ДЕРЖАВНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ

УНІВЕРСИТЕТ

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка онлайн-магазину з продажу періодичних видань»

Студента 4 курсу, 9 групи,
спеціальності
122 «Комп'ютерні науки» _____ Троц Дмитро Петрович
підпис студента

Науковий керівник
кандидат фізико-математичних наук,
доцент _____ Самойленко Анна Тимофіївна
підпис керівника

Гарант освітньої програми
кандидат технічних наук, доцент _____ Демідов Павло Георгійович
підпис керівника

Київ 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«12» грудня 2022р.

Завдання
на випускну кваліфікаційну роботу (проект) студента

Троца Дмитра Петровича
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи (проекту)
«Розробка онлайн-магазину з продажу періодичних видань»
Затверджена наказом ректора від *«09 » грудня 2022 р. № 3332*

2. Строк здачі студентом закінченої роботи *30 травня 2023 року*

3. Цільова установка та вихідні дані до роботи
Мета роботи: обґрунтування та розробка інтернет-системи для продажу періодичних видань.
Об'єкт дослідження: процес створення WEB-додатків.
Предмет дослідження: технології створення Web-додатків.

4. Перелік графічного матеріалу_____
_____
_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1		15.12.2022 р.	15.12.2022 р.
2		15.12.2022 р.	15.12.2022 р.
3		15.12.2022 р.	15.12.2022 р.

6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. Аналітичне дослідження специфіки розробки Web-додатків

1.1 Теоретичний аналіз підходів до розробки WEB-застосунків

1.1.1 Традиційний підхід.

1.1.2 Підхід з використанням систем для керування вмістом сайту.

1.1.3 Low-code підхід

1.2 Архітектура WEB-застосунків відповідно поставлених задач

1.2.1 Моноліт

1.2.2 Мікросервіси

1.2.3 Серверлесс

1.3 Етапи створення WEB-додатків

РОЗДІЛ 2. Дослідження стеку технологій для розробки

2.1 Дослідження мов програмування та фреймворків

2.1.1 Java

2.1.2 JavaScript

2.1.3 Python

2.1.4 Ruby

2.1.5 PHP

2.2 Дослідження SQL та NoSQL БД

7. Календарний план виконання роботи

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*04.10.2022*	*04.10.2022*
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*15.12.2022*	*15.12.2022*
3	*Вступ*	*03.02.2023*	*03.02.2023*
4	*РОЗДІЛ 1. Аналітичне дослідження специфіки розробки Web-додатків*	*28.02.2023*	*28.02.2023*
5	*РОЗДІЛ 2. Дослідження стеку технологій для розробки*	*06.04.2023*	*06.04.2023*
6	*РОЗДІЛ 3. Розробка WEB-додатку*	*12.05.2023*	*12.05.2023*
7	*Висновки*	*15.05.2023*	*15.05.2023*
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*30.05.2023*	*30.05.2023*
9	*Попередній захист випускної кваліфікаційної роботи*	*31.05.2023-01.06.2023*	*31.05.2023-01.06.2023*
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*02.06.2023*	*02.06.2023*
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*05.06.2023*	*05.06.2023*
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «15» грудня 2022 р.

9. Керівник випускної кваліфікаційної роботи, доцент Самойленко Г. Т.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Демідов П.Г.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент-диплломник Троц Д. П.
(прізвище, ініціали, підпис)

12. Відгук керівника випускної кваліфікаційної роботи

Керівник випускної кваліфікаційної роботи

_____ 30.05.2023 р.

(підпис, дата)

13. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента Троца Д. П.

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____Демідов П.Г._____

Завідувач кафедри _____Пурський О.І.____

«_____»_____2023 р.

Анотація

У випускній кваліфікаційній роботі проведено розробку онлайн-магазину з продажу періодичних видань. Досліджено технологічні рішення в галузі розробки електронної комерції з метою створення ефективної та зручної платформи для користувачів. Основні результати роботи включають розробку та реалізацію веб-сайту, огляд і порівняння різних систем управління замовленнями, обробки платежів та управління запасами, що допомогли визначити найефективніші технологічні рішення для веб-сайту магазину періодичних видань. Дана робота сприяє поглибленню розуміння процесів розробки електронної комерції та надає практичні рекомендації для створення успішного та зручного веб-сайту для продажу періодичних видань.

Ключові слова: онлайн-магазин, періодичні видання, електронна комерція, технологічні рішення, веб-сайт.

Abstract

The graduation thesis presents the development of an online store for selling periodicals. Technological solutions in the field of e-commerce development were investigated to create an efficient and user-friendly platform. The main outcomes of this work include the development and implementation of a website, an overview and comparison of different systems for order management, payment processing, and inventory management, which helped identify the most effective technological solutions for the periodical store website. This work contributes to a deeper understanding of e-commerce development processes and provides practical recommendations for creating a successful and convenient website for selling periodicals.

Keywords: online store, periodicals, e-commerce, technological solutions, website.

ЗМІСТ

ВСТУП

В сучасному світі інтернет-магазини є одним з найбільш перспективних напрямків бізнесу. Це пов’язано з тим, що все більше людей використовують Інтернет для покупок. Одним з видів товарів, які можна продавати через інтернет-магазин, є періодичні видання.

Мета і завдання дослідження. Метою дослідження є обґрунтування та розробка інтернет-системи для продажу періодичних видань, визначення ключових етапів розробки та впровадження онлайн-магазину з продажу періодичних видань, виявлення основних проблем та їх рішення. Для досягнення поставленої мети необхідно було вирішити наступні **завдання:**

- дослідити основні вимоги до функціональності та дизайну онлайн-магазинів;
- провести аналітичне дослідження методів, підходів та найкращих практик для розробки Web-додатків;
- дослідити актуальні технологічні рішення для створення Web-додатків;
- визначити оптимальний набір технологій для розробки;
- розробити інтернет-магазин з продажу періодичних видань;
- створення конкретних рекомендацій щодо розробки та впровадження онлайн-магазину з продажу періодичних видань, які можуть бути використані в подальшій практичній діяльності.

Об’єкт дослідження: процес створення Web-додатків.

Предметом дипломної роботи є технології створення Web-додатків.

Практичне значення. Виконана робота має важливе значення для розвитку електронної комерції. Онлайн-магазини стають все більш популярними, і розробка такого магазину для продажу періодичних видань

може бути корисною для бізнесу, який прагне розширити свою онлайн-присутність та залучити більше клієнтів. Ці фактори зумовлюють також і **актуальність** випускної кваліфікаційної роботи.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел, 16 рисунків, 5 додатків і містить 48 сторінок основного тексту.

РОЗДІЛ 1.
АНАЛІТИЧНЕ ДОСЛІДЖЕННЯ СПЕЦИФІКИ РОЗРОБКИ WEB-ДОДАТКІВ

1.1 Теоретичний аналіз підходів до розробки WEB-застосунків

Першим, з чим потрібно визначитися при розробці WEB-проекту, крім ідеї та певних маркетингових аспектів, є платформа або засіб, за допомогою якого будуть створюватися всі компоненти сайту.

Багаторокова історія розвитку WEB-додатків дозволяє проаналізувати та обрати найкращий варіант для будь яких цілей. Існують декілька підходів до розробки додатків, кожен з яких має свої переваги та недоліки.

1.1.1 Традиційний підхід.

Підхід передбачає створення частин сайту з нуля.

Першою частиною є back-end – це програмно-апаратна частина, усе, що відбувається за лаштунками ресурсу (веб-сайту, веб-додатку чи програми), його серверна сторона. Головне завдання бекенду – зв’язати базу даних з фронтендом, який у свою чергу повинен відобразити дані у зручному для користувача вигляді. І навпаки, усе що відбувається на фронтенд-частині має надходити у базу-даних через бекенд. Backend-частина використовує такі мови програмування як: PHP, Ruby, Java, Perl, Python, Node.js, React.js та інші. Часто використовують фреймворки, що надають готові реалізації для певних етапів розробки. З найпопулярніших фреймворків можна виділити Spring Framework та Django.

Іншою частиною є front-end - це клієнтська сторона, що являє собою користувацький інтерфейс, тобто усе, що бачить та з чим взаємодіє користувач, коли браузер завантажує сторінку. До фронтенду відноситься: дизайн, верстка, наповнення, функціонал (кнопки, форми, віджети), які доступні користувачу. Frontend-частина здебільшого використовує такі мови веб-програмування як: HTML, CSS, JavaScript.

З переваг можна виділити повний контроль над логікою додатку, можливість реалізувати будь які зміни. Також безперечним плюсом є контроль над хостингом, що дозволяє мігрувати або масштабувати проект відповідно до потреб. При цьому налаштування безпеки проводиться доволі легко, наслідком чого є максимальна захищеність від спроб несанкціонованого доступу.

З недоліків – час на розробку додатку за таким підходом значно перевищує кількість часу, необхідного для розробки за іншими підходами. Також проект може мати велику кількість помилок, які будуть виявлені в ході довгого терміну користування додатком, всі вони можуть бути виправлені тільки розробником з необхідною кількістю знань та розумінням логіки сайту.

1.1.2 Підхід з використанням систем для керування вмістом сайту.

Система керування вмістом сайту (Content Management System) - це програмне забезпечення, яке допомагає користувачам створювати, керувати та змінювати вміст на веб-сайті без потреби у спеціальних технічних знаннях. Найпоширеніші варіанти таких систем – це Joomla!, Drupal, Wix, Shopify, Squarespace, WordPress.

Переваги CMS:

— можна створити сайт самостійно та за короткий проміжок часу;
— не потрібно розбиратися в дизайні та програмуванні;
— розробка сайту коштуватиме дешевше;
— зручно керувати вмістом сайту.

Недоліки CMS:

— слід стежити за оновленнями та сумісністю нових версій із доповненнями;
— швидкодія зазвичай знижується, якщо на сайті є багато доповнень;
— не весь функціонал вдасться реалізувати;
— не підходять для нетипових завдань.

1.1.3 Low-code підхід

Low-code — це підхід до розробки програмного забезпечення, який допомагає технічним спеціалістам і бізнес-користувачам співпрацювати та швидше створювати рішення для цифрової трансформації за рахунок мінімізації обсягу кодування. Точніше, low-code значною мірою замінює написання програмного коду використанням готових UI-компонентів, шаблонних скриптів та інтеграцій, типових схем, візуальних конструкторів та інших зручних для користувача функцій.

Найбільшою перевагою є зменшення часу створення додатку на 50-90% відповідно до дослідження компанії «451 Research» під назвою «Intelligent Process Automation and the Emergence of Digital Automation Platforms», що було проведене у лютому 2018 року. Крім того, кожна low code платформа містить у собі засоби керування безперебійною інтеграцією та доставкою продукту до користувача (CI/CD).

З основних недоліків - можливість блокування сервісу, що надає можливості для low-code розробки, натомість ваш сайт також стане назавжди недоступним. Кількість вбудованих можливостей на low code сервісі також обмежена, у зв'язку з чим виникає потреба у знанні кодингу, але навіть використання мови програмування не дасть повного доступу до налаштувань функціоналу.

Вибираючи підхід до веб-розробки, важливо враховувати конкретні вимоги та обмеження проекту. Кожен підхід має свої переваги та недоліки, і найкращий підхід залежатиме від потреб проекту. Важливо оцінити вимоги та обмеження, взяти до уваги досвід розробника та часові рамки, врахувати аспекти масштабованості, зручності обслуговування та безпеки програми, а потім вибрати засоби створення додатку, які найбільше задовольняють ці потреби. Проаналізувавши всі ці фактори, можна вибрати найкращий підхід для свого проекту.

1.2 Архітектура веб додатків відповідно поставлених задач

Архітектура веб-додатків значно змінилася за останні кілька років, регулярно з'являються нові підходи та технології. Одним із найважливіших рішень, яке мають прийняти розробники, є вибір архітектури, адже вона відіграє вирішальну роль у продуктивності додатку, масштабованості та зручності його обслуговування.

Трьома найпопулярнішими варіантами сьогодні є монолітна архітектура, архітектура мікросервісів і «serverless» архітектура. Кожна з них має свій набір переваг і недоліків, а вибір архітектури залежить від конкретних потреб і обмежень проекту.

