    public List<Book> findAllBooks() {
        List<Book> bookList = null;
        ResultSet resultSet = null;
        try (Connection con = pool.getConnection()) {
            Statement statement = con.createStatement();
            resultSet = statement.executeQuery(DAOConstants.FIND_ALL_BOOKS);
            bookList = new ArrayList<>();
            while (resultSet.next()) {
                bookList.add(constructBookFromResult(resultSet));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeResultSet(resultSet);
        }
        return bookList;
    }
```

Рисунок 4.6 – Графічний вигляд моделі веб-системи

Джерело: побудовано автором (знімок з екрану)

Дані для доступу до бази даних зберігаються у файлі властивостей. Доступ до з'єднання через файл властивостей отримується за допомогою класу ConnectionPool. Метод отримання зєднання та файл налаштувань доступу до бази даних наведено на рисунках 4.7-4.8.

```
public Connection getConnection() throws IOException, SQLException, ClassNotFoundException {
    Properties dbProps = new Properties();
    Class.forName("com.mysql.jdbc.Driver");
    InputStream inputStream = getClass().getClassLoader().getResourceAsStream(DB_CONNECTION_POOL_PROPERTIES);
    dbProps.load(inputStream);
    return DriverManager.getConnection(dbProps.getProperty(URL_PROPS_KEY),
            dbProps.getProperty(USER_PROPS_KEY),
            dbProps.getProperty(PASS_PROPS_KEY));
}
```

Рисунок 4.7 – Налаштування з'єднання з базою даних

Джерело: побудовано автором (знімок з екрану)

