МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

«Веб-застосунок для управління запасами ліків»

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-41

_____Харламов І. Є._____
(прізвище та ініціали)

Керівник _____Зосімов В. В._____
(прізвище та ініціали)

_____ д.т.н., доц._____
(науковий ступінь, звання)

Унікальність тексту 93%

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри прикладних інформаційних систем
Протокол №14 від 23 травня 2023 р.

зав. кафедри Плескач В.Л.

Київ – 2023

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2023	
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2023	
9.	Подання роботи у першому варіанті	28.04.2023	
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	22.05.2023	
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	26.05.2023	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	
14.	Захист кваліфікаційної роботи бакалавра	29.06.2023	

Здобувач вищої освіти _____
(підпис)

Керівник _____

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	2
Вступ	2
1	13
2	40
3	21
Висновки	2
Перелік використаних джерел	7
Додатки	7

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Харламов І Є.			Відомість дипломної роботи	Лист	Листів
Керівн.	Зосімов В.В.					

Н/контр.	Макаренко С.А.		26.05.2023		
Зав. каф.	Плескач В.Л.				

АНОТАЦІЯ

Дипломна робота: 100с., 16 рис., 10 табл., 70 джерел, 2 дод.

Ця кваліфікаційна робота бакалавра присвячена проектуванню та розробленню веб-застосунку для управлінням запасами ліків.

Метою кваліфікаційної роботи бакалавра є ефективне управління запасами ліків на основі розробленої веб-системи обліку ліків.

Завдання дослідження:

- Дослідити загальні принципи побудови веб-застосунків та проаналізувати існуючі програми планування ресурсів підприємства
- Здійснити аналіз програмно-технологічних рішень для створення веб-орієнтованої системи керування ліків.
- Розробка структури системи з урахуванням основних вимог.
- На основі проведених досліджень здійснити проектування та розробку системи для управлінням ліків у формі веб-застосунку.

Об'єкт дослідження - процеси управління запасами ліків на аптечних складах

Предмет дослідження.

Програмно-технічні засоби, принципи, підходи щодо побудови програмної системи веб-застосунку управлінням запасами ліків.

Методи дослідження: критичний аналіз сучасних технологій розробки веб-застосунків; системний аналіз програмно-технічних рішень і серверної архітектури; узагальнення; систематизація; компонентно-орієнтоване програмування; UML-моделювання, та аналогія залучені в процесі проектування; розробки та побудови власного веб-застосунку управління запасами ліків.

Ключові слова: планування ресурсів підприємства, SPA, MVC, клієнт-серверна архітектура, Java Spring, Angular.

ABSTRACT

Thesis: 100 pages, 16 figures, 10 tables, 70 sources, 2 appendices.

This thesis is devoted to the design and development of a web application for managing medication inventory.

The purpose of the bachelor's qualification work is the effective management of drug stocks based on the developed web-based drug accounting system.

To achieve this goal you need to solve the following **tasks**:

- Investigate the general principles of building web applications and analyze existing enterprise resource planning (ERP) software solutions.
- Analyze programmatic and technological solutions for creating a web-oriented medication management system.
- Develop the system structure, considering the key requirements.
- Based on the conducted research, design and develop a web application for medication inventory management.

The research object is the process of managing medicine stocks in pharmacy warehouses.

The subject of the study encompasses programmatic, technical principles, and approaches for building a web application for medication inventory management.

The research methods.

Critical analysis of modern web application development technologies; system analysis of programmatic and technical solutions, server architecture; Generalization; Systematization; UML modeling, and analogy involvement during the design; Development, and construction of a custom web application for medication inventory management.

Keywords: enterprise resource planning, SPA, MVC, client-server architecture, Java Spring, Angular.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - Application Programming Interface

REST – Representational State Transfer

БД – База даних

SQL - Structured Query Language

CSRF - Cross-Site Request Forgery

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

MVC – Modal-View-Controller

CSS – Cascading Style Sheets

SCSS – Sassy CSS

XML – Extensible Markup Language

UX – User experience

UI – User interface

ERP - Enterprise Resource Planning

CRM - Customer relationship management

EBS - Oracle E-Business Suite

DBMS - Database Management System

СУБД - Система управління базами даних

SPA - Single Page Application

MPA - Multi-Page Application

PWA - Progressive Page Application

DAC - Discretionary access control

MAC - Mandatory access control

RBAC - Role-based access control

ABAC- Attribute-based access control

SLA - Service-level agreement

IasS - infrastructure as a service

PasS - platform as a service

SasS - software as a service

SSR - Server-Side Rendering

SEO - Search Engine Optimization

AJAX – asynchronous JS & XML

RxJS – Reactive Extensions for JS

JSON - JavaScript Object Notation

JVM - Java Virtual Machine

WORA - Write Once, Run Anywhere

JPA - Java Persistence API

ORM - Object-Relational Mapping

JPQL - Java Persistence Query Language

DI - Dependency Injection

IoC - Inversion of Control

CLI - Command Line Interface

AOT - Ahead-of-time

MVVM - Model-View-ViewModel

DOM - Document Object Model

SSG - Static site generator

MVCC - Multiversion concurrency control

CTE - Common Table Expressions

WAL - Write-Ahead Logging

LAMP - Linux, Apache, MySQL i PHP

JWT- JSON Web Token

DTO – Data Transfer Object

ВСТУП

Актуальність цієї теми

У сучасних умовах розвитку вітчизняної економіки та посилення конкуренції на внутрішньому ринку значно зростає роль торгівлі. Це зумовлює необхідність удосконалення існуючих систем і методів та впровадження нових інструментів управління бізнес-проектами.

Особливо актуально це питання стоїть для українського фармацевтичного ринку, останні роки для якого були дуже складними. У період сезонних захворювань, коли збільшується навантаження на аптечну систему, під час пандемії COVID-19 аптеки докладали максимум зусиль, щоб забезпечити доступність важливих лікарських засобів для населення.

Важливою умовою ефективного функціонування аптек являється управління товарними запасами. Обсяг, склад та структура товарних запасів забезпечують ритмічність і безперебійність торгово-операційного процесу, стійкість товарної пропозиції та лояльність покупців. Ефективне управління товарними запасами створює передумови успішної традиційної та Інтернет-торгівлі лікарськими засобами. Для цього в Україні, як і у всьому світі, аптечні мережі активно використовують спеціалізовані програмні рішення та веб-застосунки для обліку ліків. Ці системи дозволяють автоматизувати процеси замовлення, постачання, контролю запасів, реєстрації продажів та ведення документації. Вони допомагають покращити ефективність роботи, зменшити помилки та забезпечити дотримання нормативно-правових вимог у сфері обліку лікарських засобів.

На сьогодні під час повномасштабної війни особливо гостро питання обліку ліків виникає через блокування і втрату складів через бойові дії, проблеми з логістикою, закриття або руйнування значної кількості аптек. Оскільки роль аптек у кризових умовах має вирішальне значення для збереження здоров’я населення, створення веб-застосунків для обліку ліків, функція яких полягає у забезпеченні контролю за запасами лікарських засобів, уникненні їх нестачі або перевищення, забезпечення своєчасної

поставки, не викликає сумнівів і потребує реалізації.

Метою кваліфікаційної роботи бакалавра є розробка веб-системи обліку для ефективного управлінням запасами ліків в аптечних закладах України.

Завдання дослідження:

- Дослідити особливості побудови, функціональні можливості систем керування і обліку продукції.
- Здійснити аналіз програмно-технологічних рішень для створення веб-орієнтованої системи керування ліків.
- Розробка структури системи з урахуванням основних вимог.
- На основі проведених досліджень здійснити проектування та розробку системи для управлінням ліків у формі веб-застосунку.

Об'єкт дослідження - процеси управління запасами ліків на аптечних складах

Предмет дослідження.

Програмно-технічні, принципи, підходи щодо побудови програмної системи веб-застосунку управлінням запасами ліків.

Методи дослідження: критичний аналіз сучасних технологій розробки веб-застосунків; системний аналіз програмно-технічних рішень і серверної архітектури; узагальнення; систематизація; компонентно-орієнтоване програмування; UML-моделювання, та аналогія залучені в процесі проектування; розробки та побудови власного веб-застосунку управління запасами ліків.

Практичне значення одержаних результатів полягає у тому, що розроблений веб-застосунок може стати корисним, зручним та сучасним, а головне легко маштабованим рішенням для малого бізнесу. Розроблений веб-застосунок може бути використаний як і для ведення продажу ліків користувачам так і для менеджменту запасами ліків.

Структура роботи Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділи та висновку.

РОЗДІЛ І ТЕОРЕТИЧНІ ОСНОВИ ВЕБ-ЗАСТОСУНКІВ

1.1 Основні поняття, технічна характеристика, переваги сучасних веб-застосунків

1.1.1 Основні терміни та ключові підходи сучасних веб-застосунків

Веб-застосунки стають все більш актуальними в сучасному житті. Сучасний веб-застосунок – це унікальна розробка, орієнтована на вирішення певних завдань, наприклад бронювання готелів та квитків, автомобілів; покупка товарів; замовлення різних послуг; фінансові та банківські послуги різного типу (калькулятор валют або повноцінний онлайн-банкінг); соціальні мережі; освітні програми, отримання державних послуг онлайн тощо.

Веб-застосунки – це *програма-клієнт-сервер*, що зберігається на віддаленому сервері, яка використовує *веб-браузери* та веб-технології для виконання певних функцій через Інтернет через інтерфейс браузера [1, 2]. Базове визначення веб-застосунка – *це програма, яка працює у браузері*. Веб-застосунок вирішує конкретну проблему, є інтерактивним та має систему управління контентом. Оскільки веб-застосунки є прикладною програмою клієнт-сервер, тому в середовищі клієнт-сервер багато комп'ютерів можуть обмінюватися інформацією, як збереження інформації в базі даних.

Ключові підходи та терміни, які використовуються в сучасних веб-застосунках [2 - 4]:

- *Модель клієнт-сервер* — веб-застосунки, як правило, використовують архітектуру клієнт-сервер, де клієнт (веб-браузер) спілкується із сервером, на якому розміщено застосунок і зберігаються його дані.
- Використання, мов розміток, шаблонів стилів і створення динамічних веб-сторінок.
- *REST API* (Representational State Transfer Application Programming Interface) – це архітектурний стиль та набір правил, який використовується для створення веб-служб і взаємодії клієнт-сервер.

- Використання баз даних (БД), яка забезпечує зберігання та керування даними, необхідними для функціонування веб-застосунка.
- Використання криптографічних методів, блоків в розподіленій БД Blockchain, які пов’язані між собою та складаються з серії підтверджених транзакцій, які записуються одна за одною.
- Використання інноваційних технологій, зокрема хмарні технології, які надають гнучкий і зручний спосіб обробки, зберігання і надання доступу до інформації.

1.1.2 Основні характеристики веб-застосунку

Автори літературних джерел вказують на такі основні характеристики веб-застосунків [3 - 4]:

- *доступ через веб-браузер*: доступ до веб-програми здійснюється через веб-браузер, наприклад Chrome, Firefox або Safari. Користувачі можуть отримати доступ до програми з будь-якого місця, де є підключення до Інтернету, без необхідності завантажувати чи встановлювати програмне забезпечення.
- *незалежність від платформи*: веб-програми не прив’язані до певної операційної системи чи пристрою, їх можна використовувати на будь-якій платформі, яка підтримує веб-браузер.
- *інтерактивність:* веб-застосунки дозволяють користувачам взаємодіяти з програмою, часто через графічний інтерфейс користувача (GUI), який відображає інформацію та приймає введення користувача.
- *масштабованість*: можуть обробляти значну кількість користувачів і запитів, не відчуваючи проблем із продуктивністю.
- *безпека*: вимагають заходів безпеки для захисту від несанкціонованого доступу та порушень даних, які можуть включати шифрування, контроль доступу та автентифікацію користувачів.

- *технічне обслуговування та оновлення*: веб-програми можна легко оновлювати та підтримувати завдяки можливості додавати нові функції та виправляти помилки, не вимагаючи від користувачів завантажувати чи встановлювати нове програмне забезпечення.

1.1.3 Основні технічні поняття веб-застосунків

Аналіз доступних літературних та наукових джерел інформації дозволив виділити основні технічні поняття та вимоги при розробці веб-застосунків [1-6]:

- Веб-програми працюють *за архітектурою клієнт-сервер;*
- Веб-застосунки мають два рівні, що тісно взаємодіють один з одним:
  - *Клієнтський код*, що виконується у браузері. Розробка клієнтської частини веб-програми передбачає використання мов програмування: HTML (відповідає за структурування змісту сторінки), CSS (надає веб-сторінці певний вигляд), JavaScript (основна мова програмування). Клієнтський код взаємодіє з сервером лише через HTTP запити, й не може безпосередньо отримувати інформацію з сервера.
  - *Серверний код складається з двох частин:* логіки застосунка – це головний центр керування веб-застосунком та БД.
- Веб-програми, як правило, використовують API для зв’язку з іншими системами чи службами, такими як платіжні шлюзи, платформи соціальних мереж або постачальники послуг електронної пошти.
- Веб-програмам, зазвичай, необхідні БД для зберігання та отримання даних.
- При розробці веб-застосунків часто використовують фреймворки та бібліотеки, які надають готові модулі, компоненти, шаблоний код, реалізують архітектурні патерни.

- Веб-програми потребують надійної системи безпеки для захисту від кіберзагроз, таких як атаки впровадження SQL, міжсайтовий сценарій (XSS) або підробка міжсайтового запиту (CSRF).
- Веб-програми необхідно розробляти з урахуванням масштабованості, щоб справлятися зі зростаючим трафіком, вимогами до зберігання даних і вимогами користувачів.

1.2 Рівнева архітектура веб-застосунку

Рівнева архітектура застосування визначає схему взаємодії між різними компонентами системи, зручним є розподіл архітектури на логічні рівні, в якому відбувається фізичне розділення системи на окремі програмні компоненти та функції [5 - 8].

Презентаційний рівень (Client-side) дозволяє користувачам взаємодіяти з сервером через браузер. На цьому рівні з'являється дизайн UI/UX, інформаційні панелі, сповіщення, налаштування конфігурації, макет та інтерактивні елементи.

Рівень програми (Server-side): виконує різні функції: отримує запити користувачів, виконує бізнес-логіку та доставляє необхідні дані до зовнішніх систем. Він містить такі основні компоненти:

- *веб-сервер* є ключовим компонентом архітектури веб-застосунків, який отримує запити користувачів, обробляє HTTP-запити, виконує бізнес-логіку та доставляє необхідні дані для користувачів.
- *API* – це набір правил, протоколів та інструментів, які визначають спосіб взаємодії між клієнтською і серверною частинами веб-застосунка.
- *Бізнес-логіка* – це набір компонентів, які реалізують бізнес-правила та функціональність веб-застосунку. Вона визначає правила обробки даних, розрахунків, валідації тощо. Бізнес-логіка може бути реалізована в середині веб-сервера або окремими модулями, що взаємодіють з ним.