1.2.1 Моноліт

Монолітна архітектура — це традиційний підхід до створення веб-додатків, коли всі компоненти додатка будуються як єдине ціле. Весь додаток працює як єдиний процес та використовує одну базу даних. Ця архітектура передбачає використання однієї кодової бази, де всі компоненти, такі як інтерфейс користувача, бізнес-логіка та налаштування доступу до бази даних, об'єднані в єдиний виконуваний файл. [1]

Однією з головних переваг монолітної архітектури є її простота. ЇЇ відносно легко зрозуміти та розробити, оскільки всі компоненти знаходяться в одному місці. Це також полегшує тестування та розгортання програми. Крім того, простіше підтримувати та оновлювати систему, оскільки всі компоненти знаходяться в одній кодовій базі. Це означає, що будь-які зміни в програмі потрібно вносити лише в одному місці, що зменшує ризик помилок.

Ще одна перевага монолітної архітектури полягає в тому, що її можна реалізувати за допомогою будь-якої мови програмування, на відміну від архітектури мікросервісів, яка може мати обмеження щодо мов, які вона підтримує.

Однак монолітна архітектура має і деякі недоліки. У міру того, як додаток зростає в розмірах і складності, ним стає все важче керувати та масштабувати його. Всю програму потрібно розгортати та тестувати разом, що може зайняти багато часу. Крім того, якщо будь-яку частину програми потрібно оновити або змінити, всю програму потрібно буде скомпілювати заново та повторно розгорнути. [1]

Загалом, монолітна архітектура є хорошим вибором для малих і середніх додатків, в яких не очікуються часті зміни. У міру зростання масштабу

розроблюваної програми може виникнути необхідність перенести її на іншу, мікросервісну або безсерверну, архітектуру, щоб забезпечити масштабованість і зручність обслуговування.

1.2.2 Мікросервіси

Мікросервісна архітектура — це спосіб побудови веб-додатків шляхом їх розбиття на невеликі незалежні сервіси, які можна розробляти та розгортати незалежно. Кожен мікросервіс відповідає за певні бізнес-можливості та спілкується з іншими мікросервісами через визначені інтерфейси. Це дозволяє різним командам працювати над різними мікросервісами одночасно, збільшуючи швидкість розробки. [2]

Простий приклад взаємодії знаходиться на рис. 1.1.

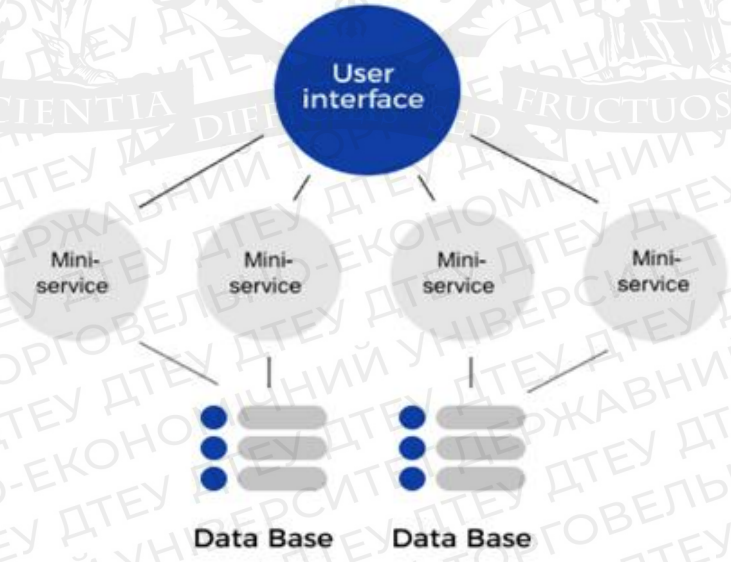


Рис. 1.1 Приклад роботи мікросервісів

Одна з головних переваг архітектури мікросервісів полягає в тому, що вона забезпечує масштабованість і гнучкість. Оскільки кожен мікросервіс є окремою одиницею, його можна масштабувати незалежно від інших

мікросервісів, що полегшує роботу зі збільшеним трафіком або навантаженням. Крім того, мікросервіси можуть бути написані різними мовами програмування та використовувати різні технології, що робить їх ще більш гнучким вибором для розробників. [2]

Ще одна перевага мікросервісів полягає в тому, що вони спрощують розгортання та тестування. Це можна робити з кожним мікросервісом незалежно, що зменшує ризик помилок. Крім того, якщо мікросервіс потрібно оновити або змінити, потрібно скомпілювати та повторно розгорнути лише цей мікросервіс, а не всю програму.

Крім того, якщо мікросервіс виходить з ладу, це не призводить до зупинки всієї програми, це вплине лише на певний модуль і залежні від нього мікросервіси. Це полегшує пошук і вирішення проблем.

Однак архітектура мікросервісів також має деякі недоліки. Вона є складнішою для розуміння та розробки, оскільки програма розбита на безліч невеликих незалежних компонентів. Крім того, зв’язок між мікросервісами може ускладнити систему та збільшити час створення відповіді для користувача.

Загалом, архітектура мікросервісів є хорошим вибором для великих, складних програм, які, як очікується, часто змінюватимуться та потребуватимуть масштабування. Це забезпечує швидшу розробку, більшу гнучкість і покращену масштабованість. Однак для цього потрібні чітко організовані процеси комунікації та взаємодії між командами.

1.2.3 Серверлесс

Безсерверна архітектура — це спосіб створення та запуску веб-додатків без серверів. Замість керування серверами розробники можуть створювати та

запускати свої програми на безсерверній платформі, наприклад AWS Lambda, Azure Functions або Google Cloud Functions. Ці платформи використовують модель оплати за використання, коли плата знімається лише за обчислювальні ресурси, які використовуються під час роботи програми.

Однією з головних переваг безсерверної архітектури є її економічна ефективність. Крім того, платформа піклується забезпечує користувачів засобами масштабування, виправлення та захисту інфраструктури, що теоретично може зменшити операційні витрати.

Ще одна перевага безсерверної архітектури полягає в тому, що вона може підвищити швидкість розробки. Розробники можуть зосередитися на написанні коду програми, не турбуючись про керування серверами.

Безсерверна архітектура також забезпечує більшу масштабованість. Платформа автоматично масштабує програму на основі кількості запитів. Це полегшує роботу зі збільшеним трафіком або навантаженням.

З недоліків безсерверної архітектури можна виділити збільшення часу очікування, оскільки потрібно запускати новий екземпляр програми при кожному запиті користувача. Крім того, може бути важче шукати та вирішувати помилки, оскільки додаток розподілений на кілька служб.

Загалом, безсерверна архітектура є гарним вибором для додатків, які мають змінний або непередбачуваний трафік, оскільки вона може заощадити гроші на витратах на інфраструктуру та підвищити швидкість розробки.

1.3 Етапи створення WEB-додатків

Першим етапом розробки веб-додатку є планування та розробка концепції. Цей етап має найвпливовіше значення на початковому етапі

створення проекту, адже робота з продуктом, що має чітку мету та напрямок завжди приводить до кращого результату.

На цьому етапі визначається обсяг і цілі проекту.

Обсяг визначає межі розробки та планований кінцевий результат.

Цілі — конкретний результат, який планується досягти розробкою продукту. Однією з цілей може бути підвищення популярності періодичних видань у конкретних регіонах.

Після визначення масштабів і цілей створюється приблизний план їх досягнення. Цей план має містити основні функції програми, технології, які плануються використовуватися, і приблизний графік виконання кожного завдання. Також повинні бути розглянуті будь-які потенційні перешкоди або проблеми, які можуть виникнути в процесі розробки, розроблені стратегії їх вирішення.

На цьому етапі також корисно провести дослідження подібних проектів, щоб виявити помилки в існуючих рішеннях та розробити можливі шляхи їх покращення.

Під час процесу розробки може знадобитися переглянути обсяг і цілі та внести необхідні зміни.

Другий етап у розробці веб-додатку — це етап проектування. Цей етап зосереджений на створенні вайрфреймів і макетів інтерфейсу користувача, а також на визначенні загального вигляду програми.

Вайрфрейми — це прості, низькоточні візуальні представлення макета та структури програми. Вони використовуються для планування розміщення таких елементів, як текст, зображення та кнопки, а також для створення загального розуміння того, як користувач буде взаємодіяти з програмою. Каркаси можна створювати за допомогою таких інструментів, як Adobe XD, Figma або Balsamiq, вони корисні для швидкого створення прототипів і

тестування різних дизайнерських ідей. [3]

У свою чергу макети — це високоточні візуальні представлення остаточного дизайну програми. Вони використовуються, щоб показати, як програма виглядатиме і зазвичай створюються за допомогою таких інструментів, як Adobe Photoshop, Sketch або Illustrator. [3]

Розробляючи проект, важливо пам’ятати про загальну естетику, колірну схему, типографіку та інші елементи дизайну, які використовуватимуться в програмі. Повинна бути розглянута цільова аудиторія програми, а кінцевий продукт повинен бути найбільш привабливим для неї.

Також важливо враховувати UX/UI дизайн програми. UI-дизайн являє собою графічну структуру програми. Він складається з кнопок, натиснених користувачами, текстів, які вони читають, зображень, полів введення тексту і всіх інших елементів, з якими взаємодіє користувач. Крім того, він включає в себе макет екрану, переходи, анімацію інтерфейсу і кожну мікро-взаємодію. Будь-який візуальний елемент, взаємодія або анімація повинні бути розроблені в процесі UI-дизайну. UX-дизайн — це процес розробки продуктів, які прості у використанні, ефективні та приємні у взаємодії. Обидва елементи мають вирішальне значення для створення позитивного досвіду користувача та забезпечення того, щоб програма була інтуїтивно зрозумілою та легкою для навігації.

Важливо протестувати ваш дизайн разом із користувачами та зібрати відгуки, щоб переконатися, що дизайн ефективний і відповідає потребам цільової аудиторії. Це можна зробити за допомогою тестування зручності використання.

Третій етап — етап розробки. Саме тут відбувається фактичне створення програми.

На цьому етапі використовуються технології та інструменти, які

визначились на етапі планування та розробки концепції.

Якщо для створення додатку був обраний традиційний підхід розробки «з нуля», то важливо пам’ятати про загальну архітектуру програми та дотримуватися найкращих практик щодо організації коду, коментування та документації. Це полегшить підтримку та оновлення коду в майбутньому.

Після написання коду потрібно перевірити програму, щоб переконатися, що вона функціонує належним чином і не має помилок. Це можна зробити за допомогою ручного або автоматизованого тестування з такими інструментами, як Selenium або Jest.

Також важливо враховувати безпеку програми. Це включає впровадження заходів безпеки для захисту від потенційних загроз.

Четвертий етап — етап розгортання. Тут програма налаштовується в середовищі хостингу та стає доступною для користувачів.

Існує кілька варіантів розміщення веб-програми, зокрема використання спільного постачальника послуг хостингу, віртуального приватного сервера (VPS) або хмарної служби хостингу, як-от AWS, Azure або Google Cloud Platform. Кожен варіант має свій набір плюсів і мінусів, і вибір залежатиме від конкретних вимог програми та ресурсів, доступних розробнику.

Після налаштування середовища хостингу потрібно налаштувати сервер. Це включає встановлення будь-якого необхідного програмного забезпечення, наприклад веб-сервера (наприклад, Apache або Nginx), бази даних (наприклад, MySQL або MongoDB) і будь-яких інших залежностей.

Після налаштування розгортається додаток. Зазвичай це робиться за допомогою контейнеризації усіх компонентів сайту та запуску цього контейнера на віддаленому сервері.

Після розгортання програми необхідно стежити за її безперебійною роботою та відсутністю проблем. Це може включати налаштування

інструментів моніторингу, таких як New Relic або Loggly, і регулярну перевірку журналів програми та звітів про помилки.

Важливо врахувати можливість збільшення трафіку або об'єму БД. Для того, щоб уникнути проблем, краще одразу обрати сервер, що буде працювати в умовах максимально очікуваного навантаження без перебоїв. Також можна використовувати балансувачі навантаження, кешування та інші методи оптимізації продуктивності.