```
1   url = jdbc:mysql://localhost:3307/library_db
2   characterEncoding=UTF-8
3   useUnicode = true
4   autoReconnect = true
5   user = java_web_app
6   password = 80maximWEB
```

Рисунок 4.8 – Параметри доступу до бази даних

Джерело: побудовано автором (знімок з екрану)

4.4 Результати розробки web-системи

В результаті розробки веб-системи було розроблено наступні сторінки та їх функціонал:

Головна сторінка каталогу крім перегляду каталогу дозволяє виконувати його сортування за назвою, датою випуску, автором, видавництвом (рисунок 4.9-4.10). На сторінці також наявний елемент зміни мови інтерфейсу, який змінює мову на всіх подальших сторінках системи(рисунок 4.11-4.12).

Рисунок 4.9 – Сортування каталогу за назвою книги у алфавітному порядку

Джерело: побудовано автором (знімок з екрану)

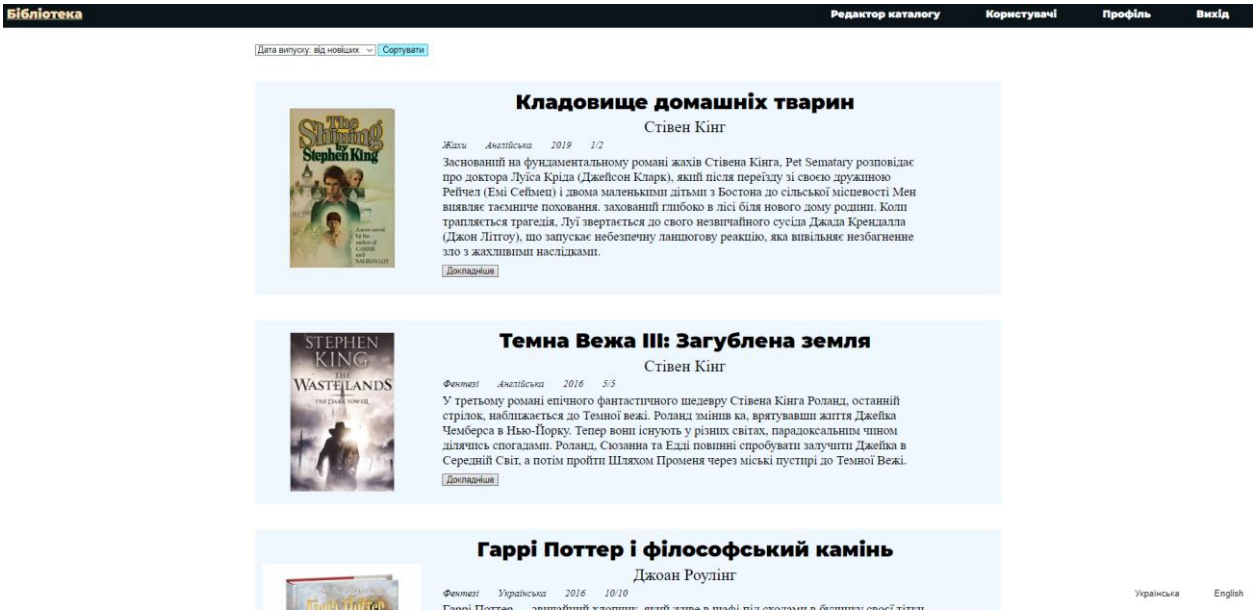


Рисунок 4.10 – Сортування каталогу за датою випуску, від новіших

Джерело: побудовано автором (знімок з екрану)

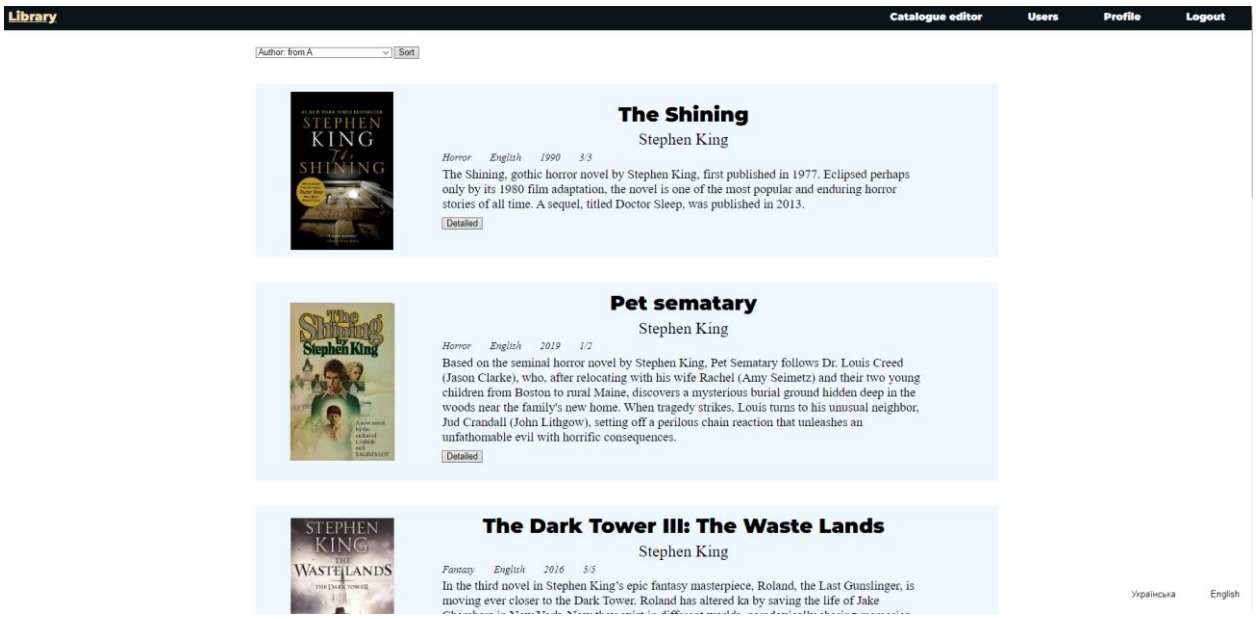


Рисунок 4.11 – Зміна мови на сторінці каталогу

Джерело: побудовано автором (знімок з екрану)

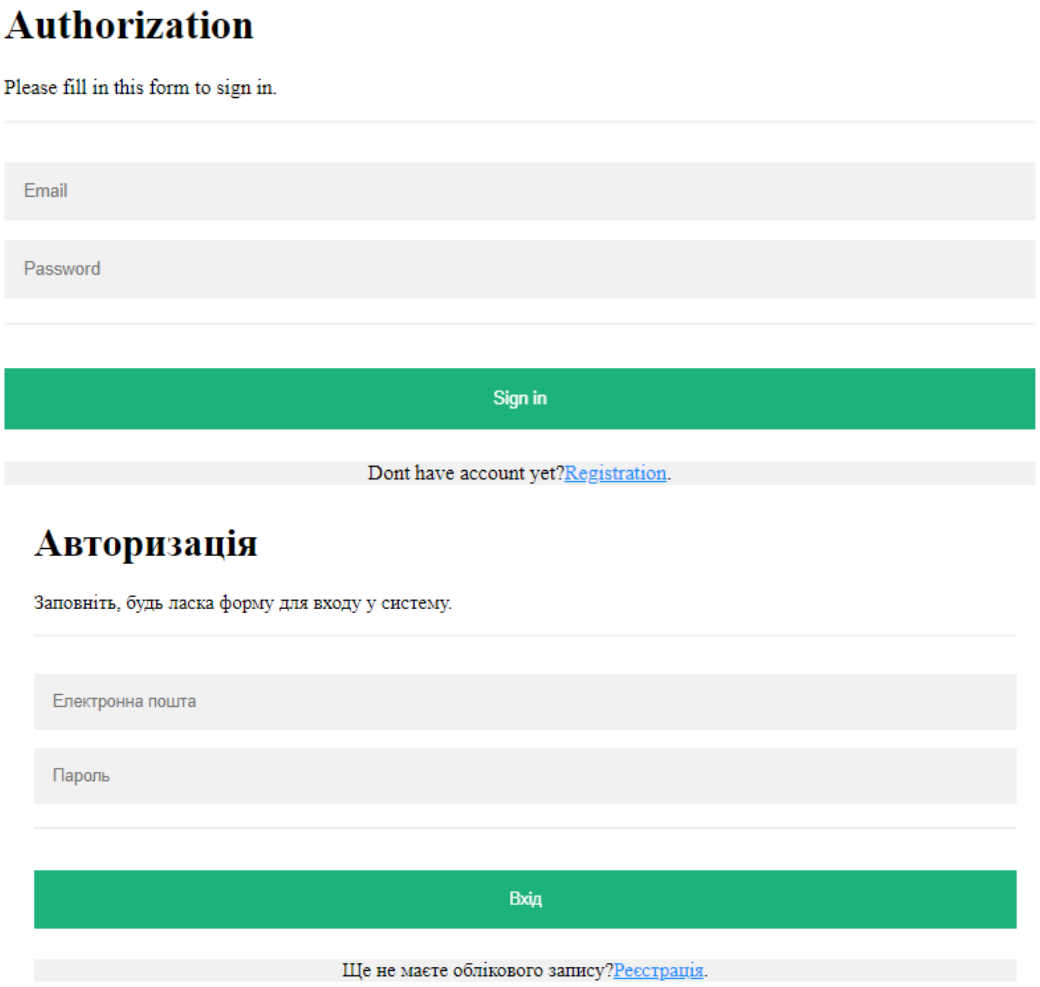


Рисунок 4.12 – Зміна мови на сторінці авторизації

Джерело: побудовано автором (знімок з екрану)

Сторінка замовлення книги, куди потрапляє користувач при виборі книги з каталогу, показана на рисунках 4.13-4.14. Тут він бачить всю повну інформацію про книгу, має можливість оформити замовлення на неї.

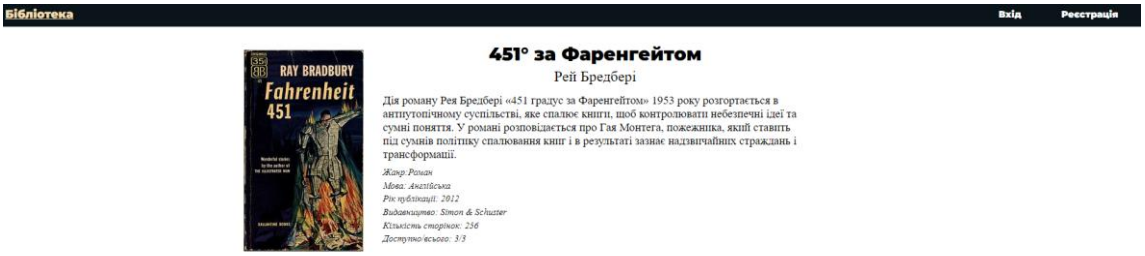


Рисунок 4.13 – Вигляд сторінки замовлення для неавторизованого користувача

Джерело: побудовано автором (знімок з екрану)

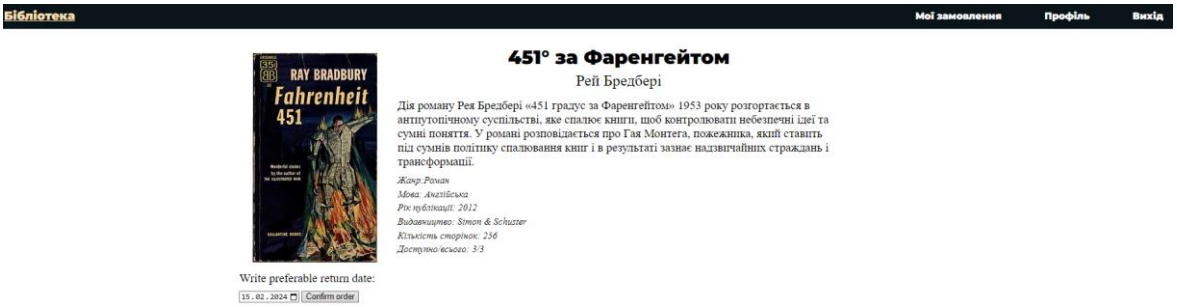


Рисунок 4.14 – Сторінка замовлення після авторизації

Джерело: побудовано автором (знімок з екрану)

Сторінка профілю, де відображається реєстраційна інформація користувача, наведена на рисунку 4.15. Для звичайних користувачів доступна можливість поповнення рахунку для подальшої оплати штрафів.

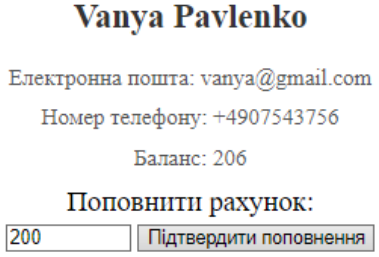


Рисунок 4.15 – Сторінка профілю користувача

Джерело: побудовано автором (знімок з екрану)

Сторінка замовлень користувача, де він може переглянути історію своїх замовлень, повернути книги, сплатити штрафи по простроченим замовленням, скасувати ще не видані йому замовлення, показана на рисунку 4.16.

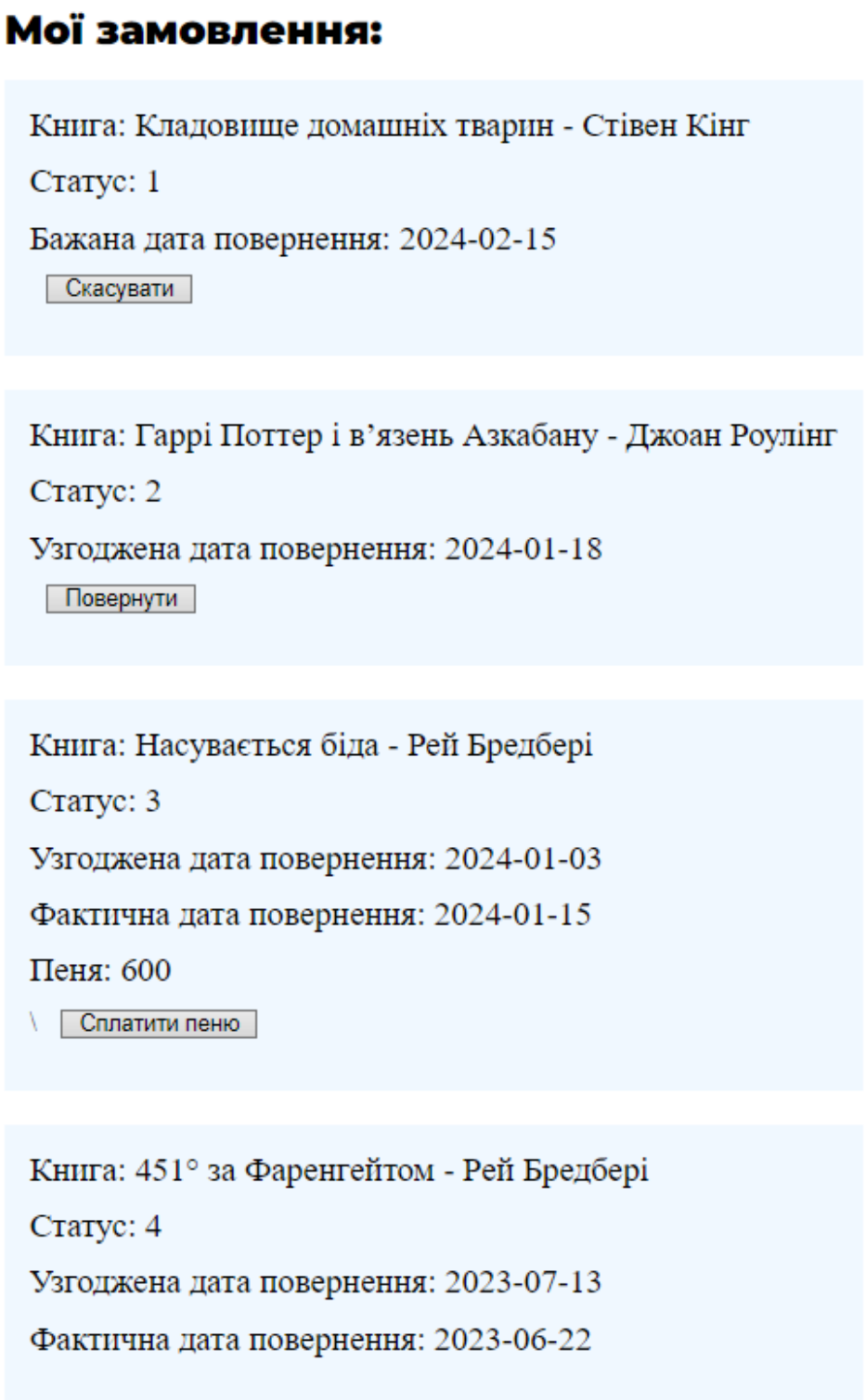


Рисунок 4.16 – Список замовлення користувача

Джерело: побудовано автором (знімок з екрану)

Сторінки реєстрації та авторизації надають можливість користувачам створити новий обліковий запис, або виконати вхід в уже існуючий (рисунки 4.17-4.18).

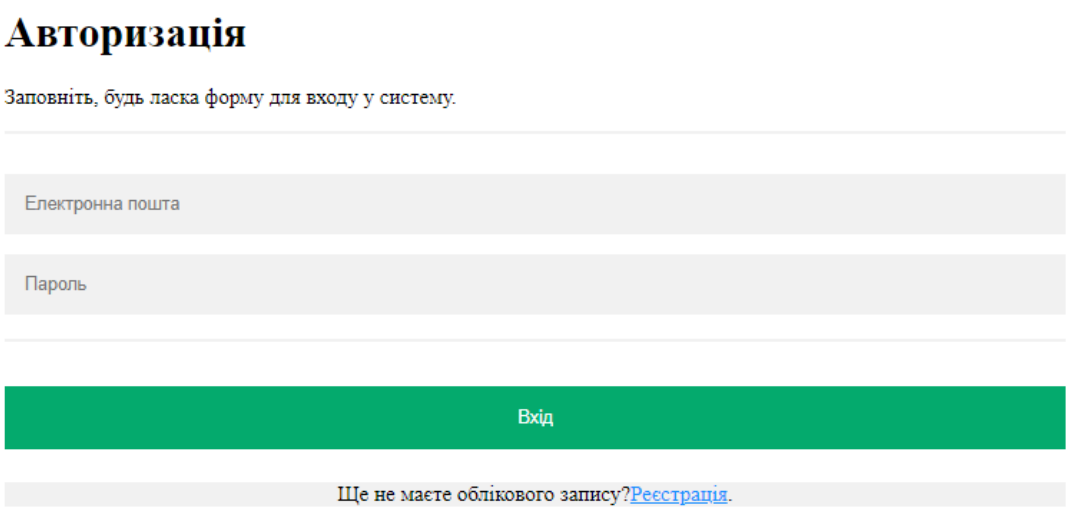


Рисунок 4.17 – Сторінка авторизації користувача

Джерело: побудовано автором (знімок з екрану)

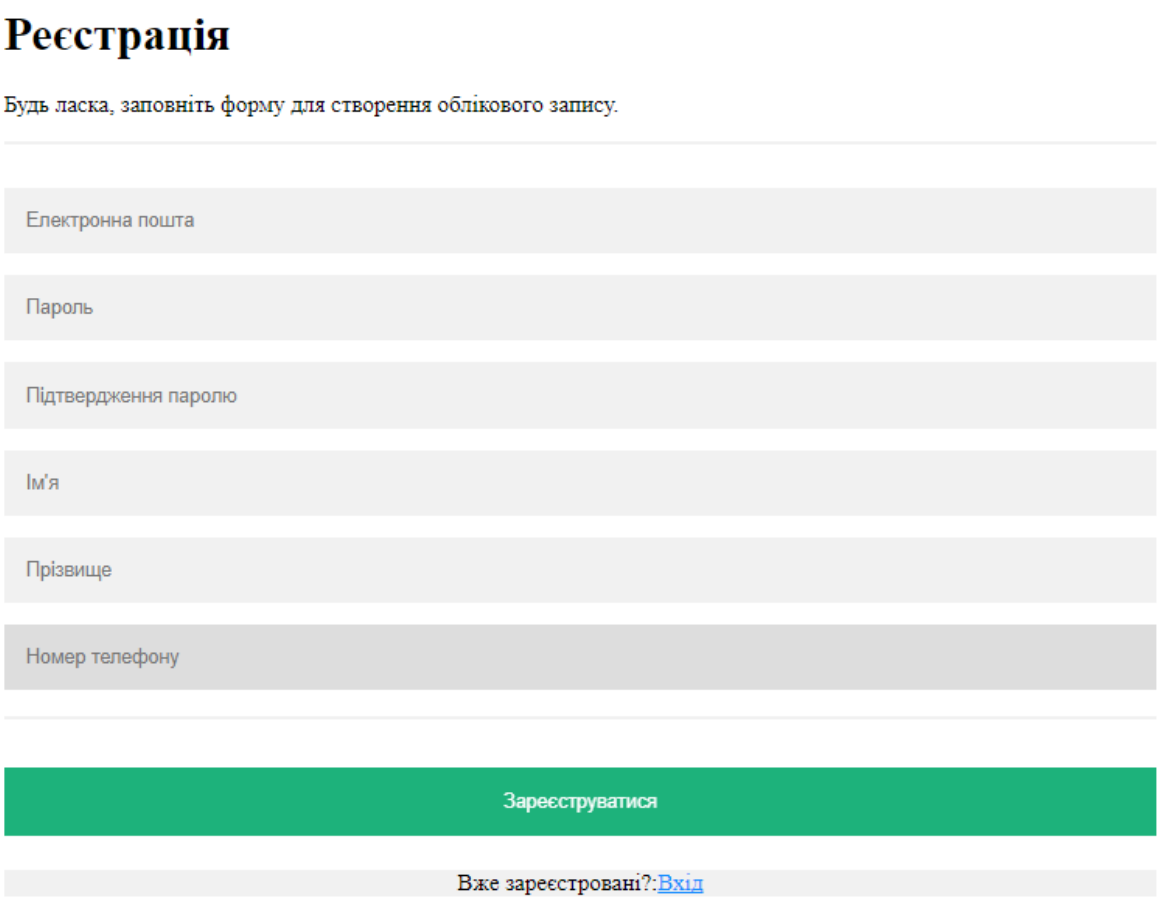


Рисунок 4.18 – Сторінка реєстрації користувача

Джерело: побудовано автором (знімок з екрану)

Сторінка нових замовлень, наведена на рисунку 4.19, дає можливість бібліотекарю змінювати планову дату повернення книги, та видавати замовлення користувачу.

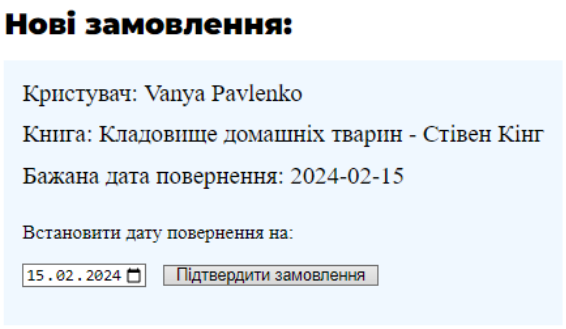


Рисунок 4.19 – Сторінка нових замовлень

Джерело: побудовано автором (знімок з екрану)

Бібліотекар системи також має доступ до списку всіх користувачів та їх замовлень. Вигляд сторінки наведено на рисунку 4.20.

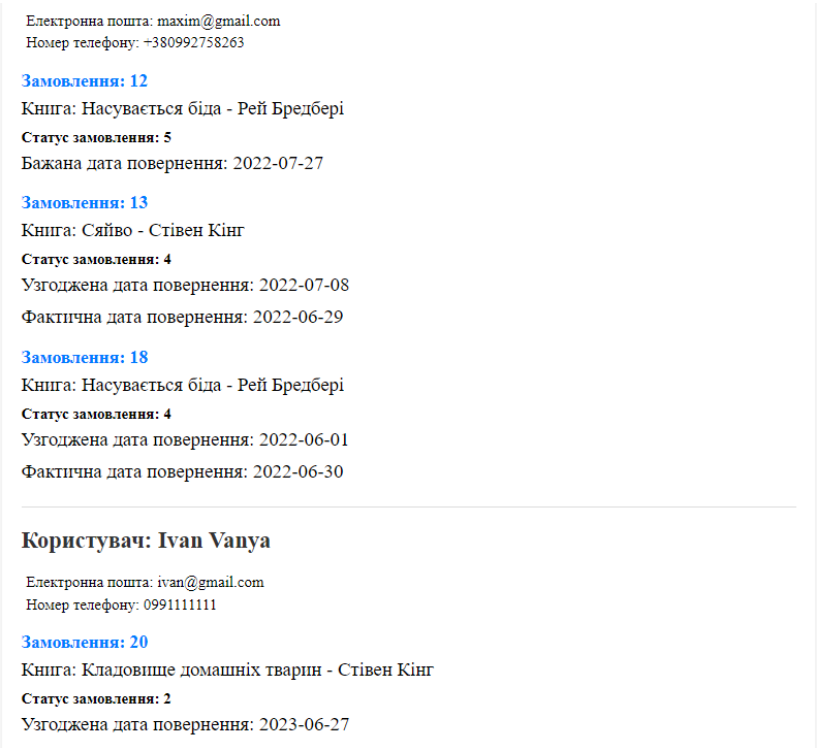


Рисунок 4.20 – Список користувачів та їх замовлень

Джерело: побудовано автором (знімок з екрану)

4.5 Адміністрування web-системи

Доступ до редагування каталогу книг має тільки адміністратор системи. Вигляд сторінки наведено на рисунку 4.21. На цій сторінці адміністратор має можливість редагувати книги (рисунок 4.22), видаляти книги додавати нову літературу та необхідні для неї дані – автора, видавництво, жанр (рисунок 4.23-4.24).

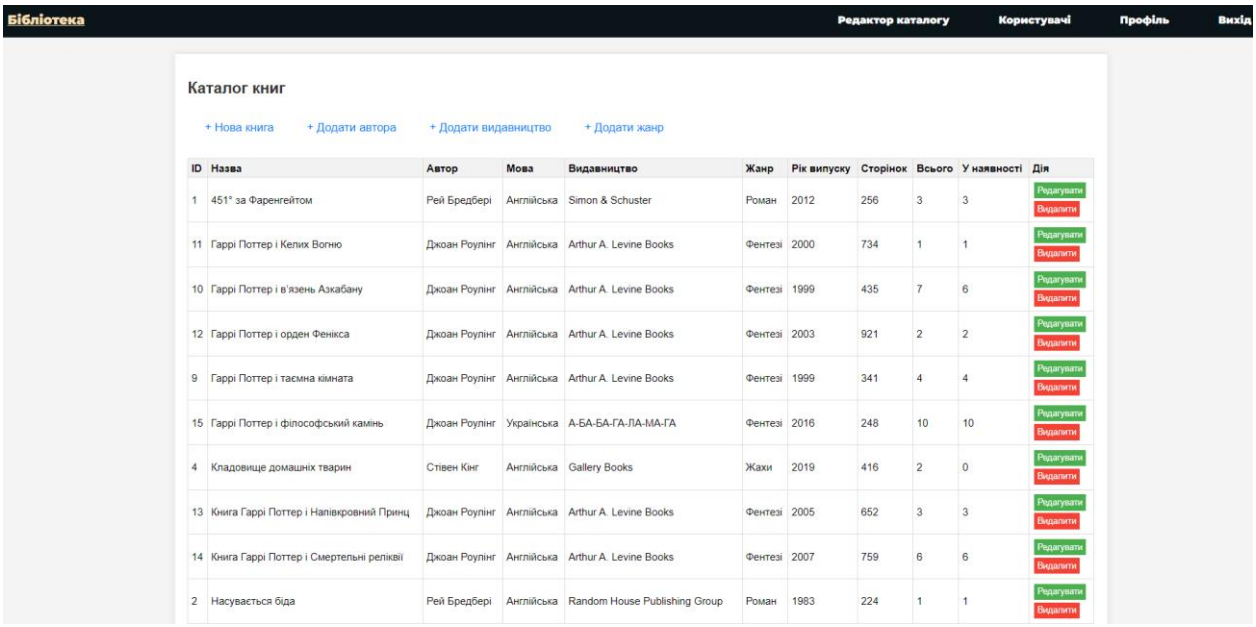


Рисунок 4.21 – Сторінка редагування каталогу літератури

Джерело: побудовано автором (знімок з екрану)

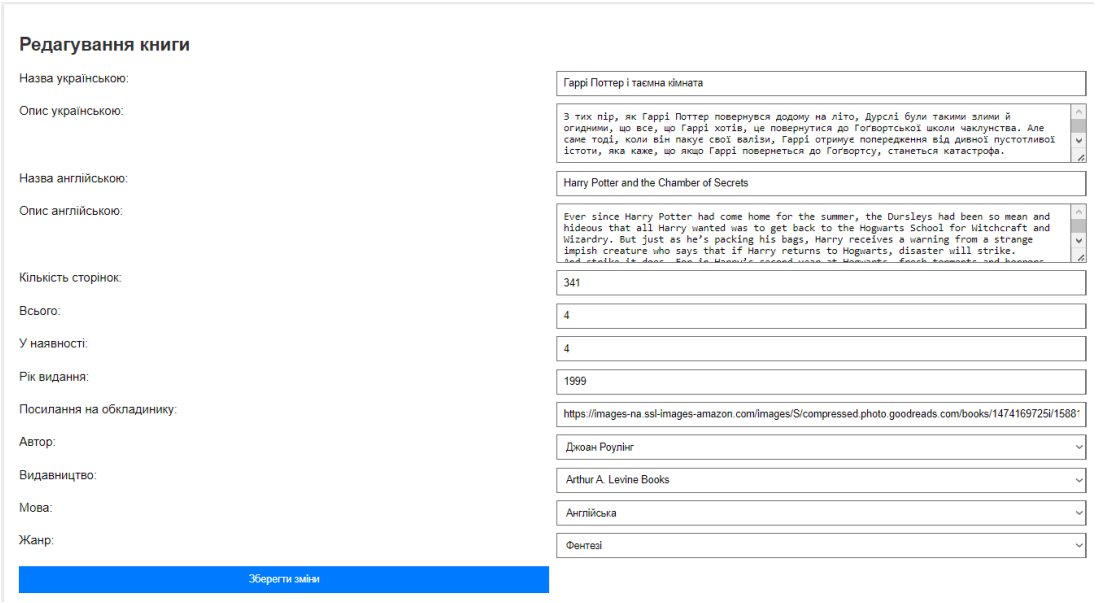


Рисунок 4.22 – Сторінка редагування книги

Джерело: побудовано автором (знімок з екрану)

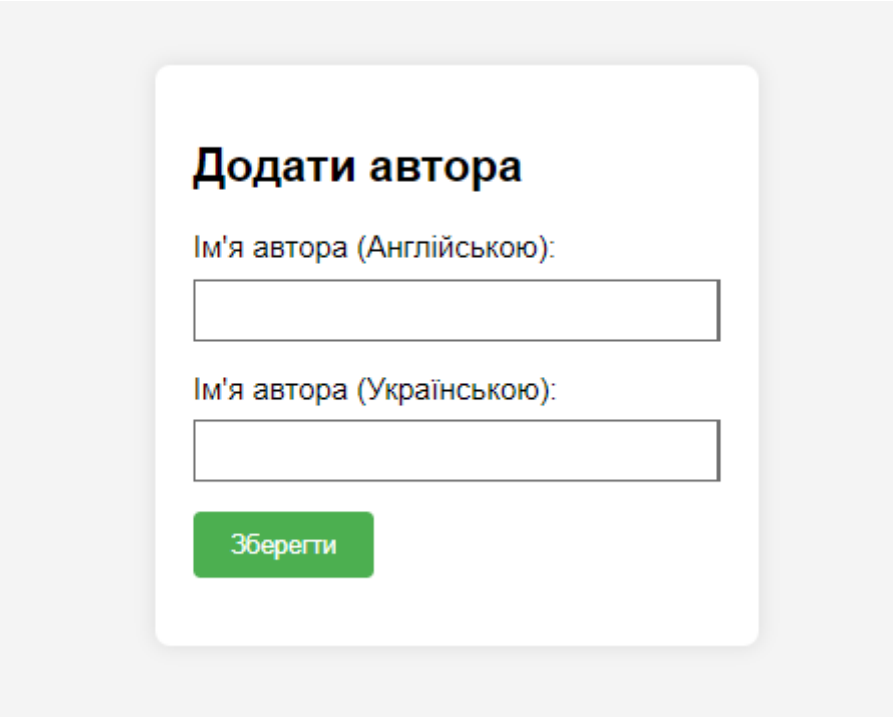


Рисунок 4.23 – Сторінка додавання автора

Джерело: побудовано автором (знімок з екрану)

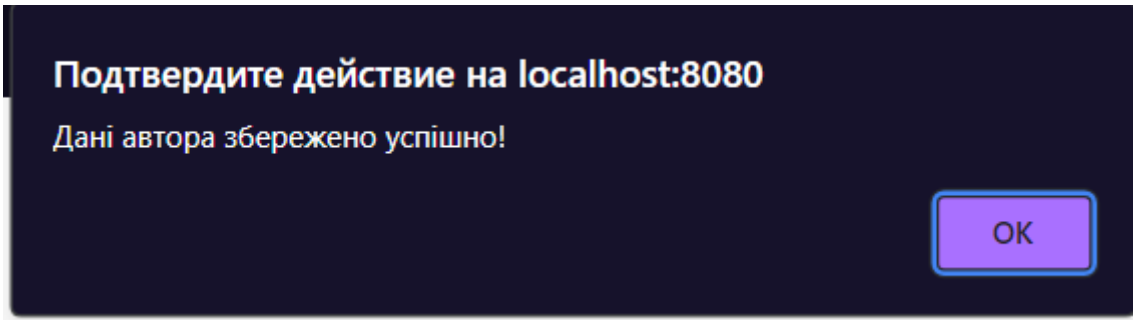


Рисунок 4.24 – Результат додавання нового автора

Джерело: побудовано автором (знімок з екрану)

4.6 Тестування web-системи

Після реалізації web-орієнтованої системи було проведено функціональне тестування, де перевірено роботу головних функції сайту. Були протестовані кнопки й посилання, а також валідація даних при заповненні форм.

На рисунку 4.25 наведено приклади реакції сторінок на некоректно введені дані в форму реєстрації. Частина інформації перевіряється без участі бізнес-логіки проекту за допомогою передбаченого мовою html функціоналу [35].

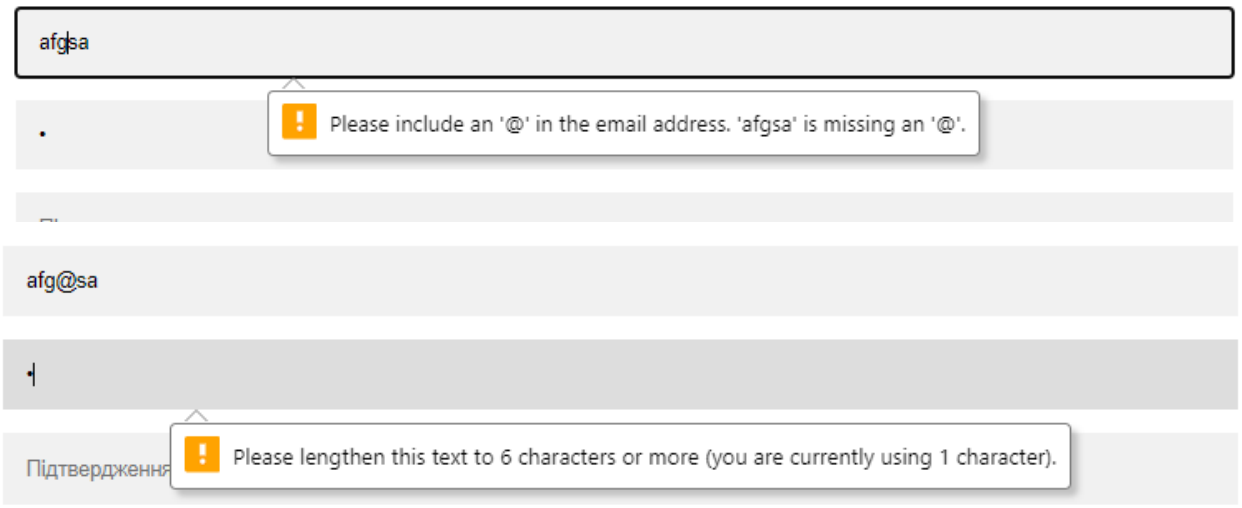
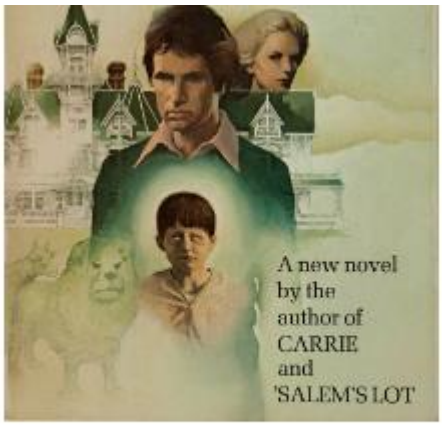


Рисунок 4.25 – Приклади підказок при некоректному заповненні форм

Джерело: побудовано автором (знімок з екрану)

На рисунку 4.26 наведено результат спроби оформлення замовлення на книгу, всі екземпляри якої вже були видані користувачам.



місцевості Мен виявляє таємниче п
дому родини. Коли трапляється тра
Джада Крендалла (Джон Літгоу), щ
вивільняє незбагненне зло з жахлив

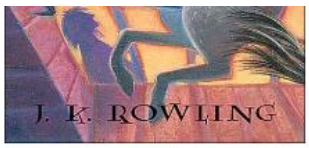
Жанр:Жахи
Мова: Англійська
Рік публікації: 2019
Видавництво: Gallery Books
Кількість сторінок: 416
Доступно/всього: 0/2

Sorry, there are no free books

Рисунок 4.26 – Оформлення замовлення на книгу, яких вже немає у наявності

Джерело: побудовано автором (знімок з екрану)

При оформленні замовлення, як і при видачі замовлення користувачу бібліотекарем, дата повернення не може бути вже минувшою датою (рисунок 4.27-4.28).



Мова: Англійська
Рік публікації: 1999
Видавництво: Arthur A. Levine Books
Кількість сторінок: 435
Доступно/всього: 6/7

Write preferable return date:

01/15/2024 Confirm order

January 2024

Su	Mo	Tu	We	Th	Fr	Sa
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Clear Today

Рисунок 4.27 – Вибір минувшої дати неможливий при оформленні замовлення

Джерело: побудовано автором (знімок з екрану)

Нові замовлення:

Кристувач: Vanya Pavlenko

Книга: Кладовище домашніх тварин - Стівен Кінг

Бажана дата повернення: 2024-02-15

Встановити дату повернення на:

01/15/2024 Підтвердити замовлення

January 2024

Su	Mo	Tu	We	Th	Fr	Sa
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Clear Today

Рисунок 4.28 – Вибір минувшої дати неможливий при видачі замовлення

Джерело: побудовано автором (знімок з екрану)

ВИСНОВОК

У ході виконання кваліфікаційної роботи магістра було проаналізовано предметну область, розглянуто літературні джерела на тему оптимізації надання бібліотечних послуг за допомогою інформаційних технологій. За результатами аналізу предметної області та аналізу існуючих методів діджиталізації бібліотечних процесів було підготовано специфікацію для розробки веб-системи підтримки надання бібліотечних послуг.

Наступним кроком до виконання поставленої задачі стало формування мети та вибір засобів реалізації – розробка веб-системи підтримки надання бібліотечних послуг на основі функціоналу Java EE з використанням технології сервлетів для забезпечення взаємодії між клієнтом та сервером.

Під час проектування розроблюваної системи було створено контекстну діаграму в нотації IDEF0 для демонстрації бізнес-процесів системи, а також її декомпозицію першого рівня. Додатково була розроблена діаграма варіантів використання, яка допомогла визначити сценарії взаємодії користувачів з веб-системою.

Завдяки схемі бази даних, вдалось реалізувати структуризацію даних в базі даних розроблюваного системи.

Результатом виконання роботи стала web-система підтримки надання бібліотечних послуг, розроблена у відповідності до функціональних вимог. За візуальне представлення системи відповідають JSP сторінки, а бекенд реалізований мовою програмування Java. Отриману систему було протестовано на коректність роботи основного функціоналу та валідацію користувацьких даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хрущ С. Інноваційний медіапростір сучасної бібліотеки: аксіологічні орієнтири функціонування. Український журнал з бібліотекознавства та інформаційних наук. 2022. № 9. С. 70–78. URL: http://librinfosciences.knukim.edu.ua/article/view/259153/255964 (дата звернення: 12.11.2023).

2. Крохмаль І., Лєсовець Н. Сучасна бібліотека україни: від книгозбірні до осередку духовності й культури. Education and science of today: intersectoral issues and development of sciences. 2021. URL: https://doi.org/10.36074/logos-19.03.2021.v1.11 (дата звернення: 12.11.2023).