- *БД* – це ключовий компонент веб-програми, який забезпечує зберігання та керування даними, необхідними для функціонування веб-застосунка. Вона зберігає інформацію про користувачів, продукти, замовлення, операції тощо та забезпечує доступ до даних і виконання запитів до них. При виборі БД необхідно враховувати її розмір, швидкість, масштабованість і структуру.
- *сервери або хмарні екземпляри* – це екземпляр віртуального сервера, створений, доставлений і розміщений за допомогою загальнодоступної або приватної хмари. Він працює як фізичний сервер, який можна легко переміщати між кількома пристроями або розгортати кілька екземплярів на одному сервері. Такі властивості надають йому динамічності, масштабованості та економічної ефективності. Використовуючи хмарні екземпляри, можна легко розгортати веб-застосунки та керувати ними в будь-якому середовищі.

Рівень даних (Database) – це ключовий компонент веб-програми, який зберігає та керує інформацією для веб-програми.

1.3 Основні переваги застосування веб-застосунків

Широке використання веб-застосунків можна пояснити такими факторами [4 - 8]:

- *доступність*: веб-програма доступна з будь-якої точки планети, де є доступ до Інтернету, з будь-якого пристрою.
- *зручність*: веб-застосунки можуть надати користувачам швидкий доступ до інформації та послуг без необхідності завантажувати та встановлювати програмне забезпечення.
- *налаштування:* функції веб-програм можна адаптувати відповідно до конкретних потреб користувачів.

- *співпраця*: веб-застосунки сприяють співпраці між користувачами, надаючи їм можливість працювати разом над проектами або обмінюватися інформацією та ресурсами.
- *управлінням даними*: веб-застосунки допомагають користувачам керувати та аналізувати велику кількість даних для прийняття відповідних рішень.

За результатами аналізу літературних та наукових джерел визначені основні переваги веб-застосунків у порівнянні з традиційними настільними програмами [1 - 8]:

- *відсутність програмного забезпечення для клієнтів*: доступ до веб-програм можна отримати з будь-якого пристрою, підключеного до Інтернету, що робить їх більш доступними, ніж програми для настільних комп’ютерів, які потребують інсталяції та оновлень. Це спрощує процес користування, також суттєво оптимізується процес обслуговування, модернізації інтерфейсу.
- *сумісність із різними платформами*: доступ до веб-програм можна отримати з різних пристроїв, операційних систем (ОС) і веб-браузерів, вони працюють на всіх ОС і у всіх існуючих браузерах. Завдяки цьому веб-програми є більш універсальними, ніж настільні програми, які обмежені певною платформою.
- *економічність*: розробка та обслуговування веб-застосунків, як правило, є менш фінансово затратними, ніж застосунків для персональних комп'ютерів (ПК), оскільки вони вимагають менше апаратних і програмних ресурсів, також немає необхідності розробки окремих продуктів і підтримки їх на інших системах.
- *легке оновлення та віддалена підтримка* без необхідності користувачам встановлювати оновлення вручну, що спрощує підтримку веб-програм в актуальному стані та їх безпеку.

- *інтеграція з іншими веб-службами*: соціальними мережами, аналітикою тощо, в результаті чого вони є більш універсальними та затребуваними для клієнтів.

Представлені властивості веб-застосунків роблять їх популярним вибором для підприємств, організацій і окремих користувачів, яким потрібен доступ до послуг та інформації з любого місця та в будь-який час.

1.2 Класифікація веб-застосунків відповідно до їх призначення

В залежності від задач та призначення веб-застосунки поділяються на такі основні типи [10 - 15]:

- Веб-застосунок для системи електронної комерції (E-commerce) розроблений для полегшення покупок онлайн для клієнтів [10, 11]. Його основні функції включають:
  - *перегляд каталогу*: надає можливість клієнту ознайомитися з представленими товарами та порівняти ціни;
  - *кошик для покупок*: дозволяє клієнтам додавати продукти до свого кошика, збільшувати або зменшувати їх кількість, та переходити до оформлення замовлення.
  - *платіжний шлюз*: полегшує онлайн-платежі за допомогою кредитно/дебетових карток, онлайн-гаманця та банківських переказів.
  - *керування замовленнями*: створює рахунки-фактури та відстежує статус замовлення до доставки.
  - *керування обліковими записами клієнтів*: зберігає профіль користувачів, їх історію покупок тощо.
  - *підтримка клієнтів*: надаються такі послуги, як поширені запитання, чат-боти, відгуки клієнтів.

Для виконання своїх функції E-commerce системи мають володіти такими характеристиками: *висока масштабованість* для роботи з високим трафіком і піковим навантаженням під час сезонної розпродажі тощо; *безпека*:

програма має забезпечувати конфіденційність, цілісність і доступність даних клієнтів, включаючи конфіденційну інформацію, таку як дані кредитної картки; *зручність використання та простота у навігації*; *продуктивність*: швидкий час відгуку, мінімальний час завантаження сторінки та підтримку кількох пристроїв, таких як ПК, планшети та мобільні телефони; *інтеграція з різними сторонніми системами*: платіжні шлюзи, постачальники послуг логістики та платформи соціальних мереж.

- Веб-застосунок системи CRM (Customer relationship management) – це програмний інструмент, який поліпшує співпрацю з клієнтами. CRM дозволяють автоматизувати продажі, від створення лідів до прискорення обробки повторних замовлень [11, 15]. Функції веб-застосунку системи CRM включають:

- *управління клієнтськими даними*: дозволяє підприємствам централізовано зберігати такі дані, як контактна інформація, історія покупок тощо.
- *управління продажами*: надає інструменти для керування та відстеження продажів, аналізу ефективності продажів.
- *автоматизація маркетингу* за допомогою електронної пошти та соціальних мереж.
- *керування* запитами клієнтів, скаргами та запитами на підтримку.
- *аналітика та звітність*: надає інформацію про поведінку клієнтів і показники продажів за допомогою функцій аналітики та звітності.

Таким чином, CRM дає можливість розробити базу даних, завдяки якій можна простежити усю взаємодію компанії з певним клієнтом, що дозволяє побудувати грамотну маркетингову стратегію, пропонувати продукт, що може зацікавити саме цього споживача.

- Система планування ресурсів підприємства (ERP-система, Enterprise Resource Planning) – корпоративна інформаційна система, призначена для автоматизації обліку та керування [12, 15]. Як правило, вона будується за модульним принципом та практично охоплює всі ключові процеси діяльності

бізнес-проекту та дозволяє керування фінансами, виробництвом, формуванням та розподілом запасів, реалізацією та маркетингом; утриманням покупців тощо.

ERP-система дозволяє реалізувати комплексне управління підприємством через смартфон або планшет навіть без мобільного застосунку. Такі рішення дозволяють впроваджувати нові бізнес-моделі, приймати оперативні рішення щодо бізнесу, контролювати фінанси тощо. Їх можна інтегрувати зі складними технологіями, зокрема машинне навчання та data-аналітики.

- Корпоративний портал – це тип веб-застосунку, який служить централізованою платформою для доступу до різноманітних організаційних ресурсів і послуг. Він має такі особливості: настроюваний інтерфейс користувача; аутентифікацію користувача та контроль доступу; інтеграцію з різними корпоративними системами; система управління контентом; параметри персоналізації та налаштування тощо. Вони дозволяють працювати одразу з декількома застосунками компанії (CRM, ERP, поштою, чатом тощо), використовуючи один інтерфейс. Кожен працівник має окремий доступ, тому має пройти аутентифікацію [14].

1.3 Огляд провідних вітчизняних та закордонних існуючих ERP-систем

Для оптимізації бізнес-процесів, покращення управління ресурсами та забезпечення ефективної е-комерції платформи інтегруються з ERP-системою, що представляють собою комплексні програмні рішення, які допомагають бізнес-проектам упорядковувати та оптимізувати свої операції, сприяючи підвищенню ефективності та збільшенню прибутковості.

Модульна структура ERP-систем дозволяє об’єднати модулі відповідно до потреб підприємства, що дає можливість підприємствам автоматизувати процеси замовлення, оплати, доставки та обробки замовлень через онлайн-канали. Інтеграція ERP-системи з модулями керування складськими запасами та логістикою дозволяє підприємствам ефективно виконувати замовлення з е-комерції, забезпечувати швидку доставку товарів та контролювати запаси.

У даному підрозділі будуть розглянуті ведучі ERP-системи, розроблені в Україні та за кордоном для проведення аналізу над існуючими рішеннями,

1.5.1 IT-Enterprise

IT-Enterprise є однією з провідних компаній в Україні, яка спеціалізується на розробці та впровадженні ERP-систем. Компанія IT-Enterprise більше 35 років займається впровадженням IT-рішень для автоматизації бізнес-процесів на підприємствах різних напрямків діяльності.

Клієнтами IT-Enterprise є найбільші підприємства України; у приватному секторі: Interpipe, Ferrexpo, ROSHEN, FED, МХП, Укрнафта, Південмаш, KNESS та багато ін.; у державному секторі: ДП "Антонов", ДП "ХМЗ "ФЕД", Укроборонпром, Міністерство інфраструктури, Національна академія аграрних наук, КМУ та багато ін. [15].

Флагманським продуктом компанії являється програма IT-Enterprise – потужний інструмент для реінжинірингу й оптимізації бізнес-процесів. Це повнофункціональна ERP-система, що охоплює такі системи: MRP II, MES, APS, EAM, SCM, CRM [15].

- MRP II (Manufacturing Resource Planning II) – підхід до планування виробництва та управління запасами, який допомагає компаніям-виробникам оптимізувати виробничі процеси, ефективно керувати ресурсами та планувати потреби в матеріалах і потужностях.
- MES (Manufacturing Execution System) – це програмна система, яка контролює та координує виробничі операції в реальному часі.
- APS (Advanced Planning and Scheduling) – набір програмних засобів і методів, які використовуються для оптимізації планування виробництва, планування та розподілу ресурсів.
- EAM (Enterprise Asset Management) – це система для управління фізичними активами організації, включає відстеження активів,

планування технічного обслуговування, керування робочими замовленнями, моніторинг продуктивності обладнання тощо.

- SCM (Supply Chain Management) – це система, що передбачає планування, координацію та контроль потоку товарів, послуг та інформації по всьому ланцюгу постачання: від закупівлі сировини до доставки продукції.
- CRM (Customer Relationship Management) – набір стратегій, практик і методів для управління та аналізу взаємодій і відносин з клієнтами. Допомагають компаніям оптимізувати процеси продажів, маркетингу та обслуговування клієнтів тощо.

Архітектура IT-Enterprise ERP-системи базується на модульній структурі, де кожен модуль відповідає за певні функції бізнесу. Це дозволяє підприємствам вибирати та впроваджувати тільки необхідні модулі, що робить систему більш гнучкою до конкретних потреб кожного клієнта.

Основні риси IT-Enterprise ERP-системи:

- Забезпечує інтеграцію окремих підрозділів підприємства, для забезпечення повного управління бізнес-процесами.
- Модульність, завдяки якої клієнти можуть вибрати необхідні модулі залежно від своїх потреб та розширити функціонал системи за необхідністю.
- Надає ефективні інструменти для аналізу та звітності.
- Мобільний доступ до системи через мобільні пристрої, що дозволяє управляти бізнесом з будь-якого місця та в будь-який час.

1.5.2 Oracle E-Business Suite

Oracle E-Business Suite (EBS) є однією з найвідоміших та широко використовуваних систем ERP. Проект розроблений компанією Oracle у 2001 р., але необхідно зазначити, що EBS базується на версії Oracle Applications, яка ще розроблялася у 80-х роках XX ст. [13, 16, 17].

EBS є гнучкою системою, яку можна адаптувати та налаштувати під потреби та вимоги конкретного підприємства, так як пропонує широкий вибір модулів і функцій. Крім того, EBS підтримує інтеграцію з іншими системами та застосунками для реалізації синхронізації даних та обміну інформацією між різними джерелами даних. EBS складається з наступних модулів [16, 17] :

- Oracle Database: основна БД, де зберігаються всі дані, пов'язані з бізнес-процесами, клієнтами, товарами, фінансами та іншими аспектами.
- Oracle Application Server: веб-сервер, який надає інфраструктуру для роботи веб-застосунка, включаючи обробку запитів, керування сеансами користувачів та інтеграцію з іншими системами.
- Oracle Forms: інструмент для розробки користувацьких інтерфейсів, що використовуються для взаємодії з даними та виконання бізнес-операцій.
- Oracle Reports: інструмент для створення звітів та друку документів на основі даних з БД.
- Oracle Workflow: механізм для автоматизації бізнес-процесів і керування потоками роботи в системі.
- Oracle Discoverer: інструмент для створення інтерактивних звітів та аналітичних даних для користувачів.

1.5.3 Odoo

Odoo – ERP система, розроблена бельгійською компанією Odoo S.A; надає інтегрований набір модулів для автоматизації різних бізнес-процесів; забезпечує цілісний підхід до управління ресурсами підприємства та зростання його ефективності. Крупними міжнародними клієнтами Odoo являються Toyota, Hyundai, Danone, Suzuki тощо; в Україні – корпорація «АТБ» [18, 19].

Структура Odoo базується на модульному рішенні та складається з таких компонентів [18]:

- Модулі: функціонал поділений на окремі модулі: облік, продажі, закупівлі, складське господарство, фінанси тощо; кожен модуль відповідає за конкретну функціональну область системи.
- Об'єкти (Records): дані зберігаються у вигляді об'єктів (клієнти, товари, замовлення); кожен об'єкт має свої атрибути та відносини з іншими об'єктами.
- Представлення (Views): веб-інтерфейс заснований на представленнях, які відображають дані та функціональні елементи. Вони використовують XML для визначення структури та вигляду сторінок.
- Бізнес-логіка: реалізується за допомогою Python; розробники можуть використовувати Python для створення логіки обробки даних, валідації, автоматизації бізнес-процесів та інших завдань.

Серед основних переваг розробники Odoo та користувачі відмічають такі властивості та характеристики [18]:

- Відкритий код дозволяє залучати розробників для побудови екосистеми повністю інтегрованих бізнес-застосунків.
- Інтегрована система ERP: містить комплексний набір функцій для управління різними бізнес-процесами: продажі, закупівлі тощо.
- Модульний підхід дозволяє користувачам вибирати та налаштовувати тільки необхідні функціональні модулі.
- Odoo може бути легко інтегрований з іншими застосунками та системами, такими як електронна пошта, платіжні шлюзи, е- магазини тощо.