П’ятий етап – етап обслуговування. Цей етап зосереджений на оновленні програми, виправленні будь-яких помилок і додаванні нових функціональних змін.

На цьому етапі важливо відстежувати будь-які помилки або проблеми, про які повідомляють користувачі і негайно вирішувати їх.

Також варто слідкувати за змінами у стандартних бібліотеках фреймворку або мови програмування, часто їх оновлення виправляють помилки у системі безпеки або дають змогу покращити швидкодію програми.

Варто зазначити, що всі етапи розробки додатку не мають однозначного порядку та термінів — це безперервні процеси, і потрібно постійно контролювати та оновлювати програму, щоб переконатися, що вона залишається безпечною, стабільною та продовжує відповідати потребам користувачів.

РОЗДІЛ 2.
ДОСЛІДЖЕННЯ СТЕКУ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

2.1 Дослідження мов програмування та фреймворків

2.1.1 Java

Java — це об’єктно-орієнтована мова програмування загального призначення на основі класів, яка спеціально розроблена для того, щоб мати якомога менше залежностей від реалізації [4]. Вона призначена для того, щоб дозволити розробникам програм написати код один раз і запустити будь-де (WORA), тобто скомпільований код Java може працювати на всіх платформах, які підтримують Java, без необхідності повторної компіляції. Java є однією з найпопулярніших мов програмування, особливо для клієнт-серверних веб-додатків.

Java має багатий набір бібліотек і фреймворків, що робить її популярним вибором для веб-розробки. Деякі популярні фреймворки веб-розробки Java включають:

— Spring Framework. Широко використовуваний фреймворк з відкритим вихідним кодом для створення програм корпоративного рівня. Він забезпечує комплексну модель програмування та налаштування для додатків на основі Java, що полегшує створення та підтримку великомасштабних веб-додатків. [5]

— Hibernate. Популярний фреймворк об’єктно-реляційного відображення (ORM) для Java. Це спрощує процес доступу та

маніпулювання даними, що зберігаються в реляційній базі даних, полегшуючи створення та підтримку веб-додатків, які покладаються на базу даних.

— Struts. Платформа Model-View-Controller (MVC) для створення веб-додатків. Він забезпечує гнучку та розширювану архітектуру для розробки веб-сервісів і широко використовується для створення корпоративних програм.

— JavaServer Faces (JSF). Платформа на основі компонентів на стороні сервера для створення веб-додатків. Вона надає набір багаторазово використовуваних компонентів інтерфейсу користувача та стандартизує процес створення веб-додатків, полегшуючи створення та підтримку програм, які відповідають шаблону MVC.

Java також надає ряд бібліотек для веб-розробки, таких як Servlet, JSP і JSTL, які широко використовуються для створення динамічних веб-сторінок. Сервлети Java — це невеликі програми Java, які працюють на веб-сервері та обробляють запити клієнтів. JavaServer Pages (JSP) подібні до сервлетів, але розроблені, щоб бути більш доступними для розробників, які знайомі з HTML і JavaScript. JSTL (стандартна бібліотека тегів JavaServer Pages) — це набір користувацьких тегів, які можна використовувати на сторінках JSP для виконання типових завдань, таких як ітерація та умовна логіка.

Java — це зріла, стабільна та широко використовувана мова програмування, яка добре підходить для створення великомасштабних веб-додатків корпоративного рівня. Незалежність Java від платформи, багатий набір бібліотек і фреймворків, а також широка спільнота розробників роблять її надійним вибором для багатьох типів проектів.

2.1.2 JavaScript

JavaScript — це мова програмування, яка в основному використовується для створення інтерактивних інтерфейсів веб-сайту, але її також можна використовувати для програмування на стороні сервера за допомогою фреймворків, таких як Node.js. Це інтерпретована, високорівнева, динамічна та багатопарадигмальна мова програмування, яка підтримується всіма сучасними веб-браузерами. JavaScript дозволяє розробникам створювати адаптивні та динамічні інтерфейси користувача, перевіряти введені користувачем дані та створювати анімацію та інші інтерактивні елементи на веб-сторінках. [6]

JavaScript в основному використовується для створення сценаріїв на стороні клієнта, але його також можна використовувати на стороні сервера за допомогою таких технологій, як Node.js. Це дозволяє розробникам використовувати JavaScript для повноцінної веб-розробки, що може бути корисним, оскільки забезпечує узгоджену мову для всього веб-додатку.

JavaScript має широкий спектр бібліотек і фреймворків, які використовуються для веб-розробки. Деякі популярні з них включають:

— React — це бібліотека JavaScript для створення інтерфейсів користувача. Він широко використовується для створення складних та високоінтерактивних веб-додатків, таких як односторінкові додатки (SPA). Це дозволяє розробникам створювати повторно використовувані компоненти інтерфейсу користувача та керувати станом програми ефективним і ефективним способом.

— Angular — це платформа JavaScript для створення веб-додатків. Він надає набір інструментів і функцій для створення складних великомасштабних веб-додатків. Angular слідує шаблону Model-View-Controller (MVC) і надає потужний набір директив і служб, які

дозволяють розробникам створювати веб-додатки з високим рівнем абстракції.

— Vue.js — це прогресивний фреймворк JavaScript для створення інтерфейсів користувача. Він дозволяє розробникам створювати веб-додатки з високим рівнем абстракції та надає потужний набір директив і служб, які спрощують створення повторно використовуваних компонентів інтерфейсу користувача.

— Express — це мінімальний і гнучкий фреймворк веб-додатків Node.js, який надає надійний набір функцій для веб- і мобільних додатків. Це дозволяє розробникам легко створювати серверну логіку та обробляти HTTP-запити та відповіді.

JavaScript є популярною та широко використовуваною мовою для веб-розробки, і її популярність продовжує зростати, оскільки все більше розробників використовують її для програмування на стороні сервера та використовують для повноцінної веб-розробки. Широкий вибір бібліотек і фреймворків, а також підтримка програмування як на стороні клієнта, так і на стороні сервера роблять його універсальним і потужним вибором для проектів веб-розробки.

2.1.3 Python

Python — це високорівнева інтерпретована мова програмування загального призначення, яка широко використовується для веб-розробки. Він має простий і легкий для вивчення синтаксис, що робить його чудовим вибором для новачків. Python пропонує широкий спектр бібліотек і фреймворків, які використовуються для веб-розробки, наприклад Django і Flask. [7]

Django — це високорівневий веб-фреймворк Python, який дозволяє розробникам швидко створювати веб-додатки з простим і прагматичним дизайном. Він відповідає архітектурному шаблону Model-View-Controller (MVC) і надає багато вбудованих функцій, таких як ORM, інтерфейс адміністратора та система автентифікації. Він також має велику й активну спільноту, що дозволяє легко знаходити підручники, посібники та інші ресурси.

Flask — це мікровеб-фреймворк, написаний мовою Python. Його класифікують як мікрофреймворк, оскільки він не потребує спеціальних інструментів чи бібліотек. У ньому немає рівня абстракції бази даних, перевірки форми або будь-яких інших компонентів, де вже існуючі бібліотеки сторонніх розробників забезпечують загальні функції. Flask прагне зберегти ядро простим, але розширюваним.

Python також використовується для сценаріїв на стороні сервера, що дозволяє розробникам використовувати Python для повноцінної веб-розробки. Це забезпечує узгоджену мову в усій веб-програмі.

Python також пропонує різноманітні бібліотеки веб-збирання, такі як BeautifulSoup, Scrapy, Selenium тощо, які можна використовувати для вилучення даних із веб-сайтів, а також використовується для створення ботів для збирання даних із різних веб-сайтів.

Популярність Python у веб-розробці зростає, оскільки все більше розробників використовують його для серверного програмування та використовують для повноцінної веб-розробки. Широкий вибір бібліотек і фреймворків, а також підтримка програмування як на стороні клієнта, так і на стороні сервера роблять його універсальним і потужним вибором для проектів веб-розробки.

2.1.4 Ruby

Ruby — це високорівнева інтерпретована мова програмування загального призначення, яка відома своєю простотою, читабельністю та виразністю. Він часто використовується для веб-розробки, і він має низку популярних веб-фреймворків, таких як Ruby on Rails (Rails), Sinatra та Hanami.

Ruby on Rails (Rails) — це веб-фреймворк, написаний мовою програмування Ruby. Його створено, щоб полегшити розробку веб-додатків за архітектурним шаблоном Model-View-Controller (MVC), і він надає багато вбудованих функцій, таких як ORM, інтерфейс адміністратора та система автентифікації. Rails наголошує на використанні розумних параметрів за замовчуванням, угодах над конфігурацією та принципі "Не повторюйся" (DRY). [8]

Sinatra — це легкий веб-фреймворк, написаний мовою Ruby, мінімалістичний, гнучкий і простий для розуміння. Sinatra не надає багато функцій, він підходить для невеликих веб-додатків, і його легко розширити за допомогою спеціальної логіки, його також дуже легко перевірити.

Hanami, подібно до Sinatra, є легкою веб-платформою для Ruby, вона розроблена, щоб бути простою, елегантною та легкою для вивчення, вона оптимізована для малих та середніх веб-додатків, а також має потужний інтерфейс командного рядка.

Ruby також використовується для сценаріїв на стороні сервера, що дозволяє розробникам використовувати Ruby для повноцінної веб-розробки. Це забезпечує узгоджену мову в усій веб-програмі.

Популярність Ruby у веб-розробці зростає, і вона має велику та активну спільноту, яка дозволяє легко знаходити підручники, посібники та інші ресурси. Його простота, читабельність і виразність роблять його чудовим

вибором для початківців, а широкий спектр бібліотек і фреймворків робить його універсальним і потужним вибором для проектів веб-розробки.

2.1.5 PHP

PHP (гіпертекстовий препроцесор) — це серверна мова сценаріїв, яка широко використовується для веб-розробки. Він особливо добре підходить для створення динамічних веб-сайтів і веб-додатків. PHP часто використовується в поєднанні з базою даних MySQL і веб-сервером Apache, які разом складають популярний стек LAMP (Linux, Apache, MySQL, PHP).

PHP має кілька популярних веб-фреймворків, таких як Laravel, Symfony, CodeIgniter і Yii.

Laravel — це фреймворк веб-додатків PHP із виразним елегантним синтаксисом. Він надає розширені інструменти, такі як маршрутизація, автентифікація, кешування тощо. Laravel розроблений для розробки веб-додатків за шаблоном Model-View-Controller (MVC), він має багато вбудованих функцій і активну спільноту, яка дозволяє легко знаходити підручники, посібники та інші ресурси. [9]

Symfony — це фреймворк веб-додатків, написаний мовою програмування PHP. Він розроблений, щоб допомогти розробникам створювати надійні веб-програми, які можна підтримувати та масштабувати. Symfony надає багато багаторазово використовуваних компонентів і бібліотек, а також має велику й активну спільноту, яка полегшує пошук підручників, посібників та інших ресурсів.

CodeIgniter — це легкий і простий у вивченні фреймворк веб-додатків для PHP, розроблений для малих і середніх веб-додатків і має невеликі розміри. CodeIgniter не вимагає від вас використання командного рядка, вам

не потрібно знати про простори імен і не вимагає від вас використання автозавантажувача.

Yii — це високопродуктивний фреймворк для веб-додатків, написаний на PHP. Він розроблений, щоб допомогти розробникам створювати надійні веб-програми, які можна підтримувати. Yii надає багато багаторазово використовуваних компонентів і бібліотек, а також має велику й активну спільноту, що полегшує знайти підручники, посібники та інші ресурси.

PHP використовується деякими з найбільших веб-сайтів у світі, включаючи Facebook, Wikipedia та WordPress. Це популярний вибір для веб-розробки завдяки простоті використання, широкому спектру бібліотек і фреймворків, а також здатності бездоганно працювати з базами даних і веб-серверами.