3. Горова С. Інформаційна еволюція. Бібліотечні установи та особа. Наукові праці національної бібліотеки україни імені В.І. вернадського. 2017. Вип. 48. С. 339–350.

4. Сидоренко Н. О. Діджиталізація: електронні адміністративні послуги. Дніпровський науковий часопис публічного управління, психології, права. 2021. № 4. С. 11–15. URL: https://doi.org/10.51547/ppp.dp.ua/2021.4.2 (дата звернення: 12.11.2023).

5. Yatsishin A. V., Tkachenko V. A. Електронна бібліотека національної академії педагогічних наук україни як організаційне нововведення. Information technologies and learning tools. 2012. Т. 26, № 6. URL: https://doi.org/10.33407/itlt.v26i6.584 (дата звернення: 12.11.2023).

6. Овсієнко А. Адаптація бібліотек до реалій цифрового простору. Society. Document. Communication. 2021. Т. 12, № 12. С. 255–269. URL: https://doi.org/10.31470/2518-7600-2021-12-255-269 (дата звернення: 12.11.2023).

7. Tomer C. Library automation: core concepts and practical systems analysis. 3rd ed. Technical services quarterly. 2015. Т. 32, № 4. С. 467–469. URL: https://doi.org/10.1080/07317131.2015.1059708 (дата звернення: 12.11.2023).

8. Fernández Ramos A. The web-based library: challenges and opportunities. Investigación bibliotecológica: archivonomía, bibliotecología e información. 2016. Т. 30, № 69. С. 11–14. URL: https://doi.org/10.1016/j.ibbai.2016.10.014 (дата звернення: 12.11.2023).

9. Eisele M., Vinto N. Modernizing enterprise java. O'Reilly Media, Incorporated, 2021.

10. Juneau J. New servlet features. Introducing java EE 7. Berkeley, CA, 2013. С. 1–14. URL: https://doi.org/10.1007/978-1-4302-5849-0_1 (дата звернення: 12.11.2023).

11. Pursnani V. An introduction to Java servlet programming. XRDS: crossroads, the ACM magazine for students. 2001. Т. 8, № 2. С. 3–7. URL: https://doi.org/10.1145/567155.567157 (дата звернення: 12.11.2023).

12. Shing Wai Chan, Ed Burns. Java servlet specification. Java EE. URL: https://javaee.github.io/servlet-spec/downloads/servlet-4.0/servlet-4_0_FINAL.pdf (дата звернення: 12.11.2023).

13. Arjan Tijms, Teo Bais, Werner Keil. Security history. Apress, Berkeley, CA, 2022.

14. Eyal Salman H. Identification multi-level frequent usage patterns from APIs. Journal of systems and software. 2017. Т. 130. С. 42–56. URL: https://doi.org/10.1016/j.jss.2017.05.039 (дата звернення: 14.11.2023).

15. Fengel J. Semantic technologies for aligning heterogeneous business process models. Business process management journal. 2014. Т. 20, № 4. С. 549–570. URL: https://doi.org/10.1108/bpmj-07-2013-0085 (дата звернення: 14.11.2023).

16. Ehmes S., Fritsche L., Schürr A. SimSG: rule-based simulation using stochastic graph transformation. The journal of object technology. 2019. Т. 18, № 3. С. 1:1. URL: https://doi.org/10.5381/jot.2019.18.3.a1 (дата звернення: 14.11.2023).

17. Al-Btoush A. A.-S. Extracting entity relationship diagram (ERD) from english sentences. International journal of database theory and application. 2015. Т. 8, № 2. С. 235–244. URL: https://doi.org/10.14257/ijdta.2015.8.2.22 (дата звернення: 14.11.2023).

18. Bloch J. Effective Java. 2-ге вид. Pearson Education, Limited, 2017. 351 с.

19. Apache Tomcat 8 (8.5.96) - JNDI Datasource How-To. Apache Tomcat® - Welcome!. URL: https://tomcat.apache.org/tomcat-8.5-doc/jndi-datasource-examples-howto.html (дата звернення: 19.11.2023).

20. Entity to DTO conversion for a spring REST API | baeldung. Baeldung. URL: https://www.baeldung.com/entity-to-and-from-dto-for-a-java-spring-application (дата звернення: 19.11.2023).

21. Gupta S., Bashambu S. Implementation of java frameworks and apis for web applications. International journal of scientific research in science, engineering and technology. 2020. С. 427–432. URL: https://doi.org/10.32628/ijsrset207284 (дата звернення: 19.11.2023).

22. Karaca B. The DAO design pattern in java / spring boot. DEV Community. URL: https://dev.to/karaca19/the-dao-design-pattern-in-java-spring-boot-2l8o (дата звернення: 19.11.2023).

23. Khorikov V. Unit testing principles, practices, and patterns. Manning Publications Company, 2020. 304 с.

24. Kurniawan B. Java web development with servlets, jsp, and ejb. 2-ге вид. Sams, 2004. 896 с.

25. Layka V. Building web applications using servlets and JSP. Learn java for web development. Berkeley, CA, 2014. С. 43–103. URL: https://doi.org/10.1007/978-1-4302-5984-8_2 (дата звернення: 19.11.2023).

26. McGregor M. 4 design patterns you should know for web development: observer, singleton, strategy, and decorator. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/4-design-patterns-to-use-in-web-development/ (дата звернення: 19.11.2023).

27. Pankaj. Java web application tutorial for beginners. DigitalOcean | Cloud Hosting for Builders. URL: https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners (дата звернення: 19.11.2023).

28. The DAO pattern in java | baeldung. Baeldung. URL: https://www.baeldung.com/java-dao-pattern (дата звернення: 19.11.2023).

29. Tutorial: Your first RESTful web service | IntelliJ IDEA. IntelliJ IDEA Help. URL: https://www.jetbrains.com/help/idea/creating-and-running-your-first-restful-web-service.html#troubleshooting (дата звернення: 19.11.2023).

30. Understanding web applications, servlets, and jsps. Oracle. URL: https://docs.oracle.com/cd/E14571_01/web.1111/e13712/basics.htm#WBAPP123 (дата звернення: 19.11.2023).

31. Späth P. Beginning Java MVC 1.0: Model View Controller Development to Build Web, Cloud, and Microservices Applications. Beginning Java MVC 1.0. 2021.

32. Design Patterns. Refactoring and Design Patterns. URL: https://refactoring.guru/design-patterns (дата звернення: 16.01.2024).

33. Shvets A. Dive Into Design Patterns. Refactoring.Guru, 2018. 406 с., с. 24-28

34. Demba M. Algorithm for relational database normalization up to 3NF //International Journal of Database Management Systems. – 2013. – Т. 5. – №. 3. – С. 39.

35. Kim H., Doh K. G., Schmidt D. A. Static validation of dynamically generated HTML documents based on abstract parsing and semantic processing //Static Analysis: 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings 20. – Springer Berlin Heidelberg, 2013. – С. 194-214.

ДОДАТОК А
ПЛАНУВАННЯ РОБІТ

Інформаційні технології щодня закріплюють своє місце в житті сучасної людини. Навряд чи вдасться знайти якусь галузь де за останні роки не відбулося ніяких змін пов’язаних з технологічним розвитком. Найцікавіше те, що впровадження подібних змін не зупиняється ні на хвилину. Щодня суспільство стикається з новітніми рішеннями, покликаними зробити наше життя більш зручним. Якщо замислитись, то функціонування будь-якої структури чи організації може мути оптимізоване за рахунок впровадження інформаційних технологій у їх функціонування.

Не стала винятком і звичайна бібліотека. Раніше читачам для того щоб отримати бажану книгу потрібно було прийти до закладу, запитати у бібліотекара про її наявність, почекати щонайменше декілька хвилин доки працівник перевірить її наявність, і після чого могло виявитися що бажаного примірника немає у наявності. Інформаційні системи бібліотек можуть покращити доступ до інформації, зробити пошук і замовлення книг більш простими, а також розширити можливості взаємодії користувачів із працівниками бібліотеки. Такий підхід підтримує концепцію сучасної бібліотеки, яка прагне надавати інноваційні та ефективні послуги своїм відвідувачам.

Деталізація мети проекту методом SMART. Визначення мети проекту за допомогою SMART-методу допомагає чітко визначити призначення кінцевого продукту та обмеження під час його розробки. Крім того, мета, сформована за цим методом дає можливість визначити напрямок, у якому необхідно рухатись для її досягнення. Результати деталізації методом SMART розміщені у таблиці Б.1.

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific (конкретна)	Створити веб-систему підтримки надання бібліотечних послуг.
Measurable (вимірювана)	Розробити веб-систему, використовуючи мінімальну кількість ресурсів.
Achievable (досяжна, узгоджена)	Для виконання проекту наявні необхідні знання мови програмування Java, середовища Intellij Idea, баз даних MySQL та навичок написання документації. Враховуючи доступні ресурсні можливості та обмеження мета є такою, яку можливо досягти.
Relevant (реалістична)	Створена веб-система покращить доступ до інформації, зробить пошук і замовлення книг більш простими, а також розширить можливості взаємодії користувачів із працівниками бібліотеки.
Time-framed (обмежена в часі)	Ціль має часове обмеження. Термін досягнення мети проекту складає один місяць.

Джерело: побудовано автором