Таким чином, в результаті опрацювання наукових джерел інформації у першій частині роботи визначені основні поняття сучасних веб-застосунків, їх технічна характеристика, надані переваги сучасних веб-застосунків у порівнянні з ПК. На підставі аналізу літературних джерел наданий загальний аналіз рівневої архітектури веб-застосунку та приведена їх класифікація відповідно до сфери призначення. Також проведений огляд провідних вітчизняних та закордонних існуючих ERP-систем, які дають можливість

ефективно виконувати замовлення з е-комерції, забезпечувати швидку доставку товарів та контролювати запаси, що оптимізує бізнес-процеси та удосконалює управління та облік продукції.

РОЗДІЛ 2 АНАЛІЗ ПРОГРАМНО-ТЕХНІЧНИХ РІШЕНЬ ДЛЯ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКІВ

2.1 Аналіз архітектурних підходів організації програмного забезпечення і взаємодія компонентів

Веб-застосунок являється незалежним від платформи, доступ до якого здійснюється через веб-браузер, та який використовує архітектуру клієнт-сервер. *Клієнт* – комп'ютер, смартфон або будь-який інший пристрій на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій. *Сервер* – це комп'ютер, призначений для вирішення певних завдань з виконання програмних кодів, сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації та баз даних [3-9].

Функції, які реалізуються на сервері:

- зберігання, доступ, захист і резервне копіювання даних;
- обробка клієнтського запиту;
- відправлення результату клієнту.

Функції, які реалізуються на стороні клієнта:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправка;
- отримання результатів запиту і подальше їх опрацювання [3-9].

Архітектура веб-застосунків – це механізм, який визначає, як компоненти застосунків взаємодіють один з одним. Веб-застосунки різних розмірів і рівнів складності мають однаковий архітектурний принцип, але можуть відрізнятися деталями [3-9].

Як правило, архітектура веб-застосунків складається з трьох основних компонентів [3, 5]:

1) Веб-браузер є ключовим компонентом, який взаємодіє з користувачем, отримує вхідні дані та керує логікою презентації, одночасно контролюючі взаємодію користувача з програмою; за потреби також перевіряються введені користувачем дані.

2) Веб-сервер керує бізнес-логікою та обробляє запити користувачів, направляючи їх до потрібного компонента та керуючи всіма операціями програми; може отримувати та контролювати запити від широкого кола клієнтів.

3) Сервер БД надає необхідні дані для програми. У багаторівневій архітектурі сервери баз даних можуть керувати бізнес-логікою за допомогою збережених процедур.

Дворівнева архітектура складається з двох компонентів:

- сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси;
- клієнт, який представляє користувацький інтерфейс.

Принцип роботи такої архітектури полягає в тому, що сервер отримує запит, обробляє його і відповідає безпосередньо без використання сторонніх ресурсів. Її недоліком являться той факт, що зі збільшенням кількості користувачів знижується продуктивність, також пряма взаємодія БД і пристрою користувача може викликати деякі проблеми безпеки [3, 5, 7 - 9].

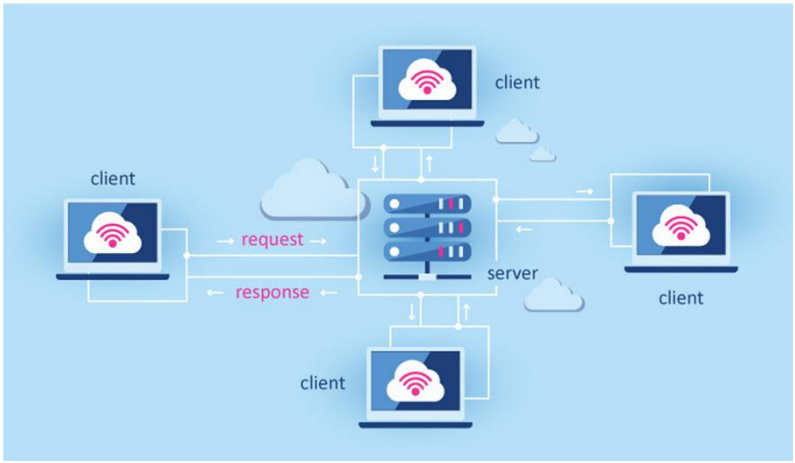

Рисунок 2.1 – Принцип роботи трирівневої архітектури

Трирівнева архітектура складається з таких компонентів рис. 2.1:

- *Presentation tier* / рівень клієнта – призначений для користувача інтерфейс;
- *Logic tier* / бізнес-рівень – сервер застосунків;
- *Data tier* / База даних – сервер БД, який надає інформацію.

Принцип роботи такої архітектури полягає в тому, що проміжні сервери отримують запити клієнтів і обробляють їх, координуючи роботу з підлеглими серверами, застосовуючи бізнес-логіку. Такий розподіл операцій знижує навантаження на сервер. Зв'язок між клієнтом і БД здійснюється проміжним прикладним рівнем, що дозволяє клієнтам отримувати доступ до даних згідно з Database Management System (DBMS) [6 - 9].

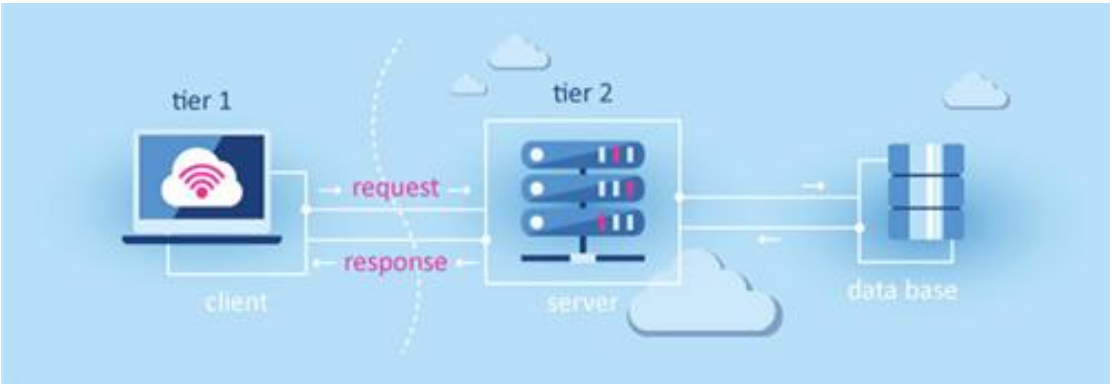

Рисунок 2.2 – Принцип роботи трирівневої архітектури

Трирівнева архітектура має такі переваги:

- *масштабованість та продуктивність*, які забезпечуються можливістю розгортання серверів застосунків на декількох машинах;
- *конфігурованість:* ізольованість рівнів один від одного дозволяє швидко і простими засобами зробити переконфігурацію системи, якщо виникли збої, або при плановому обслуговуванні на одному з рівнів;
- *високий рівень безпеки*, оскільки клієнт не має прямого доступу до даних;
- *висока надійність;*
- *низькі вимоги до швидкості каналу.*

Веб-застосунок працює за такою узагальнюючою схемою [6 - 9]:

- Користувач надсилає запит до веб-серверу, що складається з центру керування застосунком та БД.
- Сервер отримує запит, обробляє його на валідність й скеровує його до БД.
- Інформація за запитом користувача вибирається з БД й передається у сервер.
- Сервер відправляє інформацію отримувачу, яка далі обробляється чи відображається на екрані користувача.

Таким чином, переваги трирівневої архітектури у порівнянні з дворівневою включають розділення обов'язків, масштабованість, повторне використання коду. Така архітектура дозволяє краще організувати розробку та підтримку веб-застосунків, забезпечує гнучкість при масштабуванні та забезпечує безпеку та цілісність даних.

2.2 Аналіз та порівняльна характеристика архітектурних рішень у веб-розробці застосунків.

2.2.1 Односторінкові застосунки

Односторінкові застосунки (Single Page Applications, SPA) – це тип веб-програм, у яких весь необхідний зміст, шаблони, скрипти завантажується разом з першим запитом до застосунку. SPA працюють на одній HTML-сторінці, при цьому застосунок динамічно оновлюється без необхідності повного перезавантаження сторінки. Автоматичне оновлення відбувається за допомогою JavaScript, тому потреба у перезавантаженні сторінки відпадає [21-24].

При завантаженні сценарію сторінки зміст його майже порожній. У SPA прийнято відокремлювати зовнішні ресурси від серверного сервера, що обслуговує файл index.html. Це розділення дозволяє створювати роз’єднану архітектуру, де зовнішні ресурси можуть обслуговуватися з іншого сервера або CDN, тоді як внутрішні сервери зосереджуються, в основному, на обслуговуванні даних і API.

Метою обслуговування зовнішніх ресурсів з окремого статичного сервера підвищує продуктивність і оптимізацію. Основними перевагами є ефективне кешування, паралельне завантаження та оптимізація доставки вмісту, обслуговуючи статичні ресурси з іншого сервера або CDN (це розподілена мережа серверів, розташованих у різних географічних точках по всьому світу). Це покращує загальну взаємодію з користувачем за рахунок хешування HTML-сторінки, тому під час роботи SPA запитуються лише серверні дані [21-24] .

Також порівняно з більш традиційними програмами на стороні сервера, односторінкову програму дуже легко розгорнути, оскільки це лише файл index.html із набором CSS і пакетом Javascript. Ці три статичні файли можна завантажити на будь-який сервер статичного вмісту, наприклад Apache, Nginx, Amazon S3 або Firebase Hosting [23]. Узагальнюючи вищесказане, основні переваги та недоліки SPA приведені у табл. 2.1 [21-25].

Таблиця 2.1 – Основні переваги та недоліки SPA архітектури

Переваги	Недоліки
Висока швидкість UX: усі елементи завантажується з першого разу, під час виконання різних дій на сторінці дані просто змінюються.	Додаткове навантаження на браузер: якщо сторінка виявиться «важкою», старі та слабкі пристрої можуть «зависати»
Економія часу: практично вся робота відбувається у браузері, без звернень до сервера.	Необхідність постійного інтернет-з’єднання, без якого використовувати SPA практично неможливо
Просте обслуговування завдяки однієї сторінки HTML, розробка інтерфейсу та серверної частини розділені, що полегшує керування та оновлення програми	

Достатньо висока продуктивність за рахунок оптимізації коду та зменшення кількості запитів до сервера	Обмежене SEO: проблеми з оптимізацією для пошукових систем

Кінець таблиці 2.1

Швидке кешування: SPA відправляє лише один запит, збирає дані, після чого може працювати офлайн	Проблеми із безпекою: є більш вразливими для атак (XSS)

Використання SPA: їх гнучкість та здатність до масштабування робить їх ефективними для створення складних інтерфейсів та багаторівневих застосунків, таких як приватної частини інформаційної панелі платформ SaaS або Інтернет-сервісів загалом, а також для створення корпоративних застосунків, застосунків для керування даними та інтенсивних форм [21-25].

Фреймворки і бібліотеки, які реалізують SPA технологію, є Angular, React JS, Vue.js і Ember.js [21-25].

2.2.2 Багатосторінкові веб-застосунки

Багатосторінкові веб-застосунки (Multi-Page Web Application, MPA) працюють за класичною схемою, тобто кожна взаємодія користувача викликає перезавантаження сторінки з сервера. У MPA кожна сторінка представляє окремий HTML-документ із власним вмістом і розміткою [21].

Основна відмінність між MPA та SPA полягає в тому, що в MPA кожна взаємодія користувача робить повний запит на сервер, і сервер повертає нову сторінку. Це означає, що в MPA кожна сторінка має власні сценарії, стилі та інші ресурси, і вони перезавантажуються кожного разу, коли користувач переходить між сторінками.

MPA можна використовувати для розробки багатьох типів веб-застосунків, включаючи сайти е-комерції, блоги, портали новин, соціальні мережі тощо. У MPA кожна сторінка може мати власну логіку та функціональність, що забезпечує більшу гнучкість для веб-розробки [21].

Фреймворки і бібліотеки, які реалізовують MPA технологію є: Django – високорівневий веб-фреймворк на мові Python, використовує загально узгоджені правила, як і у Django, Ruby on Rails має чітко визначені конвенції, що зменшує гнучкість проекту і його подальшу маштабованість, ASP.NET – є фреймворком на мові C#, базується на серверній технології, надає готові компоненти та контролери, має вбудовану систему управління станом, Express.js має велику бібліотеку готових компонентів, легке додавання сторонніх бібліотек для розширення функціоналу веб-застосунку. Аналіз наукових джерел дозволив визначені основні переваги та недоліки MPA, які приведені у табл. 2.2 [21].

Таблиця 2.2 – Переваги та недоліки MPA

Переваги	Недоліки
Зручність для SEO: кожна сторінка має свої метадані, що полегшує пошуковим системам сканування та індексування по запитам користувачам.	Повільніші продуктивність і час відповіді, оскільки кожна сторінка потребує окремого запиту до сервера, що може збільшити час завантаження та погіршити взаємодію з користувачем
Сумісність з усіма браузерами та пристроями	Обмежена інтерактивність та оновлення в реальному часі
Забезпечення стандартного механізму оновлення сторінки	Складність розробки: потребує більше часу, тому є фінансово затратною

Можливість використання готових рішень, таких як OpenCart та WordPress.	

2.2.3 Прогресивні веб-застосунки

Прогресивні веб-застосунки (Progressive Web App, PWA) – це веб-програми, які поєднують переваги веб-сторінки та мобільного застосунка. Доступ до PWA можна отримати через веб-браузер на десктопі чи смартфоні. PWA зберігає чимало функцій, типових для нативного мобільного застосунку; візуально макет веб-застосунку практично не відрізняється від мобільної версії. Визначають, що функціонал мобільного застосунку зменшений та містить тільки основний функціонал, щоб не перевантажувати систему. Основні переваги та недоліки PWA приведені у табл. 2.3 [21, 26].

Таблиця 2.3 – Основні переваги та недоліки PWA

Переваги	Недоліки
Може працювати в автономному режимі або з мережами низької якості, кешуючі необхідні дані, що забезпечує безперебійну роботу користувача та зменшує потребу в підключенні до мережі	Має обмежений доступ до апаратного забезпечення пристрою та API, наприклад до камери чи датчиків, що може обмежити функціональність програми.
Може працювати одразу з кількома ОС на будь-якому пристрої	Firefox, Edge та деякі інші браузери погано працюють з PWA

Швидка розробка: для створення не треба робити окремий сайт, достатньо додати новий модуль для вже наявного	Мають обмежену функціональність на пристроях iOS через політику Apple щодо веб-застосунків
Автоматичне оновлення	Обмежене впровадження, так як є відносно новою технологією

2.3 Моделі керування доступом

Керування доступом являється основним елементом кібербезпеки, що визначає, хто та за яких умов може отримувати доступ до певних даних, програм і ресурсів. Політика керування доступом забезпечує захист цифрового середовища та, як правило, ґрунтується на таких методах, як автентифікація і авторизація, що дають організаціям змогу перевіряти правдивість даних користувачів і їхнє право на доступ до певних ресурсів на основі пристроїв, розташування, ролей тощо [27]. Керування доступом не дає можливості зловмисникам та іншим неавторизованим користувачам викрасти делікатну інформацію, таку як клієнтські дані або інтелектуальну власність; зменшує ризик екс фільтрації даних працівниками й забезпечує захист від веб-загроз. Більшість організацій із високим рівнем захисту використовують рішення для керування ідентичностями й доступом, щоб впровадити відповідні політики.