2.2 Дослідження SQL та NoSQL БД

2.2.1 SQL DB

SQL (Structured Query Language) — це мова програмування, яка використовується для керування реляційними базами даних і маніпулювання ними. Реляційні бази даних — це спосіб організації та зберігання даних у структурованому форматі, де дані організовані в таблиці, рядки та стовпці. SQL використовується для створення, модифікації та запитів до реляційних баз даних. [10]

Бази даних SQL зазвичай використовуються у веб-розробці для зберігання та отримання даних для веб-додатків. До популярних баз даних

SQL належать MySQL, PostgreSQL і Microsoft SQL Server. [10]

MySQL — це система керування реляційними базами даних (RDBMS) із відкритим кодом, яка широко використовується для веб-розробки. Вона відома своєю надійністю, стабільністю та простотою використання. MySQL підтримує широкий спектр типів даних, включаючи текст, числа та дати, і його можна використовувати з різними мовами програмування, включаючи PHP, Java та C#.

PostgreSQL — це потужна система керування реляційними базами даних із відкритим кодом. Він підтримує такі розширені функції, як багатоверсійний контроль паралельності, відновлення на певний момент часу та повнотекстовий пошук. PostgreSQL відомий своєю стабільністю, продуктивністю та масштабованістю, і його часто використовують для великомасштабних веб-додатків і сховищ даних. [11]

Microsoft SQL Server — це система керування реляційною базою даних (RDBMS), розроблена Microsoft. Він розроблений для використання в корпоративних середовищах і часто використовується для великомасштабних веб-додатків і сховищ даних. Microsoft SQL Server підтримує широкий спектр типів даних і може використовуватися з різними мовами програмування, включаючи .NET, C# і Java.

Бази даних SQL широко використовуються у веб-розробці завдяки їхній здатності зберігати та отримувати великі обсяги даних, а також здатності обробляти одночасний доступ кількох користувачів. Вони також відомі своєю надійністю, стабільністю та функціями безпеки.

2.2.2 NoSQL DB

Бази даних NoSQL (не тільки SQL) — це категорія нереляційних баз

даних, призначених для обробки великих обсягів неструктурованих або напівструктурованих даних. На відміну від традиційних баз даних SQL, які використовують фіксовану схему та покладаються на таблиці, рядки та стовпці для організації даних, бази даних NoSQL використовують гнучку схему та використовують різні моделі даних, такі як ключ-значення, документ, стовпці та графіки. [12]

Деякі популярні бази даних NoSQL включають MongoDB, Cassandra та Redis.

MongoDB — це база даних NoSQL на основі документів, популярна для веб- та мобільних додатків. Він зберігає дані у двійковому форматі JSON під назвою BSON і забезпечує швидке, гнучке надсилання запитів та індексування. MongoDB має високу масштабованість і може обробляти великі обсяги неструктурованих даних. [13]

Apache Cassandra — це база даних NoSQL, розроблена для високої доступності та масштабованості. Це сховище з широкими стовпцями, яке зберігає дані в стовпцях замість рядків. Cassandra відома своєю здатністю обробляти великі обсяги даних і підтримує високі навантаження на запис і читання.

Redis — це сховище ключ-значення в пам’яті, яке часто використовується як кеш або брокер повідомлень. Він відомий своєю високою продуктивністю та масштабованістю, а також підтримує широкий спектр типів даних, включаючи рядки, хеші, списки та набори.

Бази даних NoSQL стають все більш популярними у веб-розробці завдяки їхній здатності обробляти великі обсяги неструктурованих даних, їх масштабованості та здатності підтримувати швидкі, гнучкі запити. Вони особливо добре підходять для додатків, які передбачають обробку даних у реальному часі, аналітику великих даних і розподілених систем.

2.2.3 Порівняння SQL та NoSQL баз даних для веб-розробки

Порівнюючи бази даних SQL і NoSQL для веб-розробки, важливо враховувати характер і обсяг даних, тип програми та вимоги до масштабованості, продуктивності та узгодженості даних.

Бази даних SQL базуються на фіксованій схемі. Вони використовують SQL для створення, зміни та запиту даних. Бази даних SQL відомі своєю надійністю, стабільністю та функціями безпеки, і вони часто використовуються для програм, які включають складні зв’язки даних і транзакції. Вони також добре обробляють велику кількість транзакцій і забезпечують високий рівень цілісності даних.

З іншого боку, бази даних NoSQL є нереляційними і використовують різні моделі даних, такі як ключ-значення, документ, стовпчик і графік. Вони використовують гнучку схему, що означає, що структура даних може змінюватися з часом. Бази даних NoSQL відомі своєю здатністю обробляти великі обсяги неструктурованих або напівструктурованих даних, їх масштабованістю та здатністю підтримувати швидкі та гнучкі запити. [14] Вони особливо добре підходять для додатків, які передбачають обробку даних у реальному часі, аналітику великих даних і розподілених систем. Вони також відмінно справляються з високошвидкісними даними та підходять для зберігання даних із великою кількістю полів або для неструктурованих даних.

Підсумовуючи, обираючи базу даних для веб-розробки, важливо оцінити конкретні потреби проекту. Якщо дані мають складну структуру та зв’язки, бази даних SQL можуть бути кращим варіантом, а якщо дані неструктуровані, очікується дуже велика кількість записів – бази даних NoSQL є кращим вибором.

2.3 Порівняння технологічних рішень та вибір оптимального

Масштаб проекту напряму впливає на вибір архітектури для розробки. Розробка онлайн-магазину з продажу періодичних видань не є надто великим проектом в порівнянні з іншими комерційними роботами. Саме тому для свого веб-додатку я обрав монолітну архітектуру. Перед вибором монолітної архітектури я ретельно проаналізував можливість майбутнього розширення додатку та вирішив, що на даному етапі розвитку проекту монолітна архітектура є найкращим варіантом. Вона дозволяє швидко та ефективно вирішувати поточні задачі та запускати продукт на ринок. Крім того, монолітна архітектура є більш стабільною та менш складною для управління, що також є важливим аспектом для розробки невеликого проекту.

Сайт буде розроблятися з нуля, тобто обраний традиційних підхід, що дасть можливість ретельно вивчити кожен етап процесу розробки, виявити та вирішити всі проблеми, які можуть виникнути на шляху. Такий підхід дозволить розробити сайт максимально оптимальним, з точки зору якості та продуктивності.

Одним із головних інструментів, які будуть використовуватися при розробці, є Java та Spring Framework. Ці технології дозволяють розробляти високоякісний код, який буде ефективно виконуватися на стороні сервера, а також забезпечувати високу швидкість роботи сайту. Для забезпечення доступу до бази даних буде використовуватися MySQL, що дозволить зберігати та організовувати великі обсяги даних.

Hibernate та Spring Data JPA дозволять зменшити кількість коду, необхідного для роботи з базою даних, та забезпечити високу ефективність

запитів до БД. Додатково будуть підключені залежності Spring Security, що дозволить забезпечити захист сайту та його користувачів від потенційних загроз, а Spring MVC дозволить легко розробляти та управляти веб-інтерфейсом сайту.

У підсумку, використання Java та Spring Framework разом з MySQL та додатковими технологіями, які підтримують розробку веб-додатків, дозволить розробити масштабований, продуктивний та безпечний онлайн-магазин з продажу періодичних видань.

Для досягнення оптимального досвіду користувача було обрано HTML та CSS як основні мови для верстки та стилізації сторінок веб-додатку.

Щоб забезпечити гнучкий та динамічний підхід до роботи з даними, використовуватиметься шаблонізатор Thymeleaf. Він дозволяє додавати динамічний контент до HTML сторінок, що робить взаємодію з користувачем більш зручною та ефективною.

РОЗДІЛ 3.
ДОСЛІДЖЕННЯ СТЕКУ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

3.1 Створення архітектури БД

Архітектура SQL-бази даних (БД) визначає організацію та структуру даних у системі керування базами даних (СКБД) з використанням мови SQL (Structured Query Language). SQL-БД зазвичай складається з таблиць, які містять рядки (records) з даними, організованими у вигляді колонок (columns).

Одним з важливих аспектів архітектури SQL-БД є нормалізація. Нормалізація - це процес організації даних у БД з метою усунення аномалій та забезпечення цілісності даних. Вона допомагає уникнути дублювання даних та забезпечити ефективне використання простору для зберігання.

Нормалізація базується на ряді нормальних форм, таких як перша нормальна форма (1NF), друга нормальна форма (2NF), третя нормальна форма (3NF) та інші. Кожна нормальна форма має свої правила та критерії, що дозволяють розбити дані на окремі таблиці і залежності для досягнення оптимальної структури БД. [15]

Нормалізація допомагає уникнути аномалій оновлення, видалення та вставки даних, забезпечує консистентність та інтегритет даних. Це дозволяє зручно та ефективно виконувати операції пошуку, модифікації та звітування в БД.

Розроблена для проекту архітектура БД відповідає всім вимогам нормалізації та зображена на рис. 3.1.

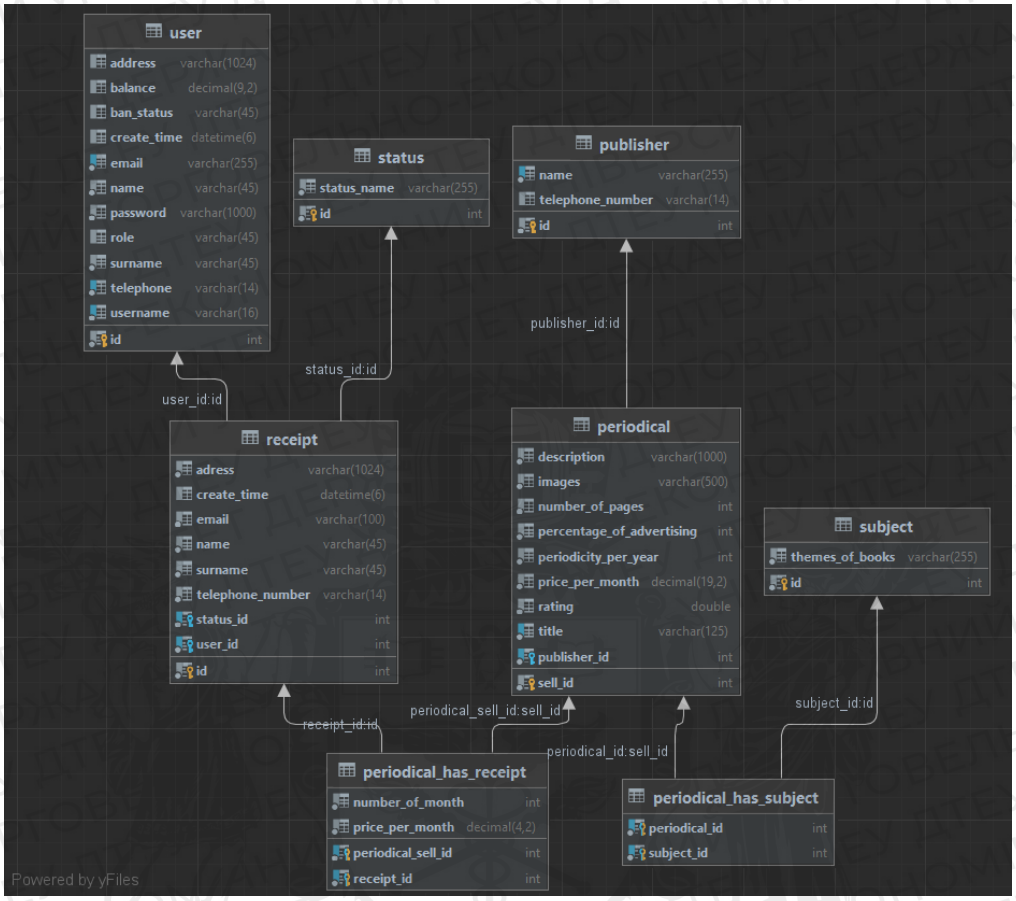


Рис. 3.1 Структура БД

У розробленій базі даних є вісім таблиць:

1. User – зберігає інформація про користувача, заповнюється при реєстрації.
2. Status – зберігає можливі статуси замовлення (оплачено, підтверджено, відхилено)
3. Publisher – інформація про видавця, заповнюється при додаванні видання.
4. Periodical – інформація про періодичне видання.
5. Receipt – інформація про замовлення.
6. Subject – тематика періодичного видання, заповнюється при додаванні нового періодичного видання.