Планування змісту робіт. WBS (Work Breakdown Structure – Ієрархічна структура робіт) – це графічне розбиття проекту на елементарні задачі, які пов’язані за ієрархією з фінальним продуктом проекту. Структура декомпозиції робіт дозволяє наглядно зобразити весь об’єм проекту, спростити організацію командної роботи, дозволяє коректно оцінити затрати та час, необхідний для виконання проекту. Крім того, WBS дозволяє структурувати та розділити проект на основні компоненти, якими легко керувати.

На найвищому (першому) рівні розміщений продукт проекту. Основні компоненти, що забезпечують досягнення мети проекту, розташовуються на другому рівні декомпозиції. Декомпозиція робіт виконується до тих пір, поки вони не стануть елементарними.

Елементарні роботи – це дії, які мають однозначний чіткий результат, відповідальність за які лежить на одній конкретній особі, для яких можна обчислити необхідні витрати ресурсів. На рисунку А.1 представлено WBS з розробки веб-системи надання бібліотечних послуг.

В результаті проведення декомпозиції проекту було отримано п’ять основних

напрямів діяльності – підготовка документації, розробка веб-дадатку, наповнення його контентом, тестування та введення в реалізацію. Кожен з цих напрямів має свої підзадачі.

На етапі підготовки документації відбувається збір інформації про предметну область продукту, аналіз вимог до нього та розробка чіткого й зрозумілого ТЗ, після чого починається розробка Під час етапу розробки визначається інтерфейс майбутнього системи, програмно реалізується його функціонал, та налаштовується взаємодія з базою даних. Коли продукт вже отримав весь необхідний функціонал, відбувається його наповнення даним, після чого відбувається стадія тестування, під час якої важливо від лиця потенційного користувача перевірити працездатність всіх його модулів. Якщо ж стадія тестування пройшла успішно, і значних недоліків знайдено не було, то продукт переходить на етап впровадження в дію і в результаті потрапляє до замовника.

Наступним етапом після декомпозиції процесів є розробка організаційної структури виконавців або OBS. Для успішного управління проектом необхідно знати яка організація чи виконавець відповідні за кожен пакет чи рівень робіт. Саме для цього і використовується OBS. В цій структурі на верхньому рівні знаходиться керівник проекту, а на нижчих організаціях розташовані організації, відділи, чи працівники що відповідають за конкретні роботи чи пакети робіт з WBS-структури.

На рисунку А.2 представлено організаційну структуру планування проекту.

Список виконавців, що функціонують в проекті описано в таблиці А.2.

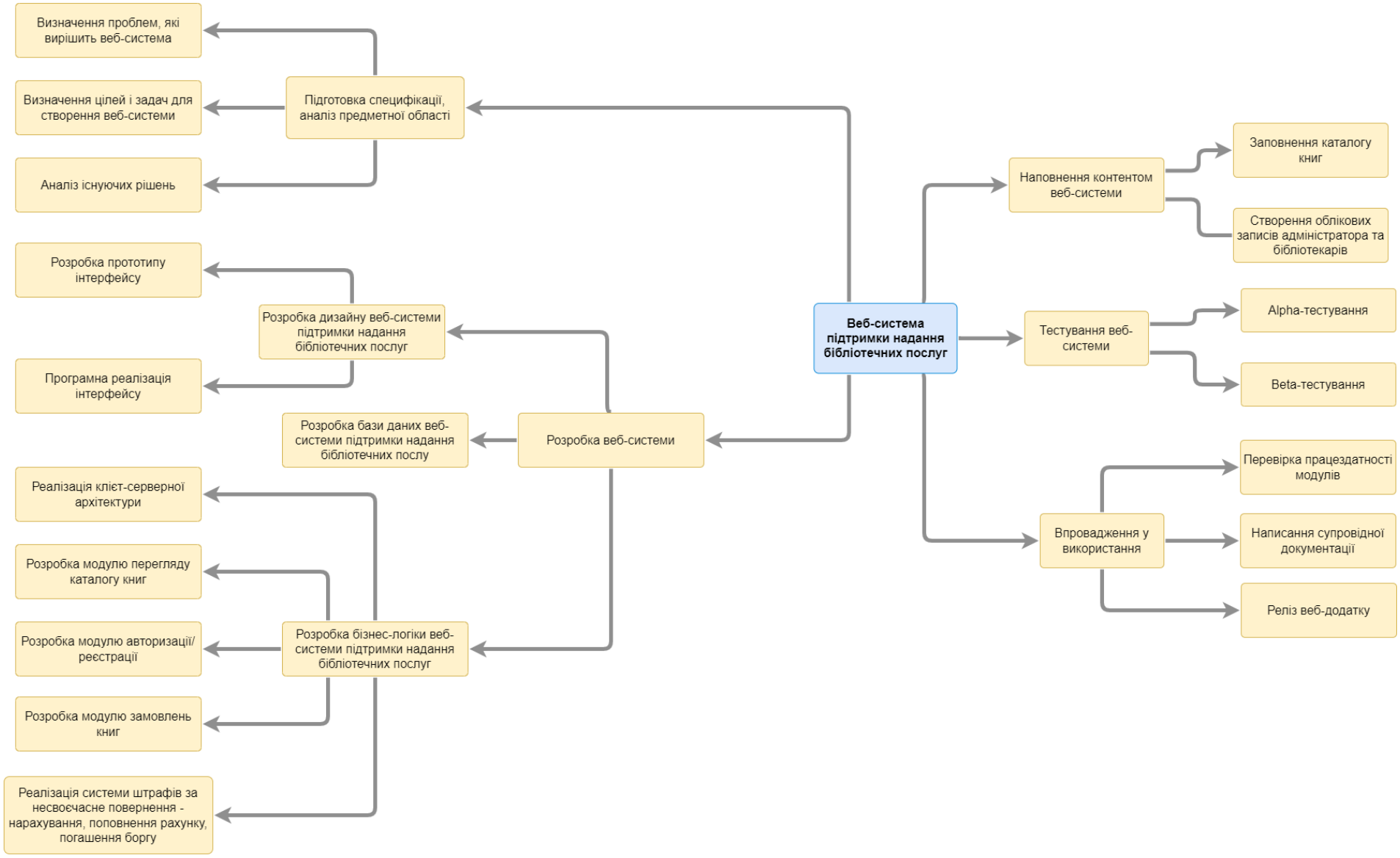


Рисунок А.1 – WBS-структура робіт проекту

Джерело: побудовано автором (знімок з екрану)

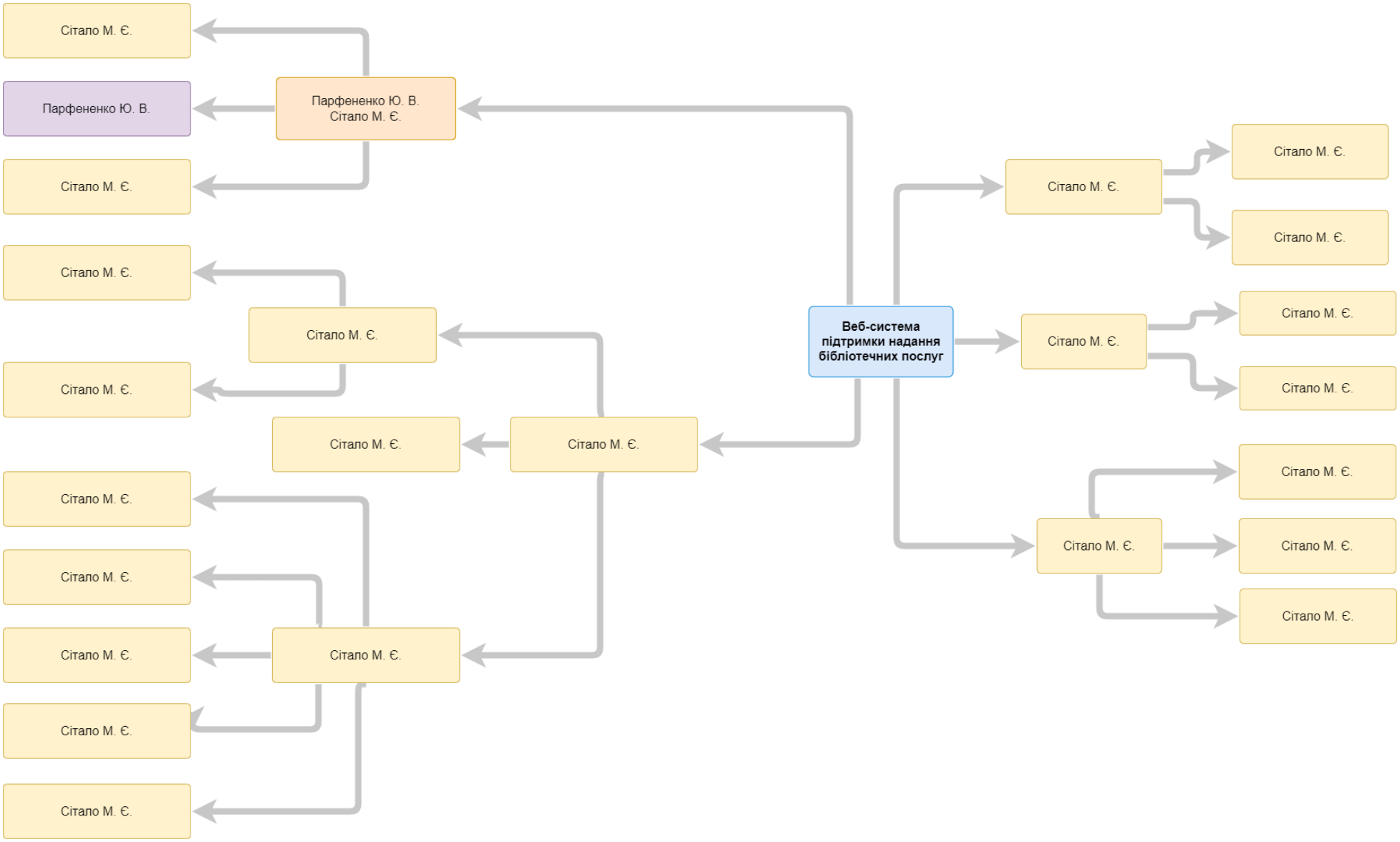


Рисунок А.2 – OBS-структура робіт проекту
Джерело: побудовано автором (знімок з екрану)

Таблиця А.2 – Виконавці проекту

Роль	Ім’я	Проектна роль
Розробник	Сітало М.Є.	Виконує розробку програмних модулів та дизайну системи.
Проектувальник	Сітало М.Є.	Виконує проектування бази даних та розробляєструктуру веб-системи.
Тестувальник	Сітало М.Є.	Відповідає за тестування функціоналу та дизайнусистеми.
Керівник проекту	Парфененко Ю. В.	Формує цілі та задачі проекту
Менеджер проекту	Сітало М.Є.	Відповідає за дотримання термінів, розподіл ресурсів та завдань між учасниками, написання документації. Виконує збір та аналіз даних.

Джерело: побудовано автором

Для відстеження ходу виконання проекту дуже зручно використовувати його календарний план (діаграму Ганта). Вона дозволяє в простій і зрозумілій формі зобразити структуру проекту, відповідальних за кожне завдання осіб, часові обмеження стосовно кожної задачі.

Календарний графік проекту представлено на рисунках Б.3-Б.5.

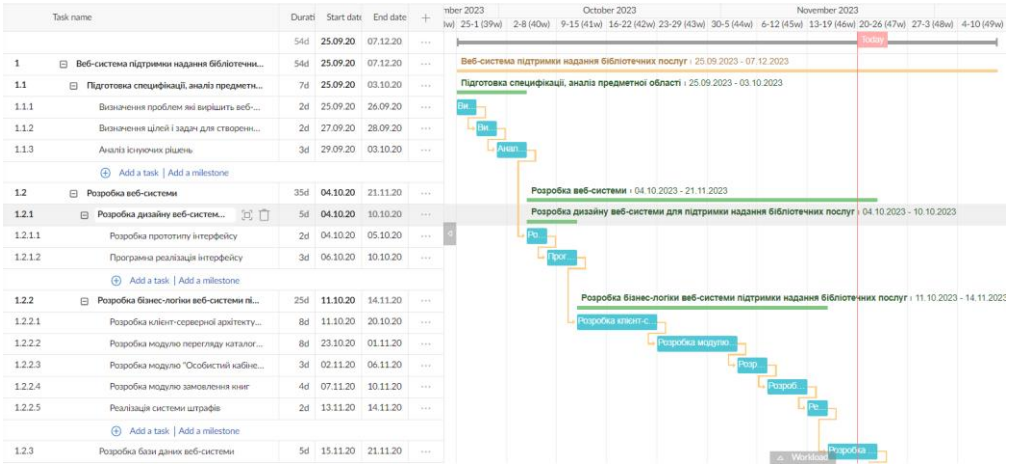


Рисунок А.3 – Календарний графік проекту, частина 1

Джерело: побудовано автором (знімок з екрану)

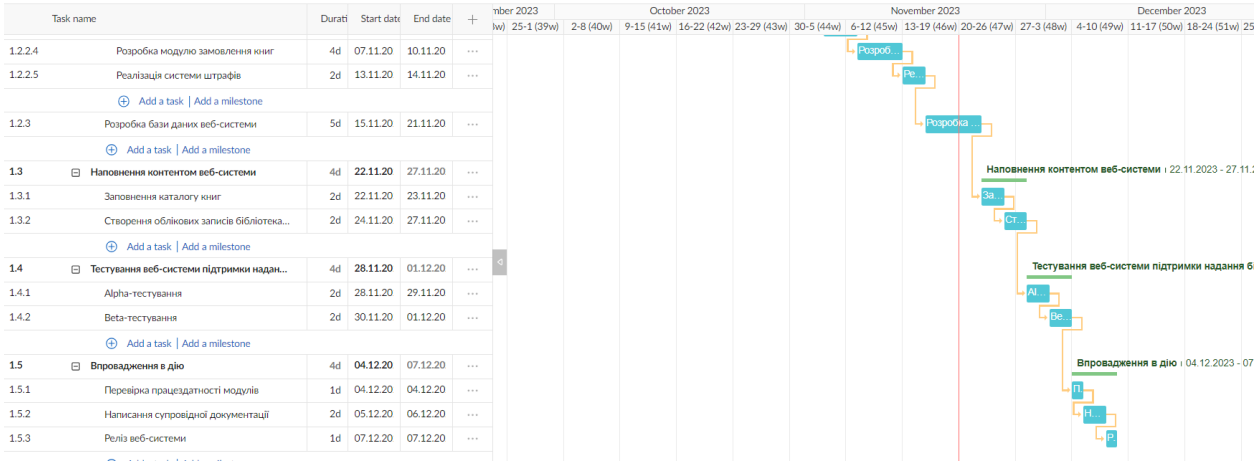


Рисунок А.4 – Календарний графік проекту, частина 2
Джерело: побудовано автором (знімок з екрану)

Управління ризиками є невід’ємною частиною будь-якого проекту. Визначення можливих проблем на шляху до розробки фінального продукту допоможе підготуватися до потенційних загроз та мінімізувати збитки, що допоможе завершити проект вчасно і якісно. Інформація про ідентифіковані ризики представлена у таблиці А.3

Таблиця Б.3 – Ідентифікація ризиків

№	Назва (опис) ризику	Ймовірність (0,1-0,9)	Вплив (0,05-0,8)	Ранг
1	Неправильно сформоване ТЗ проекту	0,5	0,5	0,25
2	Непорозуміння між розробником та замовником	0,4	0,5	0,2
3	Нечіткі та неточні вимоги від замовника	0,6	0,6	0,36
4	Недостатній рівень знань розробника	0,2	0,6	0,12
5	Нераціональний розподіл часу	0,4	0,6	0,24
6	Часте редагування ТЗ	0,3	0,3	0,09
7	Створення незрозумілого користувачеві інтерфейсу	0,3	0,5	0,15
8	Некоректна робота одного із програмних модулів	0,2	0,4	0,08
9	Несправності технічного обладнання	0,1	0,7	0,07
10	Невірна оцінка масштабу проекту	0,3	0,4	0,12

Джерело: побудовано автором

Для того, щоб оцінити небезпеку потенційних буде використано матрицю ймовірності та ризику. Таке оцінювання дозволить розділити існуючі ризики на прийнятні, виправдані та недопустимі, що в свою чергу

дозволить розробити план дій на випадок виникнення того чи іншого ризику. Прийнятні ризики не завдадуть значної шкоди в масштабах проекту, тому їх можна просто ігнорувати. Виправдані ризики завдають помітну шкоду, але при додаткових стратегіях можна уникнути значних збитків. Недопустимі ризики притягнуть за собою надзвичайні втрати ресурсів, тому необхідно будь-якими шляхами знизити їх ймовірність, або зменшити вплив на проект. Розподілення ризиків за їх впливом та імовірністю показано у таблиці Б.4. Шкала оцінювання за ризиками представлена у таблиці Б.5. Заходи реагування на виявлені ризики в проекті наявні у таблиці Б.6.

Таблиця Б.4 – Розподілення ризиків за їх впливом і ймовірністю

Ймовірність	Вплив загрози(ризику)				
	Дуже малий 0,05	Малий 0,1	Середній 0,2	Великий 0,4	Дуже великий 0,8
0,9					
0,7				R3(0,36)	
0,5			R2(0,2) R1(0,25)		
0,3	R6(0,09)	R7(0,15) R12(0,12)	R5(0,24)		
0,1	R8(0,08) R9(0,07)	R4(0,12)			

Джерело: побудовано автором

Таблиця Б.5 – Шкала оцінювання за рівнем ризику

№	Назва	Межі	Ризики, які входять (номера)
1	Прийнятні	$0,005 \leq R \leq 0,05$	1, 4, 6, 7, 8, 9, 10
2	Виправдані	$0,05 < R \leq 0,14$	2, 3, 5
3	Недопустимі	$0,14 < R \leq 0,72$	

Джерело: побудовано автором

Таблиця Б.6 - Ризики та стратегії реагування

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
1	Відкритий	Неправильно сформоване ТЗ проекту	Висока	Середній	2	Визначити технічні аспекти та умови реалізації продукту, його цільове призначення	Ухилення	Погодити всі проблеми та шляхи їх вирішення з керівником
2	Відкритий	Непорозуміння між розробником та замовником	Середня	Середній	2	Налагодити гарні відносини, дотримуватися ділового етикету.	Попередження	Необхідно знайти та усунути причину непорозуміння
3	Відкритий	Нечіткі та неточні вимоги від замовника	Висока	Великий	3	Визначити всі необхідні параметри проекту, чіткі вимоги	Перенесення	Уточнити всі деталі у замовника, за необхідності внести правки
4	Відкритий	Недостатній рівень знань розробника	Низька	Середній	1	Обирати знайомі технології та засоби	Пом’якшення	Виділити час на підвищення кваліфікації робітників

Продовження табл. Б.6

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
5	Відкритий	Нераціональний розподіл часу	Середня	Середній	2	Чітке дотримання термінів, визначених на етапі планування	Зменшення	Виділення додаткового резерву часу, більш ретельне дотримання термінів
6	Відкритий	Часте редагування ТЗ	Низька	Низький	1	Дізнатися всі деталі та чіткі вимоги по кожній задачі	Зменшення	Дізнатися причину змін, та усунути її, замість наслідків
7	Відкритий	Створення незрозумілого користувачеві інтерфейсу	Низька	Середній	1	Створити звичний для типового користувача інтерфейс, додати пояснення	Ухилення	Зробити підказки в інтерфейсі або прибрати чи переробити функції, з якими виникають проблеми
8	Відкритий	Некоректна робота одного із програмних модулів	Низька	Низький	1	Провести детальне тестування	Ухилення	Пошук проблеми та внесенн змін для її видалення

Продовження табл. Б.6

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
9	Відкритий	Несправності технічного обладнання	Низька	Середній	1	Підготовка додаткової техніки, яка у разі поломки зможе замінити зламану	Перенесення	Здати несправне обладнання на ремонт
10	Відкритий	Невірна оцінка масштабу проекту	Низька	Низький	1	Провести детальний аналіз проекту, розумно розподілити час на їх виконання, залишивши невеликий резерв.	Пом’якшення	Повторна оцінка проекту, зміна плану

Джерело: побудовано автором

ДОДАТОК Б

Лістинг програмного коду основних модулів web-орієнтованої системи

MyServlet.java