Основними аспектами використання архітектурних підходів до керуванням доступу є наступні [27]:

- Лише авторизовані користувачі можуть отримати доступ до конфіденційної інформації, що захищає важливі дані від розголошення та забезпечує конфіденційність.
- Лише авторизовані особи можуть змінювати дані у системі, таким чином зберігаючи та захищаючи її незмінність і точність.

- Користувачі отримують ідентифікацію та автентифікацію, що обмежує їхній доступ до вказаних ресурсів. Це дає змогу запобігти несанкціонованому доступу, зокрема хакерським атакам, несанкціонованому внутрішньому доступу тощо.

2.3.1 Вибіркове керування доступом (DAC)

DAC – це модель безпеки, яка надає або обмежує доступ до ресурсів на основі ідентичності та дозволів окремих користувачів або груп. Тобто, у моделях DAC кожен об’єкт у захищеній системі має власника, який надає доступ до нього користувачам. DAC забезпечує індивідуальний підхід до керування ресурсами. Основними особливостями моделі DAC являються [29]:

- *Ідентифікація окремих користувачів* і забезпечення контролю над доступом до ресурсів: кожен користувач пов’язаний із певними дозволами або правами доступу, які визначають дії, які він може виконувати з ресурсами.
- DAC *дозволяє адміністраторам визначати права доступу* на рівні ресурсу, які надаються або скасовуються окремо для кожного користувача чи групи.
- *Різні рівні доступу* дозволяють використовувати *різні рівні контролю та обмежень* залежно від потреб і обов’язків різних користувачів.
- DAC *надає дозвіл адміністраторам керувати* обліковими записами користувачів, визначати права доступу та змінювати їх за потреби.
- *Детальний контроль над доступом* до ресурсів для окремих користувачів або груп гарантує, що користувачі мають доступ лише до ресурсів, необхідних для виконання своїх завдань, зменшуючи ризик несанкціонованого доступу або витоку даних.
- *Гнучкість керування дозволами доступу*: адміністратори можуть легко змінювати дозволи для певних користувачів або груп за потреби, наприклад при зміні статусу користувачів, їх обов’язків або

організаційних вимогах тощо. Також, адміністратори можуть відстежувати дії користувачів, переглядати журнали доступу та виявляти будь-які несанкціоновані або підозрілі дії.

До недоліків та складнощів при роботі з моделлю DAC можна віднести наступні доводи [29]:

- *Невідповідний розподіл обов'язків*: DAC не забезпечує чіткого розподілу обов'язків, тобто користувач із вищими привілеями потенційно може отримувати доступ до конфіденційних ресурсів, які йому не потрібні для виконання основних обов'язків, і змінювати їх. Це може збільшити ризик несанкціонованого доступу, випадкової зміни даних або навмисного зловживання привілеями.
- *Обмежена масштабованість*: зі збільшенням числа користувачів і ресурсів керування дозволами доступу на індивідуальному рівні може ускладнитися.
- *Зменшений централізований контроль над дозволами доступу*: окремі користувачі або власники ресурсів мають право керувати власними дозволами, що ускладнить адміністраторам застосування узгодженої політики безпеки та моніторинг дій доступу в системі.

DAC широко використовується в різних веб-застосунках, зокрема онлайн-хмарних сервісах (Dropbox, Google Drive та Microsoft OneDrive) для керування доступом до файлів та папок; соціальних мережах (Facebook, Twitter та Instagram) для керування доступом до профілів користувачів та їхніх даних; онлайн-магазинах: веб-застосунках для е-комерції тощо.

2.3.2 Обов'язкове керування доступом (MAC)

MAC забезпечує структурований і централізований підхід до контролю доступу: центральний орган регулює права доступу й упорядковує їх за рівнями, які рівномірно розгортаються в усій системі. MAC також відома як модель не дискреційного контролю, тобто керування не надається на розсуд

користувача чи власника; механізми контролю MAC дотримуються принципів нульової довіри. Основні характеристики MAC приведені нижче [27, 29]:

- MAC призначає мітки безпеки як суб'єктам (користувачам), так і об'єктам (ресурсам) з класифікацією та категорією. Мітки безпеки містять інформацію про рівні безпеки, категорії, які використовуються для застосування політик контролю доступу.
- *Централізоване застосування* політики контролю доступу на основі попередньо визначених правил у системі.
- *Ієрархічна модель рівнів безпеки*: суб'єкти з вищими рівнями безпеки можуть отримати доступ до об'єктів із нижчими або однаковими рівнями безпеки. Така сувора ієрархія гарантує захист конфіденційної інформації та запобігає несанкціонованому доступу.
- *Принцип найменших привілеїв*: суб'єктам надаються лише мінімальні привілеї, необхідні для виконання їхніх завдань; такий принцип мінімізує ризик несанкціонованого доступу, випадкового порушення даних і можливого зловживання привілеями.
- *Забезпечення конфіденційності та цілісності даних* шляхом запобігання несанкціонованому доступу до конфіденційної інформації та контролю за доступом для запису гарантує, що лише авторизовані суб'єкти можуть змінювати дані.

MAC забезпечує надійну безпеку та контроль над доступом до ресурсів, але при роботі з цією моделлю можуть виникати деякі труднощі [29]:

- Впровадження та керування системою MAC є *складним і ресурсозатратним*, оскільки вимагає ретельного планування, визначення міток безпеки та призначення відповідних міток суб'єктам і об'єктам. Крім того, постійне обслуговування MAC потребує кваліфікованого персоналу.

- *Обмежена гнучкість*: системи MAC мають жорстку централізовану політику контролю доступу. Це може обмежити гнучкість швидкої адаптації до нових змінених вимог доступу.
- *Ризик неправильної класифікації та маркування* суб’єктів і об’єктів через людську помилку.

Завдяки своїм надійним характеристикам щодо безпеки даних ця модель використовується в державних, військових, банківських системах, на підприємствах критичної інфраструктури тощо.

2.3.3 Керування доступом на основі ролей (RBAC)

У моделях RBAC користувачам надається доступ залежно від їхніх функцій в організації. Мета RBAC полягає в тому, щоб надавати користувачам доступ лише до тих даних, які їм потрібні для виконання робочих завдань [29, 31].

Модель RBAC має наступні характеристики [29, 31]:

- *Дозволи на основі ролей*: RBAC організовує користувачів на основі ролей: їхніх обов’язків, посадових функцій в організації тощо; дозволи надаються ролям. Користувачам призначають конкретні ролі та надають дозволи, пов’язані з цими ролями.
- *Ієрархічна структура ролей* дає можливість визначати ролі високого рівня, яким надаються розширені набори дозволів, і ролі нижчого рівня. Ієрархічна структура забезпечує гнучкість керування дозволами та ефективне призначення ролей.
- *Спрощене адміністрування*, оскільки керування доступом відбувається на рівні ролі, а не на рівні окремого користувача.
- *Принцип найменших привілеїв* гарантує, що користувачам надаються лише дозволи, необхідні для виконання їхніх робочих функцій. Це мінімізує ризик несанкціонованого доступу та можливого зловживання привілеями.

- *Висока масштабованість* дозволяє використовувати RBAC для великих організацій із складними вимогами до контролю доступу. Коли нові користувачі приєднуються до організації або ролі змінюються, RBAC дозволяє ефективно призначати ролі та керувати дозволами.
- *Розподіл обов'язків* гарантує, що користувач не матиме надмірних привілеїв, які можуть призвести до шахрайства чи несанкціонованої діяльності.
- *Аудит і відповідність*: RBAC зобов'язує журнал аудиту, в якому відображаються дозволи та дії з конкретними ролями. RBAC також надає можливість стороннім організаціям показати контроль доступу та розподіл обов'язків відповідно до посадових інструкцій та нормативних вимог чинного законодавства.
- *Гнучкість і адаптивність*: RBAC дозволяє легко модифікувати та змінювати дозволи доступу відповідно до потреб організації.

Незважаючи на свої численні переваги щодо забезпечення безпеки даних, RBAC має потенційні недоліки, які організації повинні враховувати при виборі цієї моделі [31]:

- *Складність впровадження у великих і складно організованих колективах*, оскільки вимагає ретельного планування та координації з визначенням ролей та пов'язаних з ними дозволів. Труднощі також можуть виникнути, якщо розробляються дуже детальні ролі та різко збільшується кількість ролей; у випадках, коли з'являються винятки для дозволів призначеної ролі.
- RBAC, в основному, розроблено *для статичних середовищ контролю доступу*, тому така модель може зазнавати труднощі зі сценаріями динамічного контролю доступу, коли вимоги до доступу часто змінюються.

Така модель має широке впровадження в різних бізнес-проектах: корпораціях, соціальних мережах тощо.

2.3.4 Керування доступом на основі атрибутів (ABAC)

У моделях ABAC користувачам надається доступ на основі атрибутів і умов середовища. ABAC – це найбільш деталізована модель керування доступом, яка допомагає зменшити кількість призначень ролей [29, 30]. До основних параметрів та властивостей ABAC відносяться такі [29, 30]:

- *Детальне керування доступом*: враховуються численні атрибути, пов'язані з користувачем, ресурсом, середовищем і контекстом, такі як ролі користувачів, посади, місця розташування, час доступу тощо.
- ABAC адаптований для сценаріїв *динамічного* контролю доступу, коли вимоги до доступу часто змінюються. Атрибути можна оновлювати в режимі реального часу.
- *Політика на основі атрибутів*, яка виражається у формі правил або умов, що оцінюють *атрибути, пов'язані з користувачем, ресурсом і середовищем*. Така політика реалізується за допомогою мови політики або графічного інтерфейсу, що дозволяє легко налаштовувати та керувати складними правилами доступу.
- *Оцінка атрибутів* під час виконання для прийняття рішення щодо контролю доступу. ABAC оцінює не тільки *атрибути, пов'язані з користувачем, але й атрибути, пов'язані з ресурс*ом, до якого здійснюється доступ, і *середовищем,* у якому здійснюється доступ.
- *Висока масштабованість* підходить для складних і великомасштабних середовищ, оскільки ABAC може обробляти велику кількість атрибутів, що дає можливість задовольняти різні вимоги контролю доступу в різних системах і програмах в межах організації.
- *Повторне використання політики контролю* доступу в кількох системах і програмах спрощує керування, зменшує дублювання та сприяє узгодженості контролю доступу в усій організації.

- *Відповідність до нормативних вимог і аудит.* Модель ABAC сприяє проведенню аудиту, оскільки атрибути, пов’язані з кожним запитом на доступ, можна реєструвати та аналізувати.

До недоліків моделі ABAC можна віднести такі показники [29, 30]:

- *Складність впровадження* особливо у великих і складно організованих середовищах.
- З*більшення адміністративних та фінансових витрат* через складну організацію та політику моделі.
- *Проблеми керування атрибутами*: забезпечення якості, послідовності та своєчасності атрибутивних даних, особливо при інтеграції з декількома системами.

Таким чином, при виборі відповідної моделі керування доступом необхідно враховувати конкретні потреби бізнес-проекту та його економічні та матеріальні можливості.

2.5 Патерни проектування програмного забезпечення

Паттерн (шаблон) проектування – це іменований опис проблеми та її розв’язання, яке можна використовувати у розробці інших систем [64]. Кожен патерн має свою специфіку та використовується відповідно до конкретних потреб та вимог проекту.

Розглянемо основні властивості та характеристики таких популярних патернів проектування, як MVC (Model-View-Controller) та MVVM (Model-View-ViewModel), які використовуються в розробці веб-застосунків для оптимізації та підвищення ефективності роботи програмістів.

MVC є архітектурним патерном, який використовується для розділення логіки програми на три основні компоненти : Контролер (Controller), Модель (Model), Model (модель), Представлення (View) [64]. Архітектура MVC представлена на рисунку 2.2.

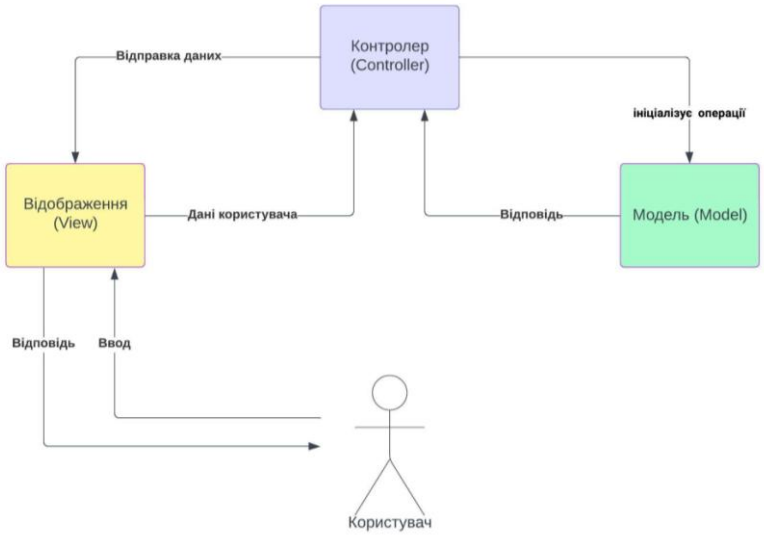

Рисунок 2.2 – Архітектура MVC

Основні функції кожного компонента MVC наведені нижче [64] :

- *Контролер*: відповідає за обробку вхідних запитів від користувачів, взаємодію з моделями та передачу відповідних даних у представлення; приймає введення від користувача, ініціалізує відповідні операції в моделі та вибирає відповідне подання для відображення результатів. Контролер діє як посередник між моделлю та представленням, але не містить бізнес-логіку або обробку даних. Він координує роботу моделі та представлення, забезпечуючи правильну обробку запитів і передачу даних між ними.
- *Модель* представляє логіку програми та контролює доступ до даних; вона містить бізнес-логіку, правила перевірки та операції для отримання, збереження та оновлення даних. Моделі можуть бути представлені у вигляді класів або об'єктів, які представляють сутності або концепції програми.
- *Представлення* відповідає за візуальне відображення даних моделі та взаємодію з користувачем.