7. Periodical_has_subject – інформація, що співставляє періодичне видання та тематику, створена через зв’язок many-to-many між таблицями Periodical і Subject.
8. Periodical_has_receipt – містить інформацію про замовлення та відповідне періодичне видання. Також зберігає дані про кількість місяців підписки та ціну видання за один місяць.

Загалом, архітектура SQL-БД забезпечує ефективне зберігання, керування та доступ до даних. Використання нормалізації допомагає створити оптимальну структуру БД, забезпечуючи цілісність даних та оптимальну продуктивність системи.

3.2 Розробка WEB-додатку «Periodicals»

Початковий етап розробки включає створення системи каталогів та підключення необхідних залежностей. Щоб спростити цей процес, можна скористатися сервісом Spring Initializr, який допомагає згенерувати початковий шаблон проекту.

Під час налаштування потрібно вказати параметри автоматичної збірки проекту, мову програмування, її версію, загальні імена проекту та залежності, необхідні для правильної роботи програми відразу після створення. На рисунку 3.2 показані базові налаштування проекту.

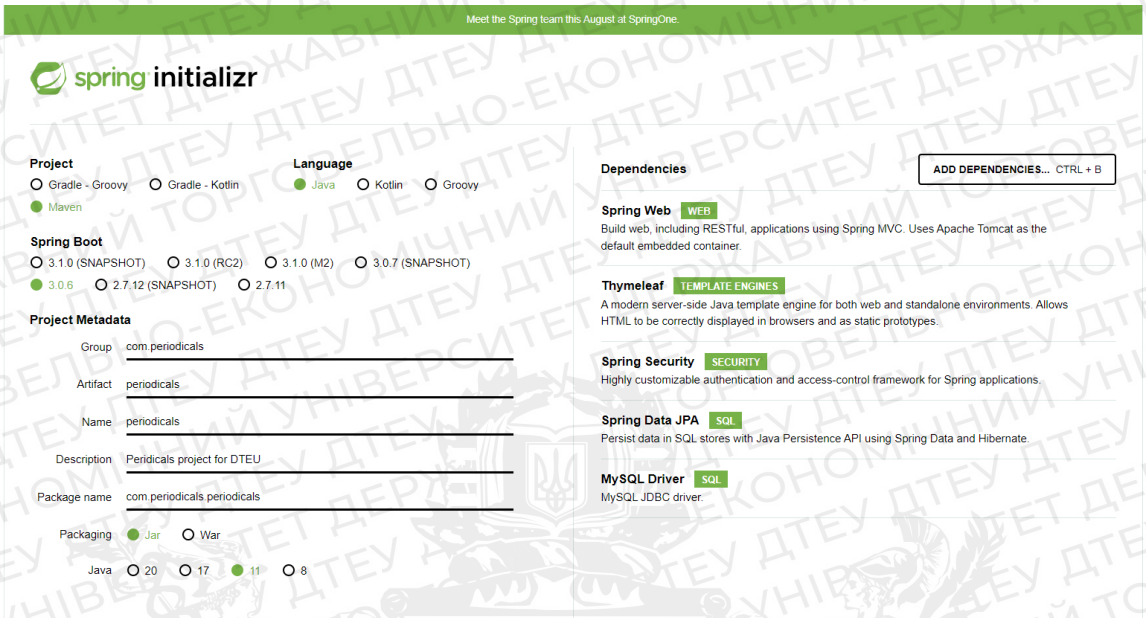


Рис. 3.2 Сторінка для створення шаблону проекту

Стандартна структура проекту, яка була згенерована, зображена на рис. 3.3

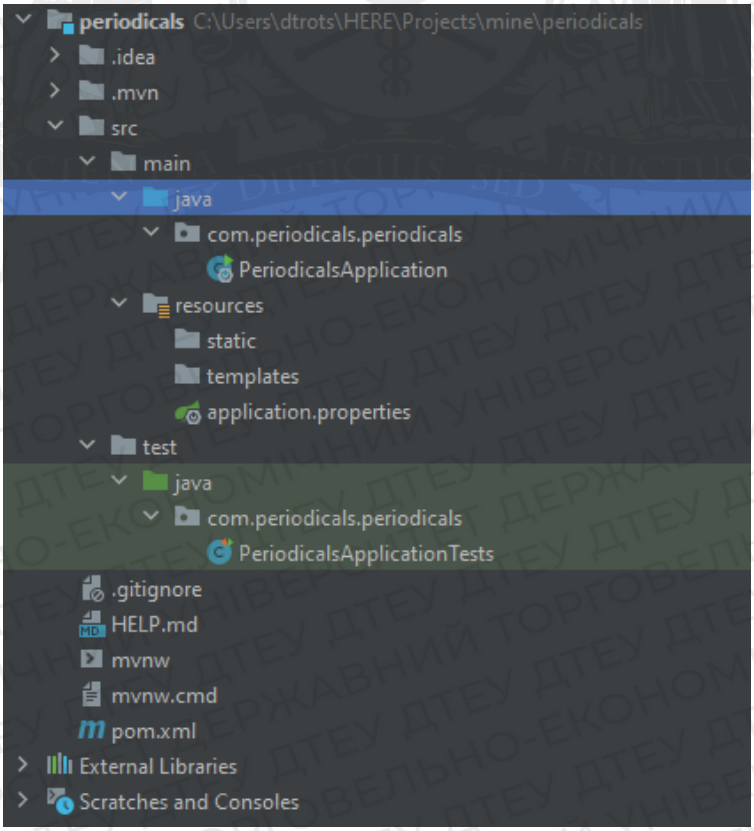


Рис. 3.3 Початкова структура проекту

При розробці додатка на основі патерну MVC, ми можемо створити та перейменувати пакети таким чином, щоб логічно розділити класи. Для цього створимо пакети controllers, services, repository, entity і utils.

У пакеті entity зберігаються класи, які відповідають таблицям у базі даних. Кожній таблиці в базі даних відповідає відповідний Java клас. Якщо у нашій програмі передбачена корзина для товарів, то її можна зберегти в сесії користувача. Крім того, якщо є таблиці зі зв'язком many-to-many, вони можуть мати представлення у вигляді двох класів. Один клас позначається анотацією @Embeddable, а другий має поле з анотацією @EmbeddedId.

Для зв'язку полів у класах, які представляють зв'язки в базі даних, використовуються анотації, такі як @ManyToMany, @OneToOne, @OneToMany, @ManyToOne, @MapsId, @JoinColumn та інші [16]. Розглянути структуру класів у папці entity можна за допомогою рис. 3.4.

```
26 usages  DmytroTrots
@Table(name = "publisher", indexes = {@Index(name = "name_UNIQUE", columnList = "name", unique = true)})
@Entity
public class PublisherEntity {
    4 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Integer id;

    3 usages
    @NotBlank(message = "Specify this field correctly")
    @Column(name = "name", nullable = false, unique = true)
    private String name;

    3 usages
    @Column(name = "telephone_number", length = 14)
    private String telephoneNumber;

    DmytroTrots
    public PublisherEntity(Integer id) { this.id = id; }

    2 usages  DmytroTrots
    public PublisherEntity() {
    }

    6 usages  DmytroTrots
    public PublisherEntity(Integer id, String name, String telephoneNumber) {
        this.id = id;
        this.name = name;
        this.telephoneNumber = telephoneNumber;
    }
```

Рис. 3.4 Структура класу PublisherEnity

У класах сутностей, назва таблиці з бази даних може бути вказана за допомогою анотації @Table над назвою класу. Також можна визначити параметри індексів, які полегшують пошук даних. Обов'язковою анотацією для класу-сутності є @Entity, яка дозволяє програмі розпізнати його призначення.

Поля класу відповідають атрибутам таблиці в базі даних. Звичайне поле позначається анотацією @Column, а поле, що відповідає унікальному ідентифікатору, додатково позначається анотацією @Id. В анотації @Column можна вказати назву атрибуту з бази даних та додаткові параметри поля для валідації.

Після цього, можна приступити до створення репозиторіїв. Такі класи дозволяють виконувати стандартні операції з базою даних, такі як збереження, оновлення, видалення та отримання записів, без необхідності писати складні SQL-запити вручну. Використання репозиторію спрощує розробку програмного забезпечення та забезпечує більшу модульність та переносимість коду.

Репозиторій зазвичай представлений інтерфейсом або класом, який визначає методи для взаємодії з даними. Можуть використовуватися анотації, такі як @Query, для декларування власних запитів до бази даних.

Приклад класу-репозиторію можна побачити на рис. 3.5.

```
5 usages  DmytroTrots
@Repository
public interface PeriodicalHasSubjectRepository extends JpaRepository<PeriodicalHasSubjectEntity, Integer> {

    3 usages  DmytroTrots
    @Query("select p from PeriodicalHasSubjectEntity p group by p.periodical.sellId")
    Page<PeriodicalHasSubjectEntity> findAllGroupBySellId(Pageable pageable);

    3 usages  DmytroTrots
    @Query("select p from PeriodicalHasSubjectEntity p where p.subject.themesOfBooks=:category")
    Page<PeriodicalHasSubjectEntity> findAllByCategory(Pageable pageable, @Param("category") String category);

    3 usages  DmytroTrots
    @Query("select p from PeriodicalHasSubjectEntity p where p.subject.themesOfBooks=:category and p.periodical.title like %:title%")
    Page<PeriodicalHasSubjectEntity> findAllByCategoryAAndPeriodicalTitle(Pageable pageable, @Param("category") String category, @Param("title") String title);

    3 usages  DmytroTrots
    @Query("select p from PeriodicalHasSubjectEntity p where p.periodical.title like %:title% group by p.periodical.sellId")
    Page<PeriodicalHasSubjectEntity> findAllByTitle(Pageable pageable, @Param("title") String title);
}
```

Рис. 3.5 Структура класу PeriodicalsHasSubjectRepository

Наступним етапом є створення сервісів.

Сервіси в Spring Boot відповідають за реалізацію бізнес-логіки та виконання операцій над даними. Сервіси знаходяться між контролерами, які приймають та обробляють запити, та репозиторіями, які забезпечують доступ до даних у базі даних.

Для створення сервісів потрібно створити клас, який представлятиме сервіс. Цей клас може бути позначений анотацією @Service.

Один з ключових аспектів сервісів – це ін'єкція залежностей. Сервіси можуть взаємодіяти з іншими компонентами, такими як репозиторії або інші сервіси. Це досягається за допомогою механізму ін'єкції залежностей Spring. Залежності можуть бути ін'єктовані через конструктор, за допомогою анотації @Autowired, або за допомогою методу-сеттера. [16]

Сервіси також реалізують методи, які виконують конкретні операції бізнес-логіки. Ці методи можуть викликати методи репозиторіїв для отримання, збереження або оновлення даних. Сервіси допомагають відокремити бізнес-логіку від контролерів та репозиторіїв.

Використання сервісів у Spring Boot дозволяє структурувати програму

відповідно до принципів GRASP (General Responsibility Assignment Software Patterns). Це допомагає покращити читабельність коду, облегшує тестування та підтримку програми у майбутньому. Крім того, сервіси можуть використовуватися для реалізації бізнес-правил, валідації даних, обробки виключень та інших аспектів, що стосуються логіки додатку.

У Spring Boot існує також можливість використання анотацій, таких як @Transactional, для позначення методів сервісу, які виконують операції з базою даних. Це дозволяє автоматично керувати транзакціями та забезпечувати консистентність даних у випадку виникнення помилок.