```
package com.maxim328;

import …
@WebServlet(name = "servlet", urlPatterns = {"/servlet", "/jsp/servlet"})
public class MyServlet extends HttpServlet {
    @Override
    public void init(ServletConfig config) throws ServletException {
        //super.init(config);
        //Create connection pool
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        handleRequest(req, resp);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        handleRequest(req, resp);
    }

    private void handleRequest(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        if (req.getSession().getAttribute("userRole") == null) {
            Cookie[] cookies = req.getCookies();
            boolean isUserFoundInCookies = false;
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if (cookie.getName().equals("userID")) {
                        String username = cookie.getValue();
                        UserService userService = new UserServiceImpl();
                        req.getSession().setAttribute("userId", username);
                        User user = userService.getUserById(req);
                        req.getSession().setAttribute("userRole",
user.getRole());
                        isUserFoundInCookies = true;
                    }
                }
            }
            if (!isUserFoundInCookies)
req.getSession().setAttribute("userRole", 3);
        }
        if (req.getSession().getAttribute("lang") == null)
req.getSession().setAttribute("lang", new Locale("ua"));
        String commandTitle = req.getParameter(JSPAttributes.COMMAND);
        if (commandTitle == null) commandTitle =
CommandsList.HOME_PAGE.getCommandName();
        Command command = CommandsList.getCommandByTitle(commandTitle);
        Router router = command.execute(req, resp);
```