Використання MVC надає такі переваги при проектуванні веб-застосунків, як [64]:

- дозволяє розробляти та підтримувати компоненти окремо, що полегшує масштабованість і обслуговування коду; внесення змін або оновлень одного компонента не впливає на інші.
- сприяє багаторазовому використанню коду, відокремлюючи бізнес-логіку від інтерфейсу користувача і взаємодії з користувачем. Таке розділення дозволяє повторно використовувати моделі та контролери в різних представленнях, що підвищує ефективність розробки.
- модель, представлення та контролер можна тестувати незалежно; такий підхід полегшує написання програми для кожного компонента та дає можливість впевнитися, що програма функціонує правильно та відповідає вибраним вимогам.
- завдяки чіткому розподілу між моделлю, представленням і контролером MVC дає можливість у виборі технології та платформи; розробникам можна вибирати різні фреймворки, бібліотеки або технології для кожного компонента відповідно до своїх задач.
- можливість відокремити модель і контролер, дозволяє розробити інтуїтивно зрозумілий та зручний інтерфейс, який покращує взаємодію з користувачем.
- MVC може реалізовувати заходи безпеки та контроль доступу на рівні контролера, який може обробляти логіку автентифікації, авторизації та перевірки, гарантуючи, що лише авторизовані користувачі мають доступ до необхідних функцій і даних у програмі.

MVVM – архітектурний патерн, який використовується для розділення логіки програми на три основні компоненти: Модель (Model), Представлення, (View) та Модель представлення (ViewModel) [43, 50, 53]. Основні складові архітектури MVVM та їх функції представлені далі [43, 50, 53, 67]:

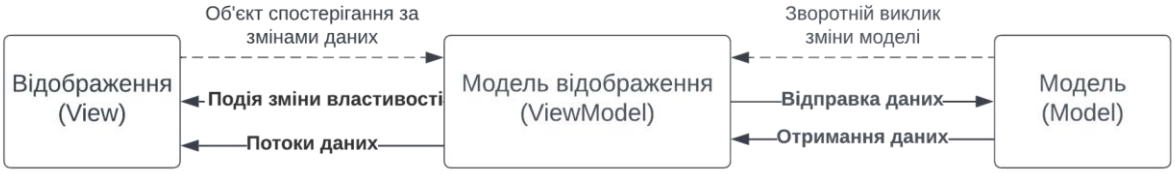

Рисунок 2.3 – Архітектура MVVM

- *Модель* представляє дані та бізнес-логіку застосунку. Вона відповідає за збереження та обробку даних без прив'язки до представлення або користувацького інтерфейсу.
- *Представлення* відповідає за візуальне відображення даних та взаємодію з користувачем.
- *Модель представлення* слугує посередником між моделлю та представленням, забезпечуючи зв'язок між даними моделі та представленням, а також містить логіку, яка впливає на відображення даних та взаємодію з користувачем. Вона дозволяє розділити відображення даних від бізнес-логіки та спрощує тестування.

Використання MVVM надає такі переваги та зручні функції при проектуванні [43, 50, 53]:

- Розділення логіки та відображення, що полегшує розробку, тестування та підтримку програмного коду.
- Можливість застосування декларативного підходу до опису взаємодії та відображення даних, що робить код більш зрозумілим та підтримуваним.
- Легка можливість тестувати логіку без прив'язки до представлення або користувацького інтерфейсу, що дає можливість швидко виявляти та виправляти помилки.
- Можливість легко масштабувати та повторно використовувати компоненти MVVM в інших частинах програми та в інших проектах.

На підставі аналізу наукових джерел були узагальнені особливості MVC та MVVM, які представлені у табл. 2.4 [43, 50, 53, 64, 67]. Необхідно уточнити, що реалізація MVC та MVVM може варіювати в залежності від конкретних

фреймворків та мов програмування; таблиця надає лише загальне порівняння, яке може змінюватися залежно від конкретного контексту та вимог веб-застосунку.

Таблиця 2.4 – Основні відмінності параметрів між MVC та MVVM

Характеристика	MVC	MVVM
Архітектура	Модель-Вид-Контролер	Модель-Вид-Модель представлення
Основна мета	Розділення обов'язків та модульний дизайн	Розділення обов'язків, зв'язування даних та можливість тестування
Потік даних	Односторонній	Двосторонній

Кінець таблиці 2.4

Взаємодія з Видом	Пряма взаємодія з Моделлю та Контролером	Взаємодія через зв'язування даних та команди
Оновлення Виду	Ручне запускання Контролером	Автоматичне оновлення через зв'язування даних
Тестування	Контролери та Моделі можна тестувати незалежно	Модель представлення можна легко тестувати за допомогою зв'язування даних
Зв'язування даних	Неінтегроване, вимагає додаткової реалізації	Інтегроване, дозволяє автоматичну синхронізацію між Видом та Моделлю представлення
Складність коду	Може ускладнювати код	Сприяє створенню чистого та більш зручного для підтримки коду

Гнучкість при виборі компонентів	Більша	Менша гнучкість через строге розділення обов'язків

Таким чином, MVC і MVVM являється архітектурними структурами, головна відмінність яких полягає в тому, що перший патерн відокремлює веб-застосунок на три основні логічні компоненти: Модель, Перегляд і Контролер, тоді як MVVM поділяє веб-застосунок на компоненти як Модель, Представлення та Модель представлення.

2.5 Використання технологій та сервісів для веб-розробки

2.5.1 Використання хмарних сервісів для створення веб-застосунків

З появою хмарних обчислень для розробників і користувачів з’являються нові можливості для зручного й ефективного використання ресурсів, зберігання даних і розгортання веб-застосунків. Ключовими перевагами використання хмарних обчислень є [32-36] :

- *Надлишкова потужність і висока доступність* за рахунок наявності, як правило, декілька центрів обробки даних, що дає можливість при екстрених ситуаціях, автоматично переключати робоче навантаження на інший центр обробки даних або сервер.
- *Висока масштабованість та продуктивність* за рахунок швидкого виділення додаткових ресурсів.
- *Резервне копіювання даних у хмарне сховище та аварійне відновлення IT сервісів* захищають всю інформацію від загроз, спричиненими кібератаками, несанкціонованим доступом, стихійними лихами тощо.
- *Угоди про рівень обслуговування (SLA)* визначають рівень надійності та гарантований час безперебійної роботи, включають гарантії щодо доступності послуг, продуктивності та ін. параметрів.

- *Заходи безпеки* для захисту від витоку даних, несанкціонованого доступу та інших загроз.

Основні моделі надання хмарних послуг розподіляють на рівні за принципом відповідальності та контролю над інфраструктурою [32]:

1. *Інфраструктура як послуга (SaaS або Software-as-a-Service)*: як правило, забезпечує базові обчислювальні ресурси – процесори і пристрої для зберігання інформації – і використовують їх для створення власних операційних систем і застосунків. Користувач не керує базовою інфраструктурою хмари, але має контроль над операційними системами, системами зберігання, розгорнутими застосунками. Дуже зручно використовувати SPA застосунки для цієї моделі.

2. П*латформа як послуга» (PaaS або Platform-as-a-Service)*: користувачі мають можливість встановлювати власні застосунки на платформі, що надається провайдером послуги. Користувач не керує базовою інфраструктурою хмари: мережами, серверами, операційними системами та системами зберігання даних, але має контроль над розгорнутими застосунками і деякими параметрами конфігурації середовища хостингу.

3. *Програмне забезпечення як послуга» (IaaS або Infrastructure-as-a-Service)*: клієнти отримують доступ до програм, які не потрібно встановлювати на персональні комп'ютери чи сервери, а користувачу для роботи потрібен тільки веб-браузер. Споживач користується застосунками провайдера, який працює в хмарній інфраструктурі. Користувач не керує базовою інфраструктурою хмари: мережами, серверами, операційними системами, системами зберігання, навіть індивідуальними настройками застосунків за винятком деяких налаштувань конфігурації програми. Прикладами такої моделі є сервіси Gmail, Google Disk, Slack, Salesforce, Microsoft 365, Cisco WebEx, Evernote.

Таким чином, технологія хмарних сервісів для створення веб-застосунків забезпечує гнучкість, масштабованість і доступність для користувачів будь-де та будь-коли. Розробники можуть використовувати хмарні ресурси для

розгортання та запуску веб-застосунків, не спираючись на локальне обладнання та інфраструктуру.

2.5.2 Серверний рендеринг

Основними підходами, що покращують роботу фронтенд застосунків, є серверний рендеринг (SSR, Server-Side Rendering) та віртуалізація. SSR – це підхід до розробки веб-застосунків, коли HTML сторінка формується на сервері та передається клієнту, який отримує готову сторінку для відображення. Основна ідея SSR полягає в тому, що весь вміст сторінки, включаючи дані та розмітку, генерується на сервері перед тим, як відправити його клієнту [37, 38].

Angular являється повноцінним фреймворком, він містить офіційний Angular Universal для серверного рендерингу та вбудовані компоненти для віртуалізації. Vue також за замовчуванням має SSR пекедж Server-Renderer.

Використання SSR дозволяє покращити такі функції, як [37, 38]:

- Покращити ініціальну швидкість завантаження, оскільки перший запит користувача отримує повністю відформатовану HTML сторінку, що дозволяє швидше показати йому вміст, особливо на початкових етапах завантаження.
- Покращити SEO, оскільки дає можливість пошуковим системам індексувати та інтерпретувати вміст сторінок, що створює більш сприятливі умови для підвищення видимості в пошукових результатах.
- Підтримати кешування, що в свою чергу, зменшує навантаження на сервер та покращує швидкодію веб-застосунків.

2.5.3 Асинхронність подій та обробка потоків даних

AJAX (асинхронний JS XML) – це технологія веб-розробки, яка забезпечує асинхронний зв’язок між веб-браузером і сервером, полегшує оновлення змісту веб-сторінок без перезавантаження всієї сторінки.

Незважаючи на те, що ця технологія не є новою, вона використовується і підтримується багатьма веб-застосунками [39].

Reactive Extensions for JS (RxJs) – це бібліотека для реактивного програмування з використанням Observables, щоб полегшити створення асинхронного коду або коду на основі зворотного виклику. Це потужна, незалежна бібліотека реактивного програмування, яка реалізує концепцію спостережуваних послідовностей, відомих як Observables, що представляє собою послідовні значення, які можуть видаватися асинхронно з часом. Можна розглядати як постачальника даних, який видає значення в різні моменти часу та доставляє їх у вигляді підписки. До основних переваг обох технологій можна віднести такі параметри та характеристики, як [39, 40]:

- *Асинхронність виконання запитів*, тобто без очікування завершення попереднього запиту. Це дозволяє користувачам працювати з веб-сторінкою без затримок, при невиконанні запросів програма продовжить далі працівати.
- Завдяки асинхронності запитів браузер може надсилати запит на сервер і отримувати відповідь *без перезавантаження сторінки, що дозволяє* динамічно оновлювати вміст сторінки, коли відбуваються зміни сервера або користувача.
- Дозволяють *обмін даними* між браузером і сервером у фоновому режимі, які, як правило, передаються у форматі XML або JSON. Це дозволяє веб-застосунку отримувати і надсилати дані на сервер.

2.6 Огляд основних фреймворків для розробки веб-застосунків

Фреймворки являються потужними засобами розробки програмного забезпечення, які надають готові архітектурні рішення та шаблонний код. Фреймворки дозволяють розробникам ефективно створювати застосунки з використанням найкращих практик і загальноприйнятих стандартів, надаючи

зрозумілий та структурований підхід до розробки. Вони визначають, як організовувати компоненти застосунку та як ефективно взаємодіяти з ними.

2.6.1 Серверна інфраструктура

Java. У 90-х роках XX ст. Java було розроблено в лабораторіях компанії Sun Microsystems, розробку проекту започаткував James Gosling. На початок 2022 р. її у своїх проектах використовують Google, Amazon, Meta, Microsoft, Uber, Netflix, eBay, Spotify, TripAdvisor, Intel, Pinterest, Groupon, Slack, Flipkart та багато ін. За даними, приведеними в джерелах інформації, близько 10 130 компаній використовують Java у своїх IT-системах [44, 45]. Так, У США доля підприємств, які використовують Java, становить близько 60 % клієнтів (близько 64 000 компаній).

Java являється мовою програмування загального призначення, що відноситься до об’єктно-орієнтованих мов програмування, мов із сильною типізацією. У веб-розробці Java використовується переважно для створення серверної частини.

До основних переваг використання Java для серверної розробки відносяться такі параметри [44, 45] :

- *Масштабованість і відмовостійкість*: Java має надійний механізм перевірки типів, що робить її довговічною. Віртуальна машина Java (JVM) забезпечує динамічне зв’язування та безпечне середовище, що дозволяє працювати Java будь-де. Автоматичне керування пам’яттю та колекція видалення пам’яті Java роблять її масштабованою, що дозволяє прискорити розробку веб-застосунків.
- *Бібліотека з відкритим кодом*: переважна більшість бібліотек Java є безкоштовними та з відкритим вихідним кодом за підтримки експертів, що значно пришвидшує бекенд програмування веб-проектів. Існує багато бібліотек Java, які призначені для різних цілей, зокрема журналювання,

розбір JSON, модульне тестування, розбір XML і HTML, обмін повідомленнями, читання PDF і Excel, криптографію та багато ін.

- *Різноманітність:* Java вже давно є основною мовою програмування для розробки веб-застосунків, застосунків для Android і програмних засобів, таких як Eclipse, IntelliJ IDEA, NetBeans IDE та ін. Варіанти використання Java розширилися й охоплюють застосунки для обробки даних, програми машинного навчання та навіть програми IoT.
- *Безпека*: Java містить вбудований Security Manager, який дозволяє встановити чіткі правила доступу до БД. Також мова не використовує вказівники, тобто частини програми можуть отримати доступ до осередку пам’яті лише після належної перевірки дозволу.
- *Незалежність платформи:* код, написаний Java працює на різних ОС без додаткових доробок. Розробники реалізували принцип WORA: (Write Once, Run Anywhere). Це завдання вирішується завдяки компіляції написаного на Java коду в байт-код. Цей формат виконує JVM або віртуальна машина Java. JVM – частина середовища виконання Java (JRE, Java Runtime Environment). Віртуальна машина не залежить від платформи та може працювати на будь-якій машині, незалежно від операційної системи.
- *Підтримка багатопотоковісті*: Java має вбудовану підтримку багатопотоковості – кілька потоків можуть працювати одночасно. Потік – це виконуваний, легкий блок, який отримує доступ до спільних ресурсів, а також до власного стеку викликів. Багатопотоковість дозволяє максимально використовувати центральний процесор (ЦП); кілька потоків використовують один і той же простір пам’яті, підвищуючи ефективність і продуктивність програми.

Spring Boot є одним з самих популярних фреймворків для розробки веб-застосунків у мові програмування Java. Spring Boot є налаштуванням над фреймворком Spring. Він надає широкий набір функцій і компонентів для

створення масштабованих та швидкорозгортуємих веб-застосунків завдяки своїй конфігурації за замовчуванням та автоматизації [46, 47].