На рис. 3.6 можна побачити реалізацію класу-сервісу, де ін’єктується створений на попередньому етапі репозиторій через конструктор. Також присутній метод updatePeridical, котрий відповідає за оновлення уже існуючого видання, що зберігається в базі даних.

```
9 usages  DmytroTrots *
@Service
public class PeriodicalService {

    4 usages
    private final PeriodicalRepository periodicalRepository;

    DmytroTrots
    public PeriodicalService(PeriodicalRepository periodicalRepository) {
        this.periodicalRepository = periodicalRepository;
    }

    2 usages  DmytroTrots
    public boolean updatePeriodical(Integer periodicalId, PeriodicalEntity periodicalForm) {
        PeriodicalEntity periodical = periodicalRepository.getBySellId(periodicalId);
        periodical.setPricePerMonth(periodicalForm.getPricePerMonth());
        periodical.setPeriodicityPerYear(periodicalForm.getPeriodicityPerYear());
        periodical.setImages(periodicalForm.getImages());
        periodical.setPublisher(periodicalForm.getPublisher());
        periodical.setDescription(periodicalForm.getDescription());
        periodical.setTitle(periodicalForm.getTitle());
        periodical.setPercentageOfAdvertising(periodicalForm.getPercentageOfAdvertising());
        periodical.setNumberOfPages(periodicalForm.getNumberOfPages());
        periodical.setRating(periodicalForm.getRating());
        periodicalRepository.save(periodical);
        return true;
    }

    3 usages  DmytroTrots
    public PeriodicalEntity getPeriodicalByTitle(String title) { return periodicalRepository.getByTitle(title); }
}
```

Рис. 3.6 Структура класу PeriodicalsService

У Spring для роботи з користувачами та керування їх ролями та правами використовується спеціальний клас, який називається User. Цей клас надає зручний спосіб представлення користувача і містить ряд методів для отримання необхідної інформації про нього.

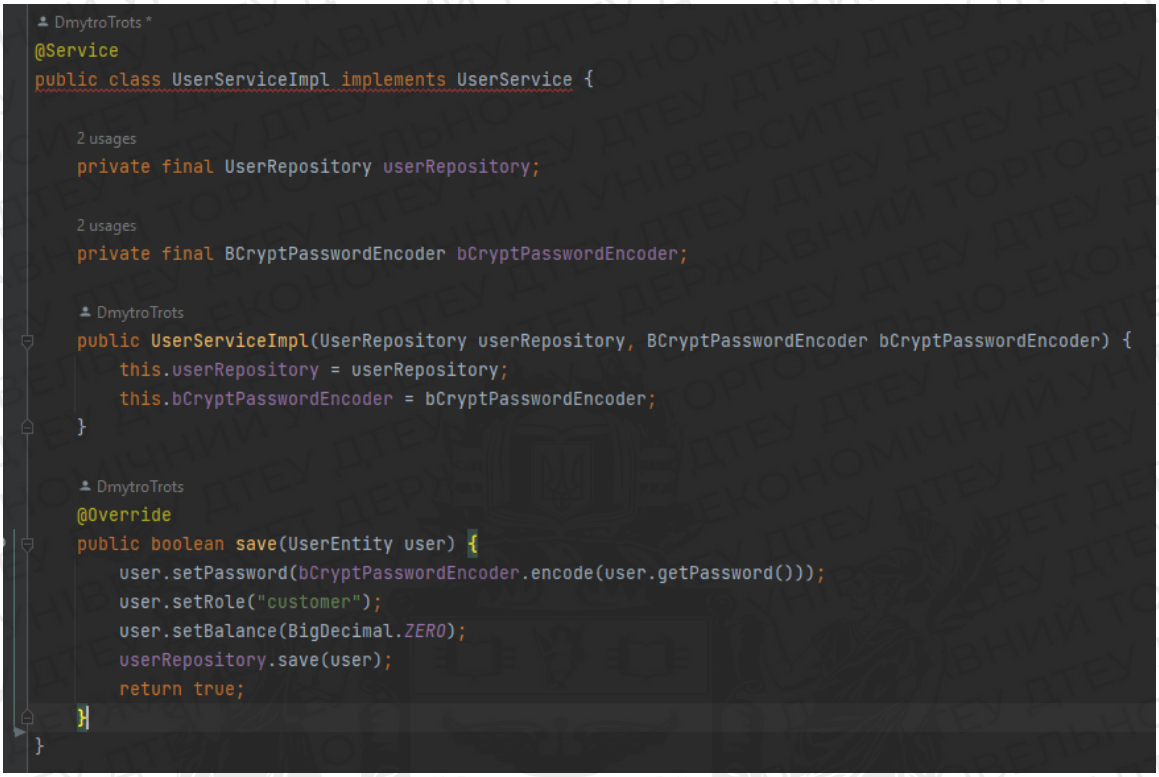
Наприклад, метод getPassword() дозволяє отримати пароль користувача.. Метод getUsername() повертає ім'я користувача. Ці методи використовуються для ідентифікації користувача в системі.

Деякі методи, такі як isAccountNonExpired() та isAccountNonLocked(), дозволяють перевірити статус облікового запису користувача. Наприклад, isAccountNonExpired() повертає значення true, якщо обліковий запис користувача не має закінченого терміну дії. Такі методи допомагають управляти статусом облікових записів та робити відповідні перевірки.

Крім цього, є метод eraseCredentials(), який очищає конфіденційну інформацію, таку як пароль, з об'єкту користувача. Це важливо для забезпечення безпеки та захисту конфіденційної інформації.

У класі User також є можливість визначення ролей користувача за допомогою методу roles(). Цей метод дозволяє встановити або отримати ролі, пов'язані з користувачем. Ролі використовуються для визначення прав доступу користувача до різних ресурсів системи.

Щодо збереження паролів, в Spring можна використовувати клас BCryptPasswordEncoder, який використовує сильний алгоритм хешування, що робить паролі надійними відносно атак злому. При цьому, сам пароль ніколи не зберігається у відкритому вигляді, а лише його хеш-представлення. Коли користувач авторизується, введений пароль порівнюється з хешем, збереженим у базі даних. Приклад використання класу BCryptPasswordEncoder можна побачити на рис. 3.7.



```
DmytroTrots *
@Service
public class UserServiceImpl implements UserService {

    2 usages
    private final UserRepository userRepository;

    2 usages
    private final BCryptPasswordEncoder bCryptPasswordEncoder;

    DmytroTrots
    public UserServiceImpl(UserRepository userRepository, BCryptPasswordEncoder bCryptPasswordEncoder) {
        this.userRepository = userRepository;
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;
    }

    DmytroTrots
    @Override
    public boolean save(UserEntity user) {
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        user.setRole("customer");
        user.setBalance(BigDecimal.ZERO);
        userRepository.save(user);
        return true;
    }
}
```

Рис. 3.7 Використання BCryptPasswordEncoder

Після цього потрібно створити контролери. У Spring Framework контролери є важливою складовою для обробки HTTP-запитів і керування потоком даних у додатку. Контролери відповідають за приймання вхідних запитів, обробку їх та повернення відповідей клієнту.

У Spring контролери визначаються за допомогою анотації @Controller або @RestController. Анотація @Controller позначає клас як компонент-контролер, який обробляє запити і повертає модель та вид (HTML, JSON і т.д.). [16]

У контролерах визначаються методи, які відповідають на різні запити HTTP (GET, POST, PUT, DELETE і т.д.). Для цього використовуються анотації, такі як @RequestMapping, @GetMapping, @PostMapping і т.д., які вказують URL шляхи та типи запитів, які методи повинні обробляти. [16]

Методи контролерів можуть мати параметри, що вказують на дані,

передані у запиті, такі як шляхи, параметри запиту або тіло запиту. Зазвичай, результатом роботи методів є модель даних або об'єкт, який серіалізується у відповідь і надсилається клієнту.

Контролери в Spring є ключовим елементом у створенні веб-додатків та API. Вони дозволяють взаємодіяти з клієнтами. Завдяки контролерам можна встановлювати маршрутизацію, обробляти різноманітні дії користувачів та виконувати бізнес-логіку додатку.

Один з ключових аспектів контролерів - це обробка валідації вхідних даних. За допомогою анотацій, таких як @Valid, можна встановити правила перевірки вхідних параметрів та тіло запиту, що допомагає забезпечити коректність та безпеку введених даних.

Крім базової функціональності, контролери можуть використовувати додаткові функціональності Spring Framework, такі як аутентифікація і авторизація за допомогою анотацій @Secured, @PreAuthorize, @PostAuthorize та інших. Це дозволяє контролерам забезпечувати доступ до ресурсів залежно від прав користувача та забезпечувати безпеку додатку.

Організація коду в контролерах також є важливим аспектом. Зазвичай рекомендується створювати окремий контролер для кожної сутності або функціональної області додатку. Це допомагає зберігати код організованим та зрозумілим, полегшує розширення та підтримку додатку. Для даного веб-додатку був обраний варіант розподілення контролерів відповідно функціональності, яку вони виконують. Тому було створено два додаткових пакети у директорії controllers – admin та user.

Приклад контролера, що відповідає за поповнення балансу користувача можна побачити на рис. 3.8.

```java
3 usages  DmytroTrots
@Controller
public class BalanceController {

    1 usage
    private static final Logger logger = LoggerFactory.getLogger(BalanceController.class);

    2 usages
    private final UserServiceImpl userService;

    DmytroTrots
    public BalanceController(UserServiceImpl userService) { this.userService = userService; }

    DmytroTrots
    @GetMapping("/top-up")
    public String topUpBalance() { return "BalancePage"; }

    DmytroTrots
    @PostMapping("/top-up")
    public String topUpBalancePost(RedirectAttributes redirectAttributes, @RequestParam("balance") BigDecimal balance, HttpServletRequest request){
        BigDecimal currentBalance = (BigDecimal) request.getSession().getAttribute( s: "BALANCE");
        Integer userId = (Integer) request.getSession().getAttribute( s: "ID");
        BigDecimal updatedBalance = userService.topUpBalance(balance, currentBalance, userId);
        String lang = String.valueOf(LocaleContextHolder.getLocale());
        if (lang.equals("en_US") || lang.equals("en")) {
            redirectAttributes.addFlashAttribute( attributeName: "ex",  attributeValue: "Balance u");
        }else{
            redirectAttributes.addFlashAttribute( attributeName: "ex",  attributeValue: "Баланс поповнено");
        }
        request.getSession().setAttribute( s: "BALANCE", updatedBalance);
        logger.info("Balance top-uped -->" + userId + ", " + balance);
        return "redirect:/top-up";
    }
}
```

Рис. 3.8 Контролер BalanceConroller

У методі topUpBalance() можна побачити, що повертається значення типу String. Насправді це назва файлу з розширенням HTML, що знаходиться у папці resources -> templates. Сторінку поповнення балансу можна побачити на рис. 2.9.

```html
<!DOCTYPE html>
<html th:lang="en" xmlns:th="http://www.thymeleaf.org" xmlns="http://www.w3.org/1999/html">
<head>
    <meta charset="UTF-8">
    <title th:text="#{label.topUpBalance}"></title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.3/css/bootstrap.min.css">
    <link type="text/css" rel="stylesheet" th:href="@{/css/style.css}">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css"/>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
</head>
<body>

<div th:replace="header :: head"></div>

<div align="center">
    <span style="..." th:if="${ex != null}">
            <div class="alert alert-warning">
                <strong th:text="${ex}"></strong>
            </div>
    </span>
    <h1 th:text="#{label.topUpBalance}"></h1>
    <form th:action="@{/top-up}" method="post">
        <input type="number" name="balance" min="1" step="0.01"/>
        <input type="submit" th:value="#{label.submit}"/>
    </form>
</div>

</body>
</html>
```

Рис. 3.9 HTML документ BalancePage

Для динамічного виведення інформації використовується HTML в поєднанні з Thymeleaf, бібліотека для якого була підключена ще на етапі генерації проекту. Для скорочення об’єму всіх HTML файлів, використовується тег та метод th:replace. За допомогою цього можна замінити даний рядок на інформацію з окремого файлу, в якому міститься код для відображення шапки сайту. Для стилізації використовується CSS документ та Bootstrap стилі, які підключені у блоці <head> за допомогою тегів <link>.

Додатковою можливістю при створенні Web-додатку є фільтри. В Spring Boot, Filter - це механізм, який дозволяє обробляти та модифікувати HTTP-запити та відповіді у веб-додатку. Фільтр діє як прошарок між запитами та відповідями, дозволяючи виконувати певну логіку перед тим, як запит буде переданий до контролера або після того, як відповідь була згенерована контролером. Принцип дії фільтра зображений на рис. 3.10.