```
        if (router.getRouteType().equals(Router.RouteType.FORWARD))
            req.getRequestDispatcher(router.getRouteLink()).forward(req,
resp);
        else
            resp.sendRedirect(router.getRouteLink());
    }
}
```

CommandList.java

```
package com.maxim328.command;

import …

public enum CommandsList {
    HOME_PAGE(new HomePage(), "indexPage"),
    BOOK_ORDER_PAGE(new BookOrderPage(), "goToBookOrderPage"),
    ADMIN_USERS_ORDERS_PAGE(new AdminUsersPage(), "userOrdersPage"),
    PROFILE_PAGE(new ProfilePage(), "userProfile"),
    REGISTER_USER(new RegisterUserCommand(), "registerUser"),
    SIGN_IN(new SignInUserCommand(), "authorizeUser"),
    LOG_OUT(new LogOutCommand(), "logOut"),
    CREATE_ORDER(new CreateOrderCommand(), "createOrder"),
    USER_ORDERS(new UserOrdersPage(), "userOrders"),
    CANCEL_ORDER(new CancelOrderCommand(), "cancelOrder"),
    ACCEPT_ORDER(new AcceptOrderCommand(), "acceptOrder"),
    RETURN_ORDER(new ReturnOrderCommand(), "returnOrder"),
    RECHARGE_BALANCE(new RechargeBalanceCommand(), "rechargeBalance"),
    PAY_FINE(new PayOrderFineCommand(), "payFine"),
    NEW_ORDERS(new NewOrdersPage(), "newOrders"),
    CHANGE_LANG(new ChangeLangCommand(), "changeLang"),
    EDIT_CATALOGUE(new EditCatalogueCommand(), "catalogueEditor"),
    EDIT_BOOK(new EditBookPage(), "adminEditBook"),
    EDIT_BOOK_COMMAND(new EditBookCommand(), "editBookCommand"),
    ADD_BOOK_ADMIN_PAGE(new AddBookAdminPage(), "adminAddBook"),
    ADD_BOOK_ADMIN_COMMAND(new AddBookAdminCommand(), "addBookCommand"),
    DELETE_BOOK_ADMIN_COMMAND(new DeleteBookAdminCommand(),
"adminDeleteBook"),
    ADD_AUTHOR_ADMIN_PAGE(new addAuthorAdminPage(), "adminAddAuthorPage"),
    ADD_AUTHOR_ADMIN_COMMAND(new AddAuthorAdminCommand(),
"adminAddAuthorCommand");

    private static final Logger LOGGER =
LogManager.getLogger(CommandsList.class);

    private final Command command;
    private final String commandName;

    CommandsList(Command command, String commandName) {
        this.command = command;
        this.commandName = commandName;
    }

    public Command getCommand() {
        return command;
    }

    public String getCommandName() {
        return commandName;
    }

    public static Command getCommandByTitle(String commandTitle) {
```

```
        return getCommandEnumByTitle(commandTitle).command;
    }

    private static CommandsList getCommandEnumByTitle(String commandTitle) {
        for (CommandsList type: CommandsList.values()) {
            if (type.commandName.equals(commandTitle)){
                return type;
            }
        }
        LOGGER.warn(String.format("Command %s is not found, forward to
HomePage", commandTitle));
        return HOME_PAGE;
    }
}
```

HomePage.java

```
package com.maxim328.command.page;

import …

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.List;

public class HomePage implements Command {
    private static final BookService bookService =
ServiceFactory.getInstance().getBookService();

    @Override
    public Router execute(HttpServletRequest req, HttpServletResponse resp) {
        Router currentRouter = new Router();
        String param = req.getParameter("sort");
        String currentPage = req.getParameter("current_page");
        if (currentPage == null) currentPage = "1";
        if (param == null) param = "0";

        List<Book> allBooks = bookService.findAllBooksByPage(param,
Integer.parseInt(currentPage),
req.getSession().getAttribute("lang").toString());
        req.setAttribute(JSPAttributes.CURRENT_PAGE_BOOK_LIST, allBooks);
        req.getSession().setAttribute(JSPAttributes.CURRENT_PAGE_NUMBER,
currentPage);
        req.getSession().setAttribute(JSPAttributes.TOTAL_PAGES,
bookService.getTotalPages());

        currentRouter.setRouteLink(PageList.HOME);
        currentRouter.setRouteType(Router.RouteType.FORWARD);
        return currentRouter;
    }
}
```

BookService.java

```
package com.maxim328.service;

import com.maxim328.model.Book;
import com.maxim328.model.BookComponent;

import java.util.Date;
```

```
import java.util.List;

public interface BookService {
    List<Book> findAllBooks(String sort, String lang);
    List<Book> findAllBooksByPage(String sort, int currentPage, String lang);
    Book findBookByID(Integer id);
    int getTotalPages();
    void addNewBook(Book newBook);
    void createOrder(Integer bookId, Integer userId, Date
preferableReturnDate);
    List<BookComponent> getAllAuthors();
    List<BookComponent> getAllPublishers();
    List<BookComponent> getAllLanguages();
    List<BookComponent> getAllGenres();
    void editBook(Book updatedBook);
    void deleteBook(Integer bookId);
    void addNewAuthor(BookComponent author);
}
```

BookServiceImpl.java

```
package com.maxim328.service.serviceImpl;

import …

public class BookServiceImpl implements BookService {
    private final BookDAO bookDAO;
    private int totalPages;

    public BookServiceImpl() {
        bookDAO = DAOFactory.getInstance().getBookDAO();
        totalPages = 1;
    }

    @Override
    public List<Book> findAllBooks(String sort, String lang) {
        List<Book> result = bookDAO.findAllBooks();
        result.sort(getComparator(sort, lang));
        totalPages = result.size()/8 + result.size()%8 > 0 ? 1 : 0;
        return result;
    }