Java Persistence API (JPA). Одним з найпопулярніших компонентів, який часто використовується для роботи з БД, на ряду з Spring Boot є JPA. JPA є стандартом для роботи з об'єктно-реляцційним відображенням (ORM) в Java-застосунках. JPA дозволяє визначати сутності (Entity) – класи, які представляють таблиці в БД, та встановлювати між ними відношення. Використання анотацій JPA дозволяє встановленню маппінгу між сутностями і стовпцями БД. Spring Boot надає зручні можливості для автоматичного створення таблиць БД на основі конкретних сутностей [48].

Ще однією перевагою використання JPA з Spring Boot є підтримка транзакцій, також існує можливість використовувати анотації JPA для позначення методів, які мають бути виконані в межах транзакції. Spring Boot забезпечує керування транзакціями та автоматичне керування сесіями БД, забезпечуючи консистентність даних і запобігання їх втрати [48].

Крім того, JPA підтримує механізм запросів JPQL (Java Persistence Query Language), який спрощує виконання складних запитів до БД, використовуючи об'єктно-орієнтований синтаксис. Це дозволяє скорочувати шаблоний код, а також підвищує читабельність коду. JPA дозволяє використовувати JPQL для: фільтрації, сортування та об'єднання даних з БД, для виконання агрегатних функцій.

Spring Boot має наступні переваги і характеристики, які роблять його популярним серед розробників [46-4?9] :

- *Спрощена розробка*: забезпечує оптимізовану розробку завдяки мінімізації конфігурації, необхідної для налаштування програми. Дотримується принципу програмування «угода головніша за конфігурацію» (Convention over configuration), який полягає в тому, що замість наявної конфігурації використовується неявна «угода». Це дозволяє зменшити шаблонний код і більше зосереджуватися на бізнес-логіці.

- *Швидка розробка застосунків*: за допомогою функції автоматичного налаштування можна швидко налаштувати та розгорнути застосунки, не витрачаючи час на налаштування вручну. Забезпечує параметри за замовчуванням і розумні конфігурації, зменшуючи потребу в ручному втручанні.
- *Архітектура мікро сервісів:* підтримує створення незалежних служб невеликого розміру, які можна розгортати та масштабувати окремо. Він також добре інтегрується з іншими проектами Spring, такими як Spring Cloud, для впровадження хмарних програм.
- *Впровадження залежностей (DI) та інверсія управління* (Inversion of Control, IoC) забезпечує вільний зв'язок і модульну розробку. Це сприяє кращій організації коду, тестуванню та зручності обслуговування.
- *Вбудований сервер*: містить вбудований сервер (Tomcat, Jetty або Undertow), який усуває необхідність розгортання програм на зовнішніх веб-серверах. Дозволяє розробнику самостійно запускати програму без необхідності використання веб-сервера.
- *Гнучкі конфігурації* з конфігураціями XML, Java Beans і транзакції БД.
- *Екосистема Spring:* використовує розгалужену екосистему Spring, яка включає Spring JDBC, Spring Data, Spring Security і Spring ORM, що надає широкий спектр бібліотек, модулів та інтеграцій для різних цілей: доступ до даних, безпека, обмін повідомленнями тощо. Він сприяє багаторазовому використанню коду та надає рішення для звичайних потреб корпоративних застосунків.
- *Легке тестування* завдяки підтримці модульного та інтеграційного тестування. Він надає інструменти CLI та анотації, які спрощують написання та виконання тестів.
- *Сумісність з екосистемою Java*, включаючи бібліотеки, фреймворки та інструменти. Це дозволяє інтегрувати з різними технологіями на основі Java

і спрощує взаємодію. Розробники Java також можуть підключати Spring Boot до різних БД, таких як Oracle, MySQL, PostgreSQL і MongoDB.

Таким чином, Java Spring Boot забезпечує надійну та ефективну структуру для розробки веб-застосунків Java, пропонуючи простоту, швидку розробку, масштабованість та можливості інтеграції.

2.6.2 Клієнтська частина

Angular – популярний фреймворк розробки веб-застосунків, яку підтримує Google та Microsoft. Angular являється повноцінним фреймворком, який відповідає архітектурі MVC, завдяки чому веб-застосунки є більш структурованими, полегшуються процеси їх тестування та розробки. Angular є відкритим кодом і написаний на Typescript. Можливо використання Angular для великих корпоративних застосунків, SPA, PWA, Gmail і YouTube TV створені на Angular [51]. Основні характеристики та функції Angular приведені нижче [50 - 52, 64, 65]:

- *Повнофункціональний фреймворк*: Angular є повним фреймворком, який надає повний набір інструментів і функцій для веб-розробки, таких як архітектура на основі компонентів, зв’язування даних, впровадження залежностей, маршрутизація та перевірка форми.
- *TypeScript*: Angular створено за допомогою TypeScript, що розширює можливості JavaScript. Завдяки TypeScript можливі такі функції, як статичний тип, класи, інтерфейси та модулі, які забезпечують ефективну організацію коду, зручність обслуговування та підтримку інструментів.
- *Двостороннє зв’язування даних*: Angular дозволяє автоматичну синхронізацію даних між інтерфейсом користувача та моделлю даних програми, що спрощує маніпуляцію даними та зменшує кількість шаблонного коду.
- *Компонентна архітектура*: Angular просуває компонентну архітектуру, де застосунок будується, як набір повторно використовуваних і модульних

компонентів. Цей підхід покращує можливість багаторазового використання коду, зручність обслуговування та тестування.

- *Впровадження залежності* (DI): полегшує керування та впровадження залежностей у компоненти; підтримує модульний і роз'єднаний код, що полегшує написання модульних тестів і підтримку кодової бази.
- *Angular CLI:* Angular надає інструмент інтерфейсу командного рядка, який автоматизує різні завдання розробки, такі як налаштування проекту, створення коду, тестування та розгортання, що спрощує процес розробки та підвищує продуктивність.
- *Кросплатформна розробка* дозволяє розробникам створювати веб-застосунки, які можуть працювати на різних платформах, включаючи настільні та мобільні застосунки.
- *Дострокова компіляція* (AOT): функція, яка перетворює код Angular HTML і TypeScript в ефективний код JavaScript на етапі збірки, перш ніж браузер завантажить і запустить цей код. Компіляція програми під час процесу збірки забезпечує швидку візуалізацію в браузері, що допомагає зменшити розмір програми, час завантаження та збільшити загальну продуктивність.

Таким чином, переваги та функції Angular роблять його потужною платформою для створення складних і масштабованих веб-застосунків, пропонуючи надійну систему розробки та широкий набір інструментів для оптимізації процесу розробки.

Vue.js – це фреймворк JavaScript, який забезпечує декларативну та компонентну модель програмування, яка допомагає ефективно розробляти прості або складні інтерфейси користувача. Vue.js в основному застосовують для розробки складних реактивних інтерфейсів, різноманітних складних односторінкових застосунків SPA і адміністративних панелей, декларативного рендерингу [25, 43].

Фреймворк Vue.js заснований на шаблоні MVVM (Model-View-ViewModel), похідному від класичного шаблону (MVC). Його поява сприяла розподілу інтерфейсної розробки графічного UI користувача та внутрішньої

бізнес-логіки, що значно підвищує ефективність створення програм. Ядро MVVM – рівень ViewModel, схожий на перетворювач значень, що відповідає за зміну об'єктів даних у моделі.

Vue.js має наступні характеристики та пропонує функції, які роблять його кращим вибором для веб-розробників [20, 43, 50]:

- *Декларативне відображення*: розширює стандартний HTML синтаксисом шаблону, який дозволяє декларативно описувати вивід HTML на основі стану JavaScript.
- *Реактивне зв'язування даних*: використовує підхід реактивного зв'язування даних, що дозволяє розробникам декларативно прив'язувати дані до шаблону HTML. Зміни в даних автоматично відображаються в інтерфейсі користувача і навпаки, без необхідності ручного маніпулювання DOM. Це дозволяє легко підтримувати синхронізацію між моделлю даних і інтерфейсом користувача.
- *Архітектура на основі компонентів*: застосунок побудовано як набір компонентів, які можна використовувати повторно. Компоненти інкапсулюють логіку HTML, CSS і JavaScript, роблячи кодову базу модульною, придатною для багаторазового використання та простішою в обслуговуванні.
- *Віртуальний DOM*: використовує віртуальний DOM для ефективного оновлення та відтворення інтерфейсу користувача, який порівнює поточний стан інтерфейсу користувача з потрібним станом і оновлює лише необхідні частини, що призводить до покращення продуктивності.
- *Гнучкість і поступова адаптація*: в залежності від задач Vue js можна використовувати різними способами:
    - покращення статичного HTML без етапу створення;
    - вбудовування як веб-компонентів на будь-якій сторінці;
    - односторінкова програма (SPA);
    - Fullstack / Server-Side Rendering (SSR);

- Jamstack / Генерація статичного сайту (SSG);
- орієнтація на комп’ютер, мобільний пристрій, WebGL, термінал.

- *Стилі API:* компоненти Vue можна створювати у двох різних стилях – Options API та Composition API .
- *Комплексна екосистема* з широким набором плагінів, бібліотек та інструментів, що розширює його можливості, надаючи рішення для маршрутизації, керування станом, перевірки форм, анімації тощо.
- *Vue CLI*: Vue.js надає інструмент інтерфейсу командного рядка, який допомагає в налаштуванні, розробці та розгортанні проекту; пропонує оптимізовану розробку з такими функціями, як створення проектів, перезавантаження та вбудований сервер розробки.
- *Кросплатформна підтримка*: Vue Native це платформа JavaScript, призначена для створення кросплатформенних мобільних застосунків, які можуть працювати як на Android, так і на iOS.

Таким чином, Vue.js містить більшість загальних функцій, необхідних для розробки інтерфейсу. Він поєднує зручність, потужність, простоту, продуктивність, гнучкість, широкий набір для створення сучасних, інтерактивних та масштабованих веб-застосунків.

2.6.3 База даних

PostgreSQL являє собою потужну об’єктно-реляційну БД з відкритим вихідним кодом, активна розробка якої триває понад 35 років, завдяки чому вона заслужила міцну репутаційну надійність, функціональність та продуктивність. Завдяки цим принципам роботи та широкому функціоналу PostgreSQL має широку сфера використання [55, 58]:

- *Серверна база даних для мобільних програм та для веб-застосунків*: PostgreSQL може обробляти великі обсяги даних, забезпечувати швидке та надійне зберігання даних і підтримувати складні запити.
- *Геопросторові програми*: PostgreSQL має надійну підтримку типів і функцій геопросторових даних, що дає можливість використовувати в застосунках Географічної інформаційної системи (ГІС), службах визначення місцезнаходження, платформах картографування та просторовій аналітиці.
- *Сховища даних:* завдяки здатності PostgreSQL ефективно зберігати й аналізувати великі масиви даних, стартапи та великі підприємства використовують базу як основне сховище даних для підтримки своїх інтернет-застосунків.
- *Фінансові системи*: банківські, страхові та торгові платформи.
- *Платформи е-комерції*: для каталогів продуктів, керування замовленнями, управління запасами та даними про клієнтів.
- *Серверна база даних для програм IoT (Інтернет речей).*

Такі принципи роботи PostgreSQL та її запропоновані численні функції роблять PostgreSQL популярним вибором для різних програм [55, 58] :

- *Надійність і стабільність*: PostgreSQL забезпечує цілісність даних і мінімізує ризик їх втрати або пошкодження завдяки дотриманню принципів Acid – набір властивостей, що гарантують надійну роботу транзакцій БД: атомарність (Atomicity), узгодженість (Consistency), ізольованість (Isolation), довговічність (Durability).
- *Розширені типи даних*: PostgreSQL підтримує різні типи даних, включаючи числові, рядкові, дата/час, логічні значення, масиви, JSON та багато ін.
- *Процедурні мови:* у PostgreSQL передбачена підтримка внутрішніх процедурних мов, в тому числі спеціалізованої мови PL/pgSQL, що є аналогом PL/SQL, процедурної мови Oracle.
- *Сумісність між платформами*: PostgreSQL працює на різних операційних системах, включаючи Linux, Windows, macOS і UNIX-подібних системах.

- *Висока масштабованість і продуктивність*: PostgreSQL розроблено для роботи з великомасштабними програмами з великим трафіком; вона підтримує такі розширені функції, як паралельна обробка, метод керування паралельним доступом (MVCC) і методи оптимізації запитів, що забезпечують високу продуктивність при значних навантаженнях. PostgreSQL має високу масштабованість як щодо кількості даних, якими він може керувати, так і щодо кількості одночасних користувачів, яких він може прийняти.
- *Розширені можливості*: PostgreSQL пропонує багатий набір функцій SQL і підтримує складні запити, підзапити, об'єднання та розширені функції; віконні функції, загальні табличні вирази (CTE) і повнотекстовий пошук. PostgreSQL має надійні набори функцій, включаючи MVCC, відновлення на певний момент часу, деталізований контроль доступу, табличні простори, асинхронна реплікація, вкладені транзакції, онлайнове/гаряче резервне копіювання, удосконалений планувальник/оптимізатор запитів і *ведення журналу* з випередженням запису.
- *Технологія MVCC*: архітектура MVCC дозволяє одночасний доступ до бази даних, зберігаючи узгодженість транзакцій, це гарантує, що кілька транзакцій можуть читати та записувати в базу даних одночасно.
- *Відповідність стандартам SQL та підтримка функцій SQL*.
- *Простота та наочністю інтерфейсу* оболонки pgAdmin.
- *Розширюваність і налаштування*: користувач може налаштовувати систему шляхом визначення нових функцій, мов, типів, агрегатів, індексів тощо. PostgreSQL забезпечує розширюваність за допомогою визначених користувачем функцій, процедурних мов PL/pgSQL, PL/Python, PL/Java, а також можливістю створювати власні розширення. Це дає можливість розробникам розширити функціональні можливості бази даних і реалізувати спеціальну бізнес-логіку в самій БД.