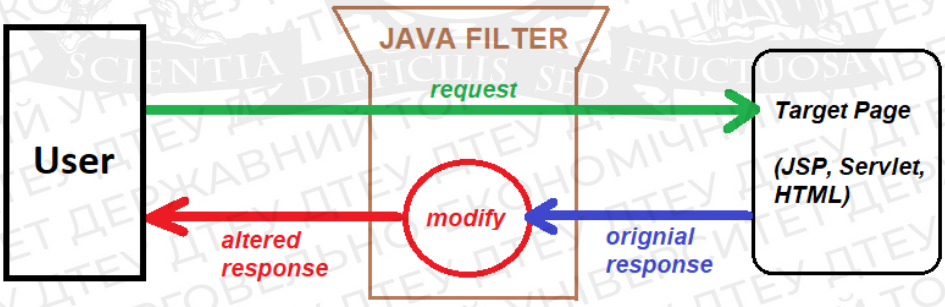


Рис. 3.10 Схема модифікації відповіді фільтром

У Spring Boot фільтри реалізуються за допомогою інтерфейсу javax.servlet.Filter, який включає метод doFilter(). Цей метод отримує об'єкти ServletRequest, ServletResponse та FilterChain, і дозволяє виконувати логіку фільтрації.

Фільтри можуть виконувати такі дії, як перевірка автентифікації, авторизація, логування, перекодування параметрів запиту, компресія

відповіді та багато іншого. Вони також можуть модифікувати або заміняти запит або відповідь.

У Web-додатку «Periodicals» фільтр відповідає за перенаправлення користувача на сторінку авторизації у випадку, якщо він заблокований адміністратором. Код даного класу можна побачити на рис. 3.11.

```
DmytroTrots
@Component
public class BanFilter implements Filter {

    2 usages
    private final UserServiceImpl userService;

    DmytroTrots
    public BanFilter(UserServiceImpl userService) {
        this.userService = userService;
    }

    DmytroTrots
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        UserEntity user = (UserEntity) ((HttpServletRequest) servletRequest).getSession().getAttribute( s: "USER");
        if (user != null && userService.findByUsername(user.getUsername()).getBanStatus() != null) {
            ((HttpServletRequest) servletRequest).getSession().invalidate();
            ((HttpServletResponse) servletResponse).sendRedirect( s: "/login");
        } else {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }
}
```

Рис. 3.11 Клас BanFilter

Клас BanFilter лежить у пакеті utils, де лежать і інші класи для імплементації додаткової логіки:

1. CreateReportOrder – клас, функціонал якого виконується по розкладу кожного дня о півночі та відправляє адміністратору на пошту сформований звіт у форматі PDF про оформлені замовлення за день.
2. DeleteSubscriptionSchedule – клас, функціонал якого також запускається кожного дня та видаляє з БД підписки на періодичні видання, якщо їх термін закінчився.
3. LoginSuccessListener – клас, який відповідає за внесення інформації до сесії, а саме ролі користувача, його ідентифікатора та ім’я. Крім того, клас логує дану інформацію в консоль, що

дозволяє бачити історію входу користувачів на сайт.

4. Mailer – клас, що дозволяє відправляти листи на Email адреси. Це корисно при зверненні користувачів до адміністрації магазину через спеціальну сторінку відправки повідомлень.

Після створення всіх вищеописаних класів, що відповідають за виконання бізнес-логіки сайту, структура проекту має вигляд як на рис 3.12. Надалі будуть створюватися тільки конфігураційні файли та класи, що допоможуть у майбутній розробці додатку або у зручності його використання.

Рис. 3.12 Структура проекту з функціональними класами

Для того, щоб проект можна було розгорнути на будь якому сервері, потрібно використати контейнеризацію за допомогою Docker.

Docker - це одна з найпопулярніших платформ контейнеризації, яка дозволяє упаковувати програмні компоненти і їх залежності в стандартизовані контейнери. Docker використовує контейнери з ізольованими середовищами, що дозволяє додаткам працювати одночасно на одній фізичній машині, але з відокремленими ресурсами та залежностями.

[17]

Контейнеризація - це методологія розробки та розгортання програмного забезпечення, яка дозволяє упаковувати програмні компоненти та їх залежності відокремлено один від одного у виконуваний контейнер. Контейнери забезпечують стандартизоване середовище для запуску програм, що дозволяє розробникам ефективно тестувати, розгортати та масштабувати свої додатки. [18]

В контейнері зазвичай запускається образ нашого проекту. Щоб його створити, потрібно створити файл Dockerfile, який запускається за допомогою команди «docker build .». Після цього можна завантажити створений образ у Dockerhub, що надасть можливість отримати наш проект з будь якої точки світу.

Для того, щоб разом з основним проектом створювалась база даних, потрібен файл docker-compose.yml, що запускається командою «docker-compose up -d». Файл docker-compose.yml має вигляд як на рис. 3.13.

```yaml
services:
  java-mysql:
    image: mysql
    container_name: database
    command: mysqld --sql_mode="STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION"
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_DATABASE=dbperiodicalsspring
    healthcheck:
      test: [ "CMD", "mysqladmin" ,"ping", "-h", "localhost" ]
      timeout: 20s
      retries: 10

  app:
    image: dmytrotrots/springboot-periodicals-image
    container_name: application
    ports:
      - "8083:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://java-mysql:3306/dbperiodicalsspring?autoReconnect=true&useSSL=false&allowPublicKeyRetrieval=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: password
    depends_on:
      java-mysql:
        condition: service_healthy
```

Рис. 3.13 Файл docker-compose.yml

Ще одним конфігураційним файлом є maven.yml, який налаштовує

CI/CD нашого сайту.

CI/CD (Continuous Integration/Continuous Deployment) - це практика розробки програмного забезпечення, яка дозволяє автоматизувати процеси інтеграції нового коду, тестування та впровадження змін на веб-сайті з мінімальною ручною втручанням. [19]

CI - це процес постійної інтеграції, коли розробники регулярно злиттяють свій код в спільний репозиторій, де він проходить автоматизовану перевірку на наявність помилок, конфліктів та збійних ситуацій. Цей процес дозволяє виявляти проблеми рано і швидко виправляти їх, забезпечуючи стабільність проекту. [19]

CD - це процес постійної доставки або розгортання, який включає автоматизоване тестування і впровадження змін на веб-сайті після успішного проходження тестів. CD дозволяє зменшити час і ризики, пов'язані з ручним розгортанням, та забезпечити швидку доставку нового функціоналу або виправлень на продуктивну веб-сторінку. [19]

Код CI/CD файлу можна побачити в додатку А.

На цьому закінчується розробка основних частин сайту, в результаті яких у Web-додатку присутні такі можливості:

1. Для користувача: створення облікового запису, поповнення балансу, покупка періодичних видань, додавання товарів у корзину, модифікація товарів у корзині, відправка повідомлень адміністратору.
2. Для адміністратора: створення нових користувачів, створення або модифікація існуючих періодичних видань, блокування користувачів, обробка замовлень.

Після запуску проекту за допомогою docker-compose.yml, доступ до сайту доступний за допомогою ресурсу localhost:8083/login, а візуальну

частину сайту можна оглянути у додатках Б-Д.

3.3 Тестування готової системи

Модульне тестування є важливим етапом у розробці програмного забезпечення, особливо коли застосовується підхід BDD (Behavior-Driven Development). Використовуючи цей підхід, тести пишуться після розробки коду, але до його впровадження.

У модульному тестуванні перевіряється правильність роботи окремих модулів системи незалежно від інших компонентів. Такий підхід дозволяє виявити та виправити помилки на ранніх етапах розробки, забезпечуючи надійність та якість системи. [20]

Під час модульного тестування використовуються спеціальні фреймворки, такі як JUnit, для написання тестових сценаріїв та перевірки очікуваних результатів. Кожен модуль системи тестується окремо, замінюючи його залежності на підроблені об'єкти (mock objects), які поводяться аналогічно реальним об'єктам.

Наша система потребує в основному модульного тестування тільки сервісів. Тому було додано вісім класів для тестування, структуру яких можна побачити на рис. 3.14.

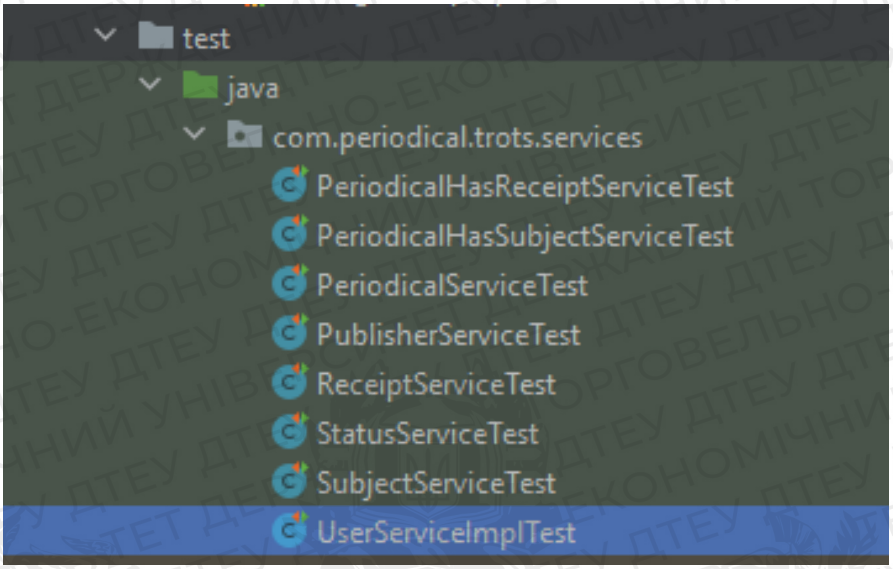


Рис. 3.14 Структура класів-тестів

Приклад класу, що тестує логіку сервісу StatusService можна побачити на рис. 3.15.

```
DmytroTrots
@ExtendWith(MockitoExtension.class)
public class StatusServiceTest {

    1 usage
    @InjectMocks
    private StatusService testInstance;

    1 usage
    @Mock
    private StatusRepository statusRepository;

    DmytroTrots
    @Test
    public void shouldGetStatusById() {
        StatusEntity expectedStatus = mock(StatusEntity.class);
        when(expectedStatus.getId()).thenReturn( t: 1);
        when(statusRepository.getById(expectedStatus.getId())).thenReturn(expectedStatus);

        StatusEntity actualStatus = testInstance.getStatusById(expectedStatus.getId());

        assertEquals(expectedStatus, actualStatus);
    }
}
```

Рис. 3.15 Клас-тест, що тестує сервіс StatusSerice

Завдяки модульному тестуванню та підходу BDD, ми можемо забезпечити високу якість та функціональність нашого онлайн магазину з

продажу періодичних видань. Тести допомагають виявляти й виправляти помилки, забезпечуючи стабільну та надійну роботу системи.

ВИСНОВКИ ТА РЕЗУЛЬТАТИ

У рамках дипломної роботи було проведено поглиблене дослідження процесу розробки магазину з продажу періодичних видань, використовуючи передові технології, зокрема Java, Spring, MySQL, Docker і CI/CD.

Під час проведення дослідження були ретельно проаналізовані основні вимоги до функціональності та дизайну онлайн-магазинів. Вивчення найкращих практик в галузі електронної комерції дало змогу визначити оптимальні шляхи реалізації функціональності, що необхідна для створення ефективного онлайн-магазину з продажу періодичних видань. Результатом цього дослідження стало програмне забезпечення, яке забезпечує повний функціонал такого магазину, надаючи зручність для клієнтів та оптимізуючи процес замовлення, оплати та керування інформацією про наявні товари.

Ключовими етапами процесу розробки були аналіз вимог, проектування, розробка, тестування та впровадження. Застосування сучасних технологій, таких як Java, Spring, MySQL, Docker і CI/CD, мало вирішальне значення для створення стабільної та масштабованої архітектури. Використання Java та Spring дозволило створити надійні та ефективні рішення, а MySQL забезпечив ефективну роботу з базою даних. Використання Docker дало змогу забезпечити швидке та просте розгортання програмного забезпечення, а CI/CD дозволив автоматизувати процеси розгортання та тестування, забезпечивши більшу ефективність розробки.

Отже, дана випускна кваліфікаційна робота є дуже важливим та актуальним дослідженням, оскільки вона не тільки дає розуміння основних вимог до онлайн-магазинів, спеціалізованих на продажу певної категорії товарів, але і надає практичні рекомендації для ефективної розробки подібних проектів. Результати цього дослідження можуть бути використані

розробниками та підприємцями для створення успішних та конкурентоздатних онлайн-магазинів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A website on building software effectively : MonolithFirst [Електронний ресурс] / M. Flower – Режим доступу: https://martinfowler.com/bliki/MonolithFirst.html.
2. Newman S. Building Microservices: Designing Fine-Grained Systems / Newman Sam. – К. : O'Reilly Media, 2014. – 201 с.
3. SCVOT – ONOFFLINE SCHOOL: Що таке вайрфрейм, мокап, прототип [Електронний ресурс] – Режим доступу: https://voll.com.ua/uk/glossary/vajrfrejmy-wireframes.
4. Oracle : Java [Електронний ресурс] – Режим доступу: https://www.oracle.com/uk/java/.
5. Spring : Spring Framework [Електронний ресурс] / VMware Tanzu – Режим доступу: https://spring.io/projects/spring-framework.
6. Crockford D. JavaScript: The Good Parts / Douglas Crockford. – К. : O'Reilly Media, 2008. – 176 с.
7. Matthes E. Python Crash Course / Eric Matthes – К. : San Francisco: No Starch Press, 2019. – 560 с.
8. Ruby S. Agile Web Development with Rails / Sam Ruby, Dave Thomas, David Heinemeier Hansson. – К. : Pragmatic Bookshelf, 2019. – 1-11 с.
9. Stauffer M. Laravel: Up & Running / Matt Stauffer – К. : O'Reilly Media, 2016. – 23 с.
10. Codd E. F. A Relational Model of Data for Large Shared Data Banks / Edgar F. Codd. – К. : Association for Computing Machinery, 1970. – 377–387 с.
11. PostgreSQL [Електронний ресурс] / PostgreSQL Global Development Group – Режим доступу: https://www.postgresql.org.
12. NoSQL: Основи та використання / В.О. Кузнецов. – К. : Діалектика, 2014.

– 216 с.

13. MongoDB: Повне керівництво з розробки / В. Банга, П. Доссотт. – К. : Видавництво "БІНОМ", 2017. – 440 с.
14. Techrocks : SQL або NoSQL: що краще вибрати для вашого проекту? [Електронний ресурс] / Ліам Джоунс - Режим доступу: https://techrocks.ua/sql-abo-nosql-shho-krashhe-vibrati-dlya-vashogo-proektu/.
15. Hernandez M. J. Database Design for Mere Mortals / Michael J. Hernandez. – К. : Indianapolis, Indiana: Addison-Wesley Professional, 2013. – 402 с.
16. Spring : Spring Data JPA Reference Documentation [Електронний ресурс] / VMware Tanzu – Режим доступу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.
17. Docker : Overview [Електронний ресурс] / Docker – Режим доступу: https://docs.docker.com/get-started/.
18. Amazon : Containerization [Електронний ресурс] / Amazon – Режим доступу: https://aws.amazon.com/ru/what-is/containerization/.
19. GitLab : What is CI/CD [Електронний ресурс] / GitLab – Режим доступу: https://about.gitlab.com/topics/ci-cd/.
20. Kolawa A. Automated Defect Prevention: Best Practices in Software Management / Adam Kolawa, Dorota Huizinga – К. : Wiley-IEEE Computer Society Press. – 21 с.

ДОДАТКИ

Додаток А

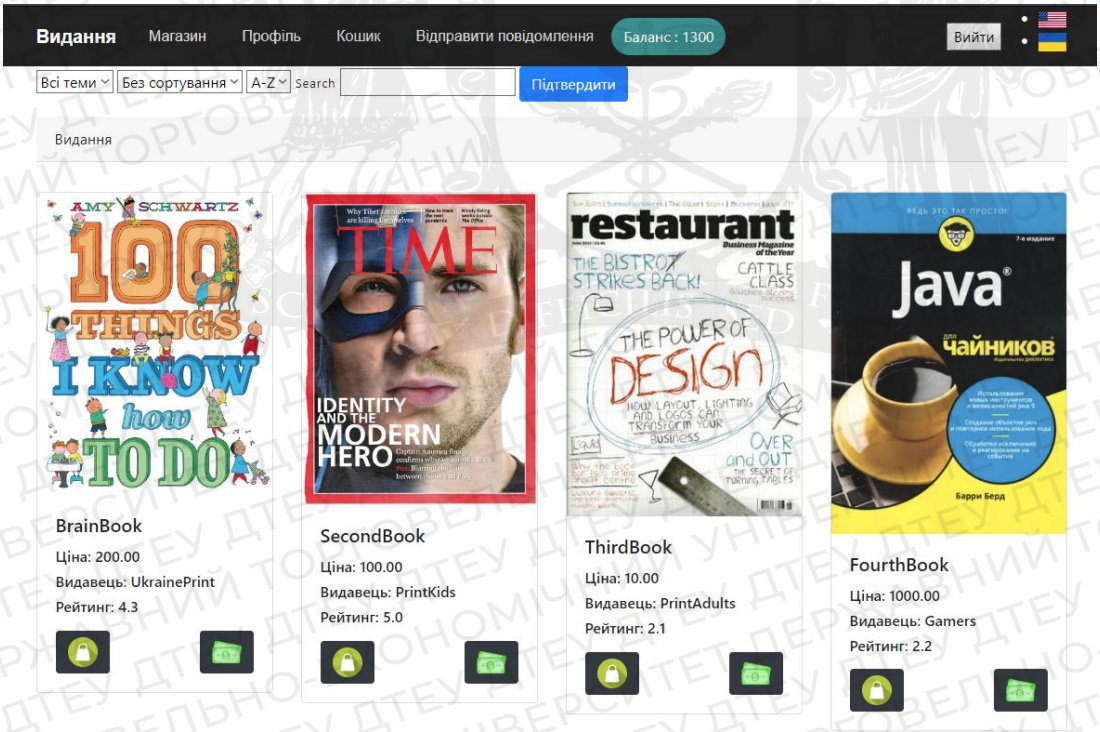
Код файлу maven.yml

```
name: Java CI/CD with Maven

on:
  push:
    branches: [ master ]
  pull request:
    branches: [ master ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 11
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn clean install
      - name: Set up QEMU
        uses: docker/setup-qemu-action@v1
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1
      - name: Login to DockerHub
        uses: docker/login-action@v1
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
      - name: Build and push
        uses: docker/build-push-action@v2
        with:
          context: .
          push: true
          tags: dmytrotrots/springboot-periodicals-image:latest

  deploy:
    needs:
      - build
    runs-on: ubuntu-latest
```

```
steps:
  - name: Deploy to Digital Ocean via SSH
    uses: appleboy/ssh-action@master
    with:
      host: ${{ secrets.HOST_SERVER }}
      username: ${{ secrets.USERNAME_SERVER }}
      passphrase: ${{ secrets.PASSWORD_SERVER }}
      key: ${{ secrets.SSH_KEY_SERVER }}
      script:
        cd Spring-WEB-Application-Periodicals-;
        docker pull dmytrotrots/springboot-periodicals-image;
        docker-compose up –d
```

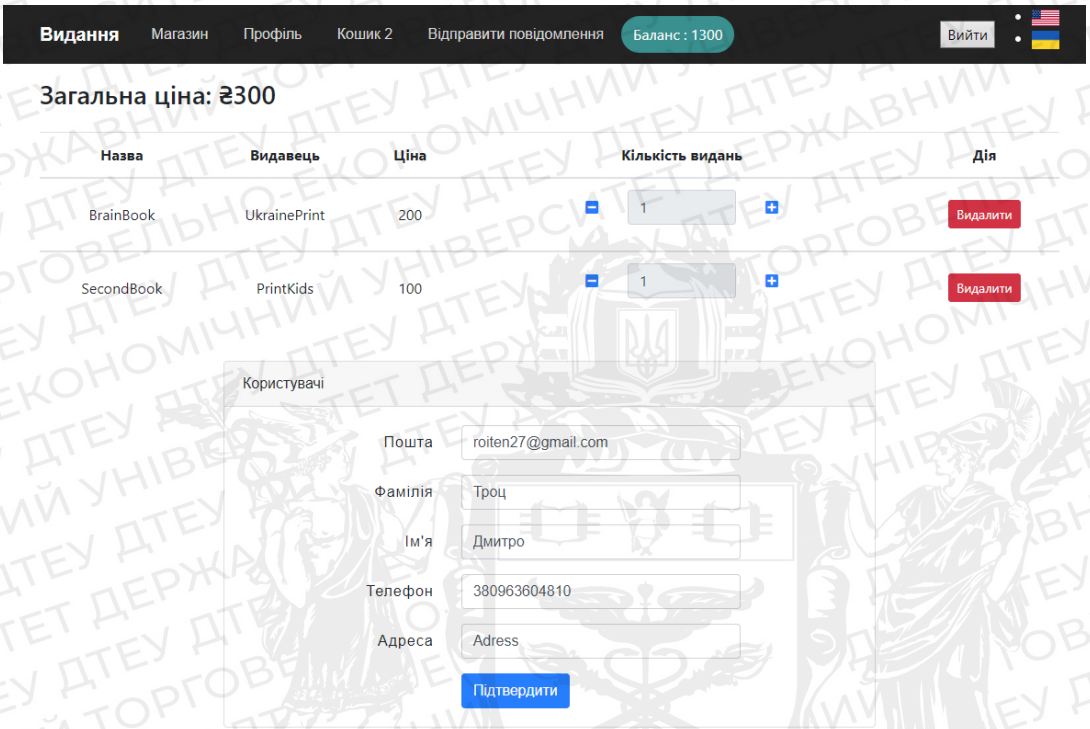
Додаток Б

Головна сторінка магазину



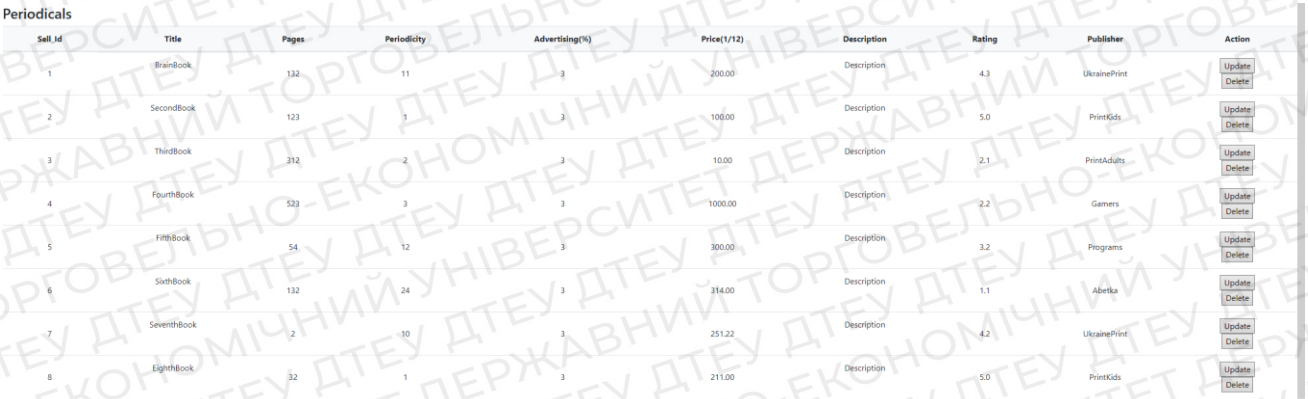
Додаток В

Корзина з вибраними товарами



Додаток Г

Сторінка керування існуючими періодичними виданнями



Додаток Д

Сторінка керування та створення користувачів адміністратором