    @Override
    public List<Book> findAllBooksByPage(String sort, int currentPage, String
lang) {
        List<Book> result = bookDAO.findAllBooks();
        result.sort(getComparator(sort, lang));
        totalPages = result.size()/8;
        if (result.size()%8 > 0) totalPages+=1;
        return result.subList((currentPage-1)*8, Math.min(currentPage * 8,
result.size()));
    }

    @Override
    public Book findBookByID(Integer id) {
        return bookDAO.findBookById(id);
    }

    @Override
    public void createOrder(Integer bookId, Integer userId, Date
preferableReturnDate) {
        bookDAO.createOrder(bookId, userId, preferableReturnDate);
```

```
    }

    private Comparator<Book> getComparator(String sort, String lang) {
        switch (sort) {
            case "1":
                if (lang.equals("ua")) return
Comparator.comparing(Book::getTitleUa).reversed();
                else return
Comparator.comparing(Book::getTitleEn).reversed();
            case "2":
                if (lang.equals("ua")) return (o1, o2) ->
o2.getAuthor().getValue_ua().compareTo(o1.getAuthor().getValue_ua());
                else return (o1, o2) ->
o2.getAuthor().getValue_ua().compareTo(o1.getAuthor().getValue_ua());
            case "3":
                if (lang.equals("ua"))  return Comparator.comparing(o ->
o.getAuthor().getValue_ua());
                else return Comparator.comparing(o ->
o.getAuthor().getValue_ua());
            case "4":
                if (lang.equals("ua"))  return (o1, o2) ->
o2.getPublisher().getValue_ua().compareTo(o1.getPublisher().getValue_ua());
                else return (o1, o2) ->
o2.getPublisher().getValue_en().compareTo(o1.getPublisher().getValue_en());
            case "5":
                if (lang.equals("ua"))  return Comparator.comparing(o ->
o.getPublisher().getValue_ua());
                else return Comparator.comparing(o ->
o.getPublisher().getValue_en());
            case "6":
                return
Comparator.comparing(Book::getYearOfPublication).reversed();
            case "7":
                return Comparator.comparing(Book::getYearOfPublication);
            default:
                if (lang.equals("ua")) return
Comparator.comparing(Book::getTitleUa);
                else return Comparator.comparing(Book::getTitleEn);
        }
    }

    public int getTotalPages() {
        return totalPages;
    }

    @Override
    public List<BookComponent> getAllAuthors() {
        return bookDAO.getAllAuthors();
    }

    @Override
    public List<BookComponent> getAllPublishers() {
        return bookDAO.getAllPublishers();
    }

    @Override
    public List<BookComponent> getAllLanguages() {
        return bookDAO.getAllLanguages();
    }

    @Override
    public List<BookComponent> getAllGenres() {
        return bookDAO.getAllGenres();
```

```
    }

    @Override
    public void editBook(Book updatedBook) {
        bookDAO.editBook(updatedBook);
    }

    @Override
    public void addNewBook(Book newBook) {
        bookDAO.addNewBook(newBook);
    }

    @Override
    public void deleteBook(Integer bookId) {
        bookDAO.deleteBook(bookId);
    }

    @Override
    public void addNewAuthor(BookComponent author) {
        bookDAO.addNewAuthor(author);
    }
}
```

BookDAO.java

```
package com.maxim328.dao;

import com.maxim328.model.Book;
import com.maxim328.model.BookComponent;

import java.util.Date;
import java.util.List;

public interface BookDAO {
    Book findBookById(Integer bookId);
    List<Book> findAllBooks();
    void createOrder(Integer bookId, Integer userId, Date
preferableReturnDate);
    List<BookComponent> getAllAuthors();
    List<BookComponent> getAllPublishers();
    List<BookComponent> getAllLanguages();
    List<BookComponent> getAllGenres();
    void editBook(Book updatedBook);
    void addNewBook(Book book);
    void deleteBook(Integer bookId);
    void addNewAuthor(BookComponent author);
}
```

BookDAOImpl.java

```
package com.maxim328.dao.DAOImpl;

import com.maxim328.dao.BaseDAO;
import com.maxim328.dao.BookDAO;
import com.maxim328.dao.DAOConstants;
import com.maxim328.model.Book;
import com.maxim328.model.BookComponent;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
```

```
import java.sql.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class BookDAOImpl extends BaseDAO implements BookDAO {
    @Override
    public Book findBookById(Integer bookId) {
        Book bookResult = null;
        ResultSet resultSet;
        try(Connection con = pool.getConnection();
                PreparedStatement ps = con.prepareStatement(
                        DAOConstants.FIND_ALL_BOOKS +
DAOConstants.BY_BOOK_ID)) {
            ps.setInt(1, bookId);
            resultSet = ps.executeQuery();
            resultSet.next();
            bookResult = constructBookFromResult(resultSet);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return bookResult;
    }

    @Override
    public List<Book> findAllBooks() {
        List<Book> bookList = null;
        ResultSet resultSet = null;
        try (Connection con = pool.getConnection()) {
            Statement statement = con.createStatement();
            resultSet = statement.executeQuery(DAOConstants.FIND_ALL_BOOKS);
            bookList = new ArrayList<>();
            while (resultSet.next()) {
                bookList.add(constructBookFromResult(resultSet));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeResultSet(resultSet);
        }
        return bookList;
    }

    @Override
    public void createOrder(Integer bookId, Integer userId, Date
preferableReturnDate) {
        Connection con = null;
        PreparedStatement createOrder = null;
        PreparedStatement changeNumOfBooks = null;
        try {
            con = pool.getConnection();
            createOrder = con.prepareStatement(DAOConstants.CREATE_ORDER);
            changeNumOfBooks =
con.prepareStatement(DAOConstants.UPDATE_BOOK_QUANTITY);
            connectionSetAutoCommit(con, false);
            createOrder.setInt(1, userId);
            createOrder.setInt(2, bookId);
            createOrder.setDate(3, new
java.sql.Date(preferableReturnDate.getTime()));
            createOrder.executeUpdate();

            changeNumOfBooks.setInt(1, bookId);
            changeNumOfBooks.executeUpdate();
```

```
            connectionCommitChanges(con);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
            connectionsRollback(con);
            closeConnection(con);
        }
    }

    @Override
    public List<BookComponent> getAllAuthors() {
        List<BookComponent> authorsList = null;
        ResultSet resultSet = null;
        try (Connection con = pool.getConnection()) {
            Statement statement = con.createStatement();
            resultSet =
statement.executeQuery(DAOConstants.FIND_ALL_AUTHORS);
            authorsList = new ArrayList<>();
            while (resultSet.next()) {
                authorsList.add(new BookComponent(resultSet.getInt("id"),
                        resultSet.getString("name_EN"),
                        resultSet.getString("name_UA")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeResultSet(resultSet);
        }
        return authorsList;
    }


    @Override
    public List<BookComponent> getAllPublishers() {
        List<BookComponent> publishersList = null;
        ResultSet resultSet = null;
        try (Connection con = pool.getConnection()) {
            Statement statement = con.createStatement();
            resultSet =
statement.executeQuery(DAOConstants.FIND_ALL_PUBLISHERS);
            publishersList = new ArrayList<>();
            while (resultSet.next()) {
                publishersList.add(new BookComponent(resultSet.getInt("id"),
                        resultSet.getString("publisher_EN"),
                        resultSet.getString("publisher_UA")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeResultSet(resultSet);
        }
        return publishersList;
    }

    @Override
    public List<BookComponent> getAllLanguages() {
        List<BookComponent> langugesList = null;
        ResultSet resultSet = null;
        try (Connection con = pool.getConnection()) {
            Statement statement = con.createStatement();
            resultSet =
statement.executeQuery(DAOConstants.FIND_ALL_LANGUAGES);
            langugesList = new ArrayList<>();
            while (resultSet.next()) {
```

```
                langugesList.add(new BookComponent(resultSet.getInt("id"),
                        resultSet.getString("language_EN"),
                        resultSet.getString("language_UA")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeResultSet(resultSet);
        }
        return langugesList;
    }

    @Override
    public List<BookComponent> getAllGenres() {
        List<BookComponent> genresList = null;
        ResultSet resultSet = null;
        try (Connection con = pool.getConnection()) {
            Statement statement = con.createStatement();
            resultSet = statement.executeQuery(DAOConstants.FIND_ALL_GENRES);
            genresList = new ArrayList<>();
            while (resultSet.next()) {
                genresList.add(new BookComponent(resultSet.getInt("id"),
                        resultSet.getString("genre_EN"),
                        resultSet.getString("genre_UA")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeResultSet(resultSet);
        }
        return genresList;
    }

    @Override
    public void editBook(Book updatedBook) {
        Connection con = null;
        PreparedStatement updateBook = null;
        try {
            con = pool.getConnection();
            updateBook = con.prepareStatement(DAOConstants.EDIT_BOOK);
            connectionSetAutoCommit(con, false);
            updateBook.setInt(1, updatedBook.getAuthor().getId());
            updateBook.setInt(2, updatedBook.getLanguage().getId());
            updateBook.setInt(3, updatedBook.getPublisher().getId());
            updateBook.setInt(4, updatedBook.getGenre().getId());
            updateBook.setString(5, new
String(updatedBook.getTitleEn().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            updateBook.setString(6, new
String(updatedBook.getTitleUa().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            updateBook.setString(7, new
String(updatedBook.getDescriptionEn().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            updateBook.setString(8, new
String(updatedBook.getDescriptionUa().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            updateBook.setInt(9, updatedBook.getYearOfPublication());
            updateBook.setInt(10, updatedBook.getPages());
            updateBook.setInt(11, updatedBook.getTotalQuantity());
            updateBook.setInt(12, updatedBook.getAvailiableQuantity());
            updateBook.setString(13, updatedBook.getImage());
            updateBook.setInt(14, updatedBook.getId());
```

```
            updateBook.executeUpdate();

            connectionCommitChanges(con);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
            connectionsRollback(con);
            closeConnection(con);
        }
    }

    @Override
    public void addNewBook(Book book) {
        Connection con = null;
        PreparedStatement createBook = null;
        try {
            con = pool.getConnection();
            createBook = con.prepareStatement(DAOConstants.CREATE_BOOK);
            connectionSetAutoCommit(con, false);
            createBook.setInt(1, book.getAuthor().getId());
            createBook.setInt(2, book.getLanguage().getId());
            createBook.setInt(3, book.getPublisher().getId());
            createBook.setInt(4, book.getGenre().getId());
            createBook.setString(5, new
String(book.getTitleEn().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            createBook.setString(6, new
String(book.getTitleUa().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            createBook.setString(7, new
String(book.getDescriptionEn().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            createBook.setString(8, new
String(book.getDescriptionUa().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            createBook.setInt(9, book.getYearOfPublication());
            createBook.setInt(10, book.getPages());
            createBook.setInt(11, book.getTotalQuantity());
            createBook.setInt(12, book.getAvailiableQuantity());
            createBook.setString(13, book.getImage());
            createBook.executeUpdate();

            connectionCommitChanges(con);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
            connectionsRollback(con);
            closeConnection(con);
        }
    }

    @Override
    public void deleteBook(Integer bookId) {
        Connection con = null;
        PreparedStatement deleteBook = null;
        try {
            con = pool.getConnection();
            deleteBook = con.prepareStatement(DAOConstants.DELETE_BOOK);
            connectionSetAutoCommit(con, false);
            deleteBook.setInt(1, bookId);
            deleteBook.executeUpdate();

            connectionCommitChanges(con);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
```

```
            connectionsRollback(con);
            closeConnection(con);
        }
    }

    @Override
    public void addNewAuthor(BookComponent author) {
        Connection con = null;
        PreparedStatement addAuthor = null;
        try {
            con = pool.getConnection();
            addAuthor = con.prepareStatement(DAOConstants.CREATE_AUTHOR);
            connectionSetAutoCommit(con, false);
            addAuthor.setString(1, new
String(author.getValue_en().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            addAuthor.setString(2, new
String(author.getValue_ua().getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8));
            addAuthor.executeUpdate();

            connectionCommitChanges(con);
        } catch (SQLException | IOException | ClassNotFoundException e) {
            e.printStackTrace();
            connectionsRollback(con);
            closeConnection(con);
        }
    }
}
```

index.jsp

```
<%@page contentType="text/html; charset=UTF-8" isELIgnored="false" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:setLocale value="${lang}" />
<fmt:setBundle basename="localization" var="locale"/>
<html lang="ua">
<head>
    <link
href="https://fonts.googleapis.com/css?family=Montserrat:900|Roboto:300&displ
ay=swap&subset=cyrillic" rel="stylesheet"/>
    <link rel="stylesheet" type="text/css" href="../css/catalogue.css">
    <script type="text/javascript" src="../js/sendToServlet.js"></script>
    <script type="text/javascript" src="../js/selectOrderItem.js"></script>
    <title>Library</title>
</head>
<jsp:include page="header.jsp">
    <jsp:param name="userRole" value="${currentUser.role}"/>
</jsp:include>
<div class="catalogue-outside">
    <div class="left-space"></div>
    <div class="catalogue">
        <select id="selectOrder">
            <option><fmt:message bundle="${locale}"
key="index.sorting.title_from_a"/></option>
            <option><fmt:message bundle="${locale}"
key="index.sorting.title_from_z"/></option>
            <option><fmt:message bundle="${locale}"
key="index.sorting.author_from_a"/></option>
            <option><fmt:message bundle="${locale}"
key="index.sorting.author_from_z"/></option>
```