- *Зберігаюча реєстрація WAL* (Write-Ahead Logging): *ведення журналу* з випередженням запису робить PostgreSQL високовідмовостійкою ДБ. Ця процедура дозволяє не переписувати сторінки даних на диску при кожній транзакції, так як в разі аварії є можливість відновити БД за допомогою журналу. Механізм WAL забезпечує такі переваги: підвищення продуктивності роботи СУБД за рахунок того, що записуються тільки внесені зміни без переписування всіх даних в таблицях; підвищення надійності зберігання даних за рахунок попереднього збереження буферизованих даних у WAL; можливість повернення стану БД на будь-який момент часу шляхом застосування WAL до існуючої резервної копії.
- *Відповідність стандартам*: PostgreSQL сумісний з ACID і має повну підтримку зовнішніх ключів, об’єднань, представлень, тригерів і збережених процедур багатьма різними мовами. Він містить більшість типів даних SQL:2008. PostgreSQL також підтримує зберігання великих двійкових об’єктів, включаючи зображення, звуки або відео.
- *Реплікація та технологія Hot Standby*: технологія Hot Standby надає можливість звертатися до сервера та виконувати запити на читання, поки сервер перебуває в режимі відновлення або очікування, що удосконалює процес реплікації та відновлення резервної копії до потрібного стану з великою точністю.
- *Гнучке налаштування серверу:* основний конфігураційний файл postgresql.conf включає більше 150 параметрів, що настроюються по розділах: файли і шляхи до них, авторизація та безпеку, виділення ресурсів та ін. Додатковий конфігураційний файл pg_hba.conf включає в себе налаштування доступу до окремих БД, такі як вказівку конкретних IP-адрес і мереж, з яких дозволений доступ, а також метод авторизації для доступу в БД і можливість включення безпечних зашифрованих з'єднань.
- *База даних об’єднаного концентратора*: оболонки зовнішніх даних PostgreSQL і підтримка JSON дозволяють йому зв’язуватися з іншими

сховищами даних, включаючи типи NoSQL, і діяти як об’єднаний центр для багатомовних систем баз даних.

- *Стек з відкритим кодом LAPP*: PostgreSQL може запускати динамічні веб-сайти та програми як надійну альтернативну частину стеку LAMP (набору із чотирьох різних програмних технологій, які розробники використовують для створення веб-сайтів та веб-застосунків, Linux, веб-сервер Apache, сервер баз даних MySQL та мови програмування PHP).
- *Клієнт-серверна архітектура* з розподілом процесів між користувачами.
- *Підтримка геопросторових даних* через розширення PostGIS.
- *Відкритий вихідний код*: PostgreSQL є безкоштовною для використання та не вимагає жодних фінансових витрат на ліцензування.
- *Підтримка спільноти паралелізму*: кілька користувачів PostgreSQL можуть мати доступ до даних одночасно.
- *Високий професіоналізм команди розробників* забезпечує PostgreSQL постійне вдосконалення, регулярні оновлення тощо, нові інноваційні розробки прагнуть утримати її на передових позиціях серед БД.

Таким чином, структура та функціонал PostgreSQL з відкритим вихідним кодом, надійністю, розширюваністю, продуктивністю роблять її придатною для широкого спектру застосунків від невеликих проектів до великих корпоративних систем.

MongoDB – документо-орієнтована система керування БД з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB створена на основі розподіленої масштабованої архітектури та пропонує комплексну хмарну платформу для керування та доставки даних у програми; обробляє транзакційні, операційні та аналітичні навантаження в масштабі [57, 58].

MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СУБД, функціональними і зручними у формуванні запитів. MongoDB працює з документо-орієнтованою моделлю даних, на відміну від реляційних СУБД,

MongoDB не потребує таблиці, схеми або окремої мови запитів. Інформація зберігається як документ чи колекція.

MongoDB має ряд властивостей, які виділяють її від інших БД [57, 58]:

- *Документо-орієнтована*: MongoDB зберігає дані в гнучких напівструктурованих документах (колекціях), а не в традиційних таблицях із чітко заданою кількістю стовпців та типів даних. Кожна колекція має своє унікальне ім'я – довільний ідентифікатор. Дані зберігаються у форматі під назвою BSON, який є двійковим представленням JSON, що полегшує роботу з даними у веб-застосунках та ін. системах на основі JSON.
- *Масштабованість і продуктивність:* MongoDB розроблено для горизонтального масштабування на кількох серверах, що забезпечує високу масштабованість і продуктивність, та підтримує сегментування, що дає змогу розподіляти дані між кількома машинами, і набори реплік, які забезпечують високу доступність і відмовостійкість.
- *Гнучкий дизайн схеми* дозволяє створювати динамічні структури даних, що змінюються. На відміну від традиційних реляційних БД, надає можливість мати різноманітні структури для документів у колекції, що спрощує виконання нових вимог до даних.
- *Гнучка модель даних* дозволяє легко розвивати схему та підтримує розширені запити та складні агрегації.
- *Відкритий вихідний код*: вільна для використання та може бути змінена та налаштована відповідно до конкретних потреб.
- *Підтримка спеціальних запитів*, які можна виконувати без попередньо визначених схем або структур.
- *Автоматичне сегментування*: MongoDB має вбудовану підтримку автошардингу, тому може автоматично розподіляти дані між кількома серверами на основі попередньо визначених правил, що дозволяє легко масштабувати БД даних відповідно до зростання обсягу даних.

- *Потужна спільнота та екосистема*, яка надає підтримку, ресурси та величезну екосистему бібліотек, інструментів і фреймворків.
- *Сумісність з різними ОС*, включаючи Linux, Windows і macOS та ін; має офіційні драйвери та клієнтські бібліотеки для багатьох мов програмування, що полегшує інтеграцію в різні програм.

Таким чином, MongoDB забезпечує гнучку модель даних, динамічну схему, потужну мову запитів, підтримку вбудованих документів і масивів, індексування та інтеграцію з різними мовами програмування, що дозволяє розробникам створювати дуже гнучкі та адаптовані програми.

2.6.4 Порівняльна характеристика PostgreSQL та MongoDB

Реалізація різних проектів, корпоративних та хмарних застосунків вимагають різні вимоги до їх баз. Для обґрунтованого вибору БД для вирішення поставлених задач на підставі опрацьованих літературних та наукових джерел проведена порівняльна характеристика двох БД: SQL – PostgreSQL та NoSQL – MongoDB. Порівняльна характеристика приведена у таблиці 2.5 [58] .

Оскільки PostgreSQL має монолітну/одновузлову архітектуру та використовує моделі реплікації master-slave, PostgreSQL не володіє такими важливими особливостями, як лінійна масштабованість запису (автоматичного поділу на кілька вузлів) та автоматичної/нульової втрати даних.

Архітектура ядра БД MongoDB розподілена: дані розбиваються на секції та розподіляються по кількох вузлах, що потребує їх денормалізації. MongoDB акцентується на досягненні високої продуктивності в розподіленому кластері, одночасно поєднує гарантії ACID, в першу чергу, операцій в межах одного документа чи транзакції. Існує думка, що, незважаючи на те, що MongoDB забезпечує лінійну масштабованість запису і високу відмовостійкість, втрата транзакційних гарантій робить їх непридатними для критично важливих даних.

PostgreSQL забезпечує застосування структурованих схем, що робить її придатною для програм із чітко визначеними та статичними структурами

даних. Тоді як, гнучка безсхемна модель MongoDB дозволяє створювати структури даних, які змінюються, це дозволяє використовувати її для програм з вимогами до динамічних даних.

Серед характеристик, які властиві як PostgreSQL, так і MongoDB, можна виділити наступні:

- пропонують *функції реплікації*, що дозволяє створити копію вже наявної БД, в результаті чого підвищуються масштабованість і доступність, надійне збереження даних;
- *підтримують індексування* для покращення продуктивності запитів, що дозволяє скоротити час доступу до даних на основі певних критеріїв;
- являються системами керування базами даних *з відкритим кодом;*
- забезпечують такі *функції безпеки*, як автентифікація, авторизація та шифрування для захисту даних;
- мають *активні спільноти та екосистеми*.

Таблиця 2.5 – Порівняльна характеристика PostgreSQL та MongoDB

Характеристика	PostgreSQL	MongoDB
Модель даних	Реляційна БД	Документно-орієнтована БД NoSQL
Архітектура ядра БД	Монолітна/одновузлова, транзакційна SQL	Розподілена, не транзакційна NoSQL

Продовження таблиці 2.5

Підхід до проектування БД	Моделювання даних у SQL	Моделювання запитів у NoSQL
SQL	Підтримує всі основні операції SQL	Використовує власну мову запитів

Відповідність вимогам ACID	Сумісна з ACID і забезпечує виконання всіх вимог	Надає гарантії ACID на рівні документа або транзакції. Підтримує транзакції з кількома документами, починаючи з версії 4.0.
Організація даних	Нормалізована	Денормалізована
Гнучкість даних	Підтримує структуровані динамічні схеми	Гнучка безсхемна модель даних
Масштабованість	*Підтримує* функцію горизонтального *масштабування,* але більш потребує ручного налаштування	Має вбудовану підтримку горизонтальної масштабованості та автошардингу.
Гнучкість схеми	Забезпечує застосування структурованих схем, вимагаючи попередньо визначених структур таблиць і типів даних.	Безсхемний характер забезпечує гнучке та динамічне моделювання даних

Кінець таблиці 2.5

Сфера застосування	Складні реляційні дані	Прості нереляційні дані
Індексування	Підтримують індексування для покращення продуктивності запитів	
Відкритий код	Являються системами керування базами даних з	

	відкритим кодом
Безпека	Забезпечують такі функції безпеки, як автентифікація, авторизація та шифрування для захисту даних

Таким чином, у другому розділі роботи проведений аналіз програмно-технічних рішень для створення веб-застосунків. Для цього проаналізовані різні архітектурні підходи щодо організації програмного забезпечення і взаємодії компонентів, розглянуті принципи роботи дворівневої та трирівневої архітектури, визначені переваги та недоліки кожної.

У рамках другого розділу проведений аналіз та надана порівняльна характеристика архітектурних рішень у веб-розробці односторінкових, багатосторінкових та прогресивних застосунків, проведений їх порівняльний аналіз з визначенням їх сильних та слабких характеристик.

Також було розглянуто основні характеристики моделей керування доступом, які являються основним елементом кібербезпеки. Проведений аналіз патернів MVC та MVVM, на підставі чого зроблена порівняльна характеристика їх основних функцій та параметрів.

Для коректного вибору фреймворків розглянуті та вивчені основні фреймворки для розробки серверної інфраструктури та клієнтської частини веб-застосунків. Визначені принципи роботи та основні функції баз даних PostgreSQL PostgreSQL, на підставі опрацьованих даних зроблена їх порівняльна характеристика.

РОЗДІЛ 3 ПРОЕКТУВАННЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ЗАПАСАМИ ЛІКІВ

3.1 Інженерія вимог веб-застосунку для управління запасами ліків

На підставі проведеного аналізу програмно-технічних рішень для створення веб-застосунків сформовано основне завдання поставленої роботи: спроектувати та розробити веб-застосунок для управління запасами ліків для підприємства малого бізнеса, грунтуючись на аналізованих ERP-системах [16 - 20]. Для створення веб-застосунку обраний підхід клієнт-серверної архітектури [4 - 6], яка враховує сучасні тенденції та можливості масштабованості системи.

Основними вимогами до проекту, що розробляється, є:

- Запобігання несанкціонованого доступу до даних, а також до конфіденційної інформації користувачів, обмеження функціоналу.
- Уникання довгого завантаження та оновлення інформації.
- Створення рольової моделі для керування ліками.
- Розробка серверної частини.
- Розробка СУБД.
- Розробка зручної клієнтської частини, основуючись на шаблоні проектування програмного забеспечення.

Захист даних при їх обробці можна забезпечити шляхом вибору політики розмежування доступу з урахуванням особливостей призначення веб-застосунку та специфіки фармацевтичного обліку та торгівлі. На підставі визначених переваг та недоліків MAC, RBAC і DAC, а також того, що працівники аптеки мають чітко визначені права та обов’язки згідно з посадовими інструкціями, найбільш релевантною політикою розділення доступу являється RBAC [27 - 31].

Для розробки клієнтської частини був використаний фреймворк Angular. Вибір Angular обґрунтований його характеристиками, які відповідають вимогам веб-застосунків з обліку та керування ліками, а саме архітектурне рішення SPA [23, 53].

Такий вибір архітектурного рішення має декілька підстав:

- висока швидкість, коли усі елементи завантажується з першого разу, під час виконання різних дій на сторінці дані просто змінюються;
- економія часу: практично вся робота відбувається у браузері, без звернень до сервера.
- досить висока продуктивність за рахунок оптимізації коду та зменшення кількості запитів до сервера.

Такі параметри є оптимальними для веб-застосунків, у яких передбачена велика кількість запитів до сервера.

Компонентна архітектура: Angular дозволяє розбити інтерфейс, сторінки на окремі компоненти, що спрощує організацію і підтримку коду, особливо великих і складних адміністративних інтерфейсів. Angular спрощує реалізацію маршрутизації та має набір інструментів для захищеності компонентів. Крім того, Angular має широку та активну спільноту розробників, що забезпечує наявність різноманітних інструментів, бібліотек і розширень для підтримки розробки SPA веб-застосунків [23, 38, 42, 51, 52, 64].

Для розробки сучасного дизайну UX/UI були запроваджені SASS, Bootstrap [52, 68-70]

Для розробки серверної частини був використаний фреймворк Spring Boot, який має вбудовані бібліотеки для забезпечення безпеки даних, а також реалізацію сучасних технологій безпеки, таких як Blockchain і JWT [60 - 63]. Реалізується на MVC патерні проектування, що спрощує розробку і тестування функціоналу [67]. Спрощує розробку і проектування СУБД, за допомогою модуля JPA [48], для тестування API [3], було використано залежність Swagger2 [49].

3.2 Розподіл ролей та структура доступу, використання jwt-токенів для реалізації рольової моделі у веб-застосунку для управління запасами ліків

Для забезпечення безпеки і конфіденційності даних, високої продуктивності роботи, пов'язаної з управлінням ліків, запровадимо модель

RBAC, яка дозволяє чітко визначити права доступу кожного користувача згідно з його роллю та професійними обов'язками та повноваженнями [29, 31].

Основними перевагами моделі RBAC при використанні в аптечному обліку медичних товарів являється:

- Забезпечення безпеки, оскільки дозволяють надати доступ до конфіденційної інформації про наявність та запаси ліків тільки відповідним працівникам аптеки відповідно до їх посадових інструкцій. Це зменшує ризик несанкціонованого доступу та зловживання перевагами.
- Збільшує продуктивність, так як дає можливість працівникам аптеки швидко та своєчасно отримувати необхідну інформацію стосовно наявності та запасів ліків.
- Зменшує ризик неправильного введення даних щодо наявності лікарських препаратів та їх запасів, оскільки цю функції виконує відповідальна особа, яка пройшла інструктаж та відповідає саме за операціями на всіх етапах:
    - надходження товару та розміщення на складі;
    - переміщення товару;
    - продаж товару;
    - списання товару зі складу.

  Ролі та права доступу для кожної ролі приведені у таблиці 3.1

Таблиця 3.1 - Ролі та права доступу

Назва ролі	Права доступу
Незареєстрован	- Здійснювати пошук і перегляд як конкретних медичних

ий користувач	препаратів, так і наявного асортименту продуктів. Для цього вони можуть використовувати пошукові фільтри для звуження пошуку за такими критеріями, як назва ліків. - Можуть додавати ліки до свойого локального середовища для покупок і переходити до розміщення замовлень; визначити потрібну кількість ліків. - Переглядати інформацію обраного лікувального засобу - Має функції сортування і фільтрації для ефективного та зручного пошуку ліків
Зареєстрований користувач	- Має всі права незареєстрованого користувача - Можуть створювати свої облікові записи та керувати ними: можуть оновлювати свою особисту інформацію, зокрема ім’я, контактні дані тощо. Також можуть змінювати паролі своїх облікових записів. - Можуть додавати ліки до своїх кошиків для покупок і переходити до розміщення замовлень; визначати потрібну кількість товару
Фармацевт-керівник/ Модератор	- Може створювати замовлення на закупівлю препаратів. Він має можливість оформити необхідну документацію та внести деталі щодо кількості та виду лікарських засобів, які потрібно замовити. - має можливість ознайомитись зі списком доступних лікарських засобів, які є в аптеці. Це дозволяє йому бути в курсі наявності та асортименту препаратів. - може створювати замовлення на закупівлю препаратів

Кінець таблиці 3.1

Адміністратор системи	- Додавання нових обліків медичних препаратів до складу

	системи. Може вносити головну інформацію про препарати, включаючи: зображення, назву, опис, ціну покупки, додаткові вказівки та інші важливі характеристики. - Реєстрація нових керівників. Може створювати нові облікові записи для фармацевтів керівників / модераторів, які будуть відповідальні за управління аптекою і запасами ліків. - Закріплення фармацевтів керівників / модераторів за аптекою: Адміністратор може призначати керівників для конкретної аптеки, забезпечуючи їх відповідальність за управління запасами ліків і виконання інших функцій. - Створення аптек: Може створювати новий облік аптеки в системі. - Управління каталогами.Надається динамічне керування каталогами медичних препаратів, зокрема створення, видалення, реданування, а також створення підкаталогів. - Перегляд і управління замовленнями. Може переглядати список замовлень, підтверджувати, відхиляти або відправляти замовлення на обробку.

Щоб забезпечити сформований чіткий план ролей, і їх повноважень використаємо JSON Web Token (JWT) - це об’єкт JSON, визначений у відкритому стандарті RFC 7519. Є одним із стандартів безпечного способу передачі інформації у формат JSON між сторонами. JWT складається з трьох частин: заголовка (header), тіла JWT(payload) та підпису (signature) [60, 62].

Для реалізації RBAC JWT-токени використовують для таких цілей [60]:

- *Автентифікація:* після успішної аутентифікації або реєстрації користувача сервер видає токен jwt, що містить інформацію про користувача та його роль, повноваження в системі.

- *Авторизація*: дозволяє системі авторизуватися на основі визначених ролей.
- *Верифікація:* сервер може перевіряти токени JWT і дозволити або заборонити доступ до ресурсів, (навіть якщо користувач підмінив токен) залежно від ролі користувача.
- *Інформаційна безпека*: токени JWT можна підписувати приватними ключами з секретним словом, що забезпечує цілісність і відсутність доступу до інформації, що зберігається в токені. Якщо токен було змінено або неправильно підписано, сервер відхилить токен.
- *Незалежність сеансу*: токени JWT не потребують збереження стану сеансу на сервері, тобто немає необхідності зберігати та відстежувати стан сеансу на сервері, що спрощує масштабування та розподіл архітектури.
- *Термін дії сесії:* токену JWT можна встановлювати термін дії сесії, він визначає період часу, протягом якого користувач може отримати доступ до системи без повторної ідентифікації. Основними причинами використання терміну сеансу є:
    - безпека, оскільки допомагає запобігти можливості несанкціонованого доступу до даних, якщо користувач залишається неактивним або забуває вийти з системи.
    - керування ресурсами: неактивні сеанси можна автоматично закривати, коли вони закінчуються. Ці зекономлені ресурси можна використати для підтримки важливих сеансів. Такі процеси дозволяють економити пам’ять і зберігати для поточних активних користувачів і процесів.

Процес роботи з використанням JWT токена для автентифікації та авторизації можна описати наступними крокамами:

1. Аутентифікація користувача:

a. Користувач звертається до сервера аутентифікації та надає свої облікові дані для входу, такі як пара значень логін/пароль, ключ від соціальної мережі або інший вид ідентифікаційного ключа.
b. Сервер аутентифікації перевіряє ці облікові дані і, якщо вони є правильними, генерує JWT токен для користувача.
c. Заголовок визначає тип токена та алгоритм шифрування, в пейлоад міститься клієнтська інформація (термін дії сесії, логін, роль, права), а підпис забезпечує цілісність і автентичність маркера.
d. Сервер аутентифікації надсилає згенерований JWT назад користувачеві.

2. Авторизація запитів з використанням JWT:
   a. Коли користувач робить запит до API, він включає отриманий JWT токен до заголовка або параметра запиту.
   b. Серверна частина отримує запит із JWT та перевіряє його для перевірки автентичності та облікових даних користувача.
   c. Перевірка JWT включає наступні кроки:
      i. Перевірка цілісності: сервер перевіряє підпис токена, використовуючи секретний ключ, яким він спільно володіє з сервером аутентифікації. Такі дії дозволяють переконатися, що токен не був змінений під час передачі.
      ii. Перевірка терміну дії: сервер перевіряє закінчення терміну дії маркера, щоб переконатися, чи маркер дійсний і термін його дії не минув. Якщо термін дії минув, сервер застосунків поверне повідомлення про помилку, і користувачеві доведеться повторно автентифікуватися.
      iii. Перевірка дозволів: пейлоад токена містить інформацію про користувача та його повноваження. Сервер застосунків перевіряє цю інформацію, щоб переконатися, що користувач авторизований для виконання запиту.

iv. Якщо перевірка маркера пройшла успішно, запит виконується. Якщо перевірка маркера не вдається, сервер повертає повідомлення про помилку або запитує у користувача повторну автентифікацію.

Процес роботи запита з jwt-токеном представлений на рисунку 3.1.

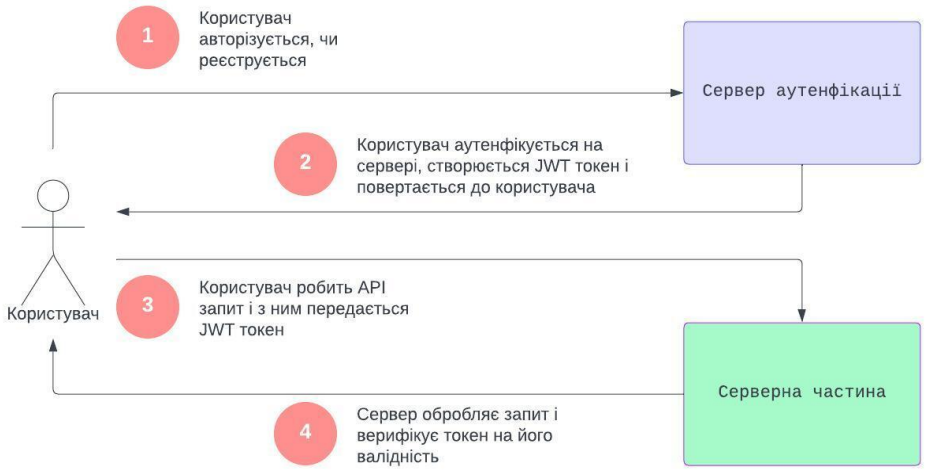

Рисунок 3.1 – Процес роботи запита з jwt-токеном

3.3 Розробка функціональності та структури веб-застосунку для управління запасами ліків

Для розробки веб-застосунку управління ліків був використаний популярний патерн проектування проектної системи, що базується на принципі моделі MVC. MVC дозволяє відокремити логіку програми від її представлення та взаємодії з даними, спрощуючи розробку, тестування та обслуговування програмного забезпечення. Архітектура MVC в Angular [64] представляється:

- Модель: В Angular дані можуть бути представлені у вигляді об’єктів або класів, що містять необхідні властивості та методи.
- Перегляд: Angular містить шаблони HTML для відображення даних моделі за допомогою директив і компонентів Angular, також можна використовувати бібліотеку Angular з ще більшим вибором mat-angular компонентів. Представлення також відповідає за дії користувача, ввід

даних або переходження на інші сторінки. Представлення можна редагувати використовуючи модель чи змінивши стиль.

- Контроллер: в Angular контроллер є компонентом, що діють як посередники між моделями та представленнями. Контроллер отримує дані з моделі, обробляє їх і передає в представлення для відображення. Крім того, компонент також може відстежувати та реагувати на зміни даних шляхом відповідного оновлення представлень.

За принципом паттерна MVC і компонентної архітектури програми, сформовано структуру клієнтської системи. У таблиці 3.2.1 представлена організаційна структура папок у проекті.

Таблиця 3.2 – Організаційна структура папок

Назва папок	Опис
component	Містить набір основних компонентів: форми керування, сторінки веб-застосунку, спискові структури представлення даних, child-компонентів, які можна перевикористати
form-type	Містить набір форм керування, створення, редагування, сортування, фільтрації даних
scroll-type	Містить набір, демонструючих компонентів з можливістю прокручування.
popup-type	Містить набір спливаючих, повідомляющих компонентів
tile-type	Містить набір child-компонентів, зачастую використовуються в циклах
list-type	Містить набір структурних списків з можливістю редагування чи перегляду в залежності прав користувача
page-type	Містить всі сторінки веб-застосунку

Кінець таблиці 3.2

directive	Група директив, які керують візуалізацію компонентів в залежності від ролі чи повноважень користувача
DTO	Група об’єктів виду DTO
guard	Група класів, які реалізують механізм захисту маршрутів, від несанкційного доступу користувача
service	Містить набір основних класів управління даними чи компонентами
api.service	Містить набір класів для реалізації HTTP-запитів, основна мета абстрагувати логіку HTTP-запитів
storage.service	Містить набір класів для реалізації зберігання, управління, отримання локальних даних
auth.service	Містить набір класів для реалізації автентифікацію користувача
assets	Зберігання статичних ресурсів, такі як банери, зображення веб-застосунку,
environment	Містить набір конфігураційних сталих параметрів
style	Містить набір глобальних і локальних файлів управлінням стилю веб-застосунку

Головна структура поділяється на три блоки: адмін-панелі для адміністратора і модератора відповідно, окрема ієрархія сторінок для користувача.

Розглянемо детальну структуру ієрархії сторінок веб-застосунку:

Користувач

- Домашня сторінка

    - Сторінка логін
    - Сторінка реєстрації
    - Каталог
        - Під-каталог
    - Довідковий центр
    - Аккаунт
        - Замовлення
        - Особиста інформація
    - Оформлення замовлення
    - Сторінка продукту

Адміністратор

- Адмін панель
    - Додати продукт
        - Новий продукт
        - Чернетка
    - Додати каталог
    - Додати модератора (керівника)
    - Замовлення модераторів
    - Додати аптеку
        - Конфігурація аптеки

Модератор (Керівник)

- Адмін панель
    - Зробити замовлення
    - Підтвердити замовлення

Дотримуючись політики безпеки даних і доступу до функціоналу, окремі сторінки і блоки системи програми були захищені директивами, які можна накласти на будь-який компонент, а також створити свої директиви використовуючи ключове слово @Directive в Angular і передавати свої дані для керуванням зовнішнім виглядом і структурою компонента, для захисту роутінгу

були використані класи реалізуючі CanActivate. Таким чином забезпечується повний захист персональних даних і функціоналу компонентів.

Для дотримання єдиної стилистики сайту і недопущення перевантаження змісту на кожен компонент розроблений його макет і вигляд кожної сторінки, як і для користувача, так і для адміністративного блоку керування. Використання макету дозволить досягти таких цілей [66, 68]:

- Сформувати структурний і функціональний зміст кожної сторінки, а також внутрішніх компонентів. Сформувати логічний ланцюг взаємодій компонентів.
- Сформувати єдину стилістику і оригінальність веб-застосунку та продемонструвати, як будуть виглядати веб-сторінки під час розробки. До параметрів стилистики відносяться такі компоненти дизайну, як кольорова гамма, шрифти, розміщення зображень, стилі кнопок та ін. візуальні компоненти.

Розглянемо макет головної сторінки і різновид компонентів, використаних для веб-застосунку, з якими часто будуть стикатися користувачі. Макет головної сторінки задасть основний тон і стандарт веб-застосунку рис. 3.2.

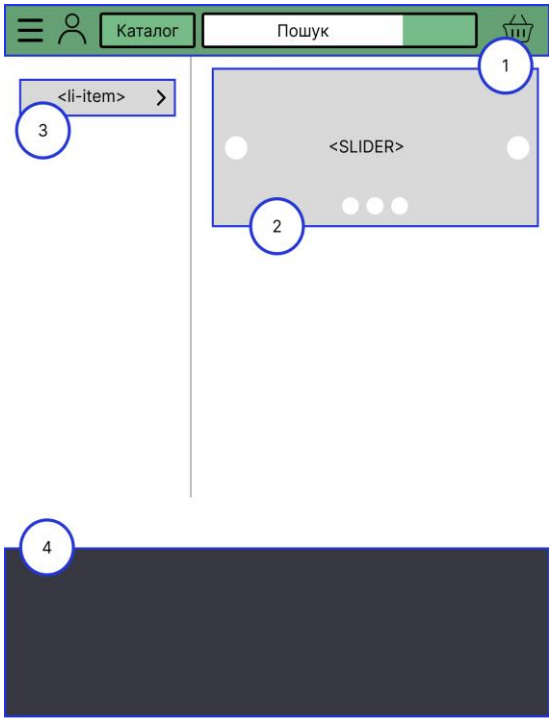

Рис. 3.2 – Макет головної сторінки

На головній сторінці веб-застосунку традиційно розміщено:

1. Верхнє навігаційне меню, з основними елементами, як
- каталог товарів,
- пошукова строка
- корзинка
- кнопка для кабінету користувача.

Також змістом і функціоналом компонента можна керувати в залежності від ролі користувача і його повноважень.

2. Карусель з банерною рекламою, яка є важливим елементом реклами; її основними функціями є привернення уваги користувачів і презентування акційних пропозицій онлайн-аптеки.
3. Каталог товарів для швидкого перегляду, представлений у вигляді компонента, який можна перевикористати в інших сторінках.
4. Нижне навігаційне меню, як і головне меню можна використати на всіх сторінках користувача.

Як каталог, так і ліки, створені адміністратором, презентуються на сторінці з усією продукцією онлайн-аптеки. Для зручного показу продукції є необхідним наявність компонента-картки, що містить зображення товару і основну інформацію для покупців, таку як: назва, ціна, знижка на продукт, функціонал додавання його то кошика