```
            <option><fmt:message bundle="${locale}"
key="index.sorting.publisher_from_a"/></option>
            <option><fmt:message bundle="${locale}"
key="index.sorting.publisher_from_z"/></option>
            <option><fmt:message bundle="${locale}"
key="index.sorting.date_from_newest"/></option>
            <option><fmt:message bundle="${locale}"
key="index.sorting.date_from_oldest"/></option>
        </select>
        <script>setElement()</script>
        <button onclick="sendToServlet()"><fmt:message bundle="${locale}"
key="index.sort"/></button>
        <c:forEach var="book" items="${books_list}">
            <div class="catalogue-item">
                <table>
                    <tr>
                        <td>
                            <div class="book-image">
                                <img class="book-image-content"
src="${book.image}">
                            </div>
                        </td>
                        <td>
                            <div class="text-info">
                                <c:choose>
                                    <c:when test='${lang == "ua"}'>
                                        <div class="book-
title">${book.titleUa}<br></div>
                                        <div class="book-
author">${book.author.value_ua}<br></div>
                                        <div class="additional-info">
                                            <div class="info-
item">${book.genre.value_ua}</div>
                                            <div class="info-
item">${book.language.value_ua}</div>
                                            <div class="info-
item">${book.yearOfPublication}</div>
                                            <div class="info-
item">${book.availiableQuantity}/${book.totalQuantity}<br></div>
                                        </div>
                                        <div class="description">
                                            <c:out
value="${book.descriptionUa}"/>
                                        </div>
                                    </c:when>
                                    <c:otherwise>
                                        <div class="book-
title">${book.titleEn}<br></div>
                                        <div class="book-
author">${book.author.value_en}<br></div>
                                        <div class="additional-info">
                                            <div class="info-
item">${book.genre.value_en}</div>
                                            <div class="info-
item">${book.language.value_en}</div>
                                            <div class="info-
item">${book.yearOfPublication}</div>
                                            <div class="info-
item">${book.availiableQuantity}/${book.totalQuantity}<br></div>
                                        </div>
                                        <div class="description">
                                            <c:out
value="${book.descriptionEn}"/>
```

```
                                        </div>
                                    </c:otherwise>
                                </c:choose>
                                <div class="order-div">
                                    <form action="servlet" method="post">
                                        <input type="hidden" name="action"
value="goToBookOrderPage">
                                        <input type="hidden" name="bookID"
value="${book.id}"/>
                                        <c:if test="${userRole == 2}">
                                            <button
type="submit"><fmt:message bundle="${locale}" key="index.order"/></button>
                                        </c:if>
                                        <c:if test="${userRole == 3 ||
userRole == 1 || userRole == 0}">
                                            <button
type="submit"><fmt:message bundle="${locale}" key="index.detailed"/></button>
                                        </c:if>
                                    </form>
                                </div>
                            </div>
                        </td>
                    </tr>
                </table>
            </div>
        </c:forEach>
    </div>
</div>
<form action="servlet" method="get">
    <ul class="pagination">
        <c:forEach begin="1" end="${total_pages}" var="pageNumber">
            <li>
                <button type="submit" name="current_page"
value="${pageNumber}"
                        class="${pageNumber == page_number ? 'active' : ''}">
                        ${pageNumber}
                </button>
            </li>
        </c:forEach>
    </ul>
</form>
<jsp:include page="footer.jsp"/>
```

catalogue.css

```
.catalogue-outside {
    display: flex;
    flex-wrap: wrap;
    flex-direction: row;
}

.catalogue {
    width: 60%;
}

.left-space {
    width: 20%;
}

.catalogue-item {
    margin-top: 40px;
    display: flex;
```

```
    flex-wrap: wrap;
    flex-direction: row;
    background: aliceblue;
    padding: 10px;
}

.text-info {
    width: 85%;
    padding-left: 30px;
}

.order-div {
    width: 15%;
    display: flex;
    justify-content: left;
    align-items: start;
    padding-top: 10px;
}

.book-image {
    width: 250px;
    display: flex;
    justify-content: center;
    align-content: center;
    align-items: center;
}

.book-image-content {
    max-height: 250px;
}

.order-button {
    text-align: center;
    font-size: 16px;
    padding-left: 10px;
    padding-right: 10px;
}

.book-title {
    font-size: 32px;
    text-align: center;
    font-weight: bold;
    margin-bottom: 5px;
    font-family: 'Montserrat', sans-serif;
}

.book-author {
    font-size: 26px;
    margin-bottom: 5px;
    text-align: center;
}

.additional-info {
    display: flex;
    flex-wrap: wrap;
    flex-direction: row;
}

.info-item {
    width: fit-content;
    margin-right: 30px;
    margin-bottom: 5px;
    font-style: italic;
```

```
}

.description {
    font-size: 20px;
}

.body {
     font-family: Arial, sans-serif;
 }

.pagination {
    display: flex;
    list-style: none;
    padding: 0;
    justify-content: center;
    margin-top: 20px;
}

.pagination li {
    margin: 0 5px;
}

.pagination a {
    text-decoration: none;
    padding: 8px 12px;
    border: 1px solid #ddd;
    border-radius: 4px;
    color: #333;
    transition: background-color 0.3s;
}

.pagination a:hover {
    background-color: #ddd;
}

.pagination .active {
    background-color: #007BFF;
    color: #fff;
    cursor: default;
}
```

Book.java

```
package com.maxim328.model;

import com.maxim328.buider.BookBuilder;

import java.io.Serializable;

public class Book implements Serializable {
    private final Integer id;
    private final BookComponent author;
    private final BookComponent genre;
    private final BookComponent publisher;
    private final BookComponent language;
    private final String title_en;
    private final String title_ua;
    private final String description_en;
    private final String description_ua;
    private final Integer yearOfPublication;
    private final Integer pages;
    private final Integer totalQuantity;
```

```
private final Integer availiableQuantity;
private final String image;

public Book(BookBuilder builder) {
    this.id = builder.getId();
    this.genre = builder.getGenre();
    this.language = builder.getLanguage();
    this.publisher = builder.getPublisher();
    this.author = builder.getAuthor();
    this.title_en = builder.getTitleEn();
    this.title_ua = builder.getTitleUa();
    this.yearOfPublication = builder.getYearOfPublication();
    this.pages = builder.getPages();
    this.description_en = builder.getDescriptionEn();
    this.description_ua = builder.getDescriptionUa();
    this.totalQuantity = builder.getTotalQuantity();
    this.availiableQuantity = builder.getAvailiableQuantity();
    this.image = builder.getImage();
}

public Integer getId() {
    return id;
}

public BookComponent getAuthor() {
    return author;
}

public BookComponent getGenre() {
    return genre;
}

public BookComponent getPublisher() {
    return publisher;
}

public BookComponent getLanguage() {
    return language;
}

public String getTitleEn() {
    return title_en;
}

public String getTitleUa() {
    return title_ua;
}

public String getDescriptionEn() {
    return description_en;
}

public String getDescriptionUa() {
    return description_ua;
}

public Integer getYearOfPublication() {
    return yearOfPublication;
}

public Integer getPages() {
    return pages;
}
```

```
    public Integer getTotalQuantity() {
        return totalQuantity;
    }

    public Integer getAvailiableQuantity() {
        return availiableQuantity;
    }

    public String getImage() {
        return image;
    }
}
```

Order.java

```
package com.maxim328.model;

import com.maxim328.buider.OrderBuilder;

import java.util.Date;

public class Order {
    private Integer orderId;
    private Integer userId;
    private Integer bookId;
    private String bookTitleEn;
    private String bookTitleUa;
    private String bookAuthorEn;
    private String bookAuthorUa;
    private Integer orderStatus;
    private Date returnDate;
    private Date realReturnDate;
    private Integer fine;
    private Integer finePaid;
    private String userName;
    private String userSurname;

    public Order(OrderBuilder orderBuilder) {
        orderId = orderBuilder.getOrderId();
        userId = orderBuilder.getUserId();
        bookId = orderBuilder.getBookId();
        bookTitleEn = orderBuilder.getBookTitleEn();
        bookTitleUa = orderBuilder.getBookTitleUa();
        bookAuthorEn = orderBuilder.getBookAuthorEn();
        bookAuthorUa = orderBuilder.getBookAuthorUa();
        orderStatus = orderBuilder.getOrderStatus();
        returnDate = orderBuilder.getReturnDate();
        realReturnDate = orderBuilder.getRealReturnDate();
        this.fine = orderBuilder.getFine();
        finePaid = orderBuilder.getFinePaid();
        userName = orderBuilder.getUserName();
        userSurname = orderBuilder.getUserSurname();
    }

    public Integer getOrderId() {
        return orderId;
    }

    public void setOrderId(Integer orderId) {
        this.orderId = orderId;
    }
```

```
public Integer getUserId() {
    return userId;
}

public void setUserId(Integer userId) {
    this.userId = userId;
}

public Integer getBookId() {
    return bookId;
}

public void setBookId(Integer bookId) {
    this.bookId = bookId;
}

public String getBookTitleEn() {
    return bookTitleEn;
}

public void setBookTitleEn(String bookTitleEn) {
    this.bookTitleEn = bookTitleEn;
}

public String getBookTitleUa() {
    return bookTitleUa;
}

public void setBookTitleUa(String bookTitleUa) {
    this.bookTitleUa = bookTitleUa;
}

public String getBookAuthorEn() {
    return bookAuthorEn;
}

public void setBookAuthorEn(String bookAuthorEn) {
    this.bookAuthorEn = bookAuthorEn;
}

public String getBookAuthorUa() {
    return bookAuthorUa;
}

public void setBookAuthorUa(String bookAuthorUa) {
    this.bookAuthorUa = bookAuthorUa;
}

public Integer getOrderStatus() {
    return orderStatus;
}

public void setOrderStatus(Integer orderStatus) {
    this.orderStatus = orderStatus;
}

public Date getReturnDate() {
    return returnDate;
}

public void setReturnDate(Date returnDate) {
    this.returnDate = returnDate;
```

```
    }

    public Date getRealReturnDate() {
        return realReturnDate;
    }

    public void setRealReturnDate(Date realReturnDate) {
        this.realReturnDate = realReturnDate;
    }

    public Integer getFinePaid() {
        return finePaid;
    }

    public void setFinePaid(Integer finePaid) {
        this.finePaid = finePaid;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserSurname() {
        return userSurname;
    }

    public Integer getFine() {
        return fine;
    }

    public void setFine(Integer fine) {
        this.fine = fine;
    }

    public void setUserSurname(String userSurname) {
        this.userSurname = userSurname;
    }
}
```

User.java

```
package com.maxim328.model;

import com.maxim328.buider.UserBuilder;

import java.util.Objects;

public class User {
    private Integer id;
    private Integer role;
    private String email;
    private String password;
    private String name;
    private String surname;
    private String phone;
    private Integer balance;

    public User() {
```

```
}

public User(UserBuilder builder) {
    id = builder.getId();
    role = builder.getRole();
    email = builder.getEmail();
    password = builder.getPassword();
    name = builder.getName();
    surname = builder.getSurname();
    phone = builder.getPhone();
    balance = builder.getBalance();
}

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public Integer getRole() {
    return role;
}

public void setRole(Integer role) {
    this.role = role;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getSurname() {
    return surname;
}

public void setSurname(String surname) {
    this.surname = surname;
}

public String getPhone() {
    return phone;
```

```
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public Integer getBalance() {
        return balance;
    }

    public void setBalance(Integer balance) {
        this.balance = balance;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id) && Objects.equals(role, user.role)
&& Objects.equals(email, user.email) && Objects.equals(password,
user.password) && Objects.equals(name, user.name) && Objects.equals(surname,
user.surname) && Objects.equals(phone, user.phone);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, role, email, password, name, surname, phone);
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", role=" + role +
                ", email='" + email + '\'' +
                ", password='" + password + '\'' +
                ", name='" + name + '\'' +
                ", surname='" + surname + '\'' +
                ", phone='" + phone + '\'' +
                '}';
    }
}
```

BookComponent.java

```
package com.maxim328.model;

import java.util.Objects;

public class BookComponent {
    private Integer id;
    private String value_en;
    private String value_ua;

    public BookComponent() {
    }

    public BookComponent(Integer id, String value_en, String value_ua) {
        this.id = id;
        this.value_en = value_en;
        this.value_ua = value_ua;
```

```
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getValue_en() {
        return value_en;
    }

    public void setValue_en(String value_en) {
        this.value_en = value_en;
    }

    public String getValue_ua() {
        return value_ua;
    }

    public void setValue_ua(String value_ua) {
        this.value_ua = value_ua;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        BookComponent that = (BookComponent) o;
        return Objects.equals(id, that.id) && Objects.equals(value_en,
that.value_en) && Objects.equals(value_ua, that.value_ua);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, value_en, value_ua);
    }

    @Override
    public String toString() {
        return "BookComponent{" +
                "id=" + id +
                ", value_en='" + value_en + '\'' +
                ", value_ua='" + value_ua + '\'' +
                '}';
    }
}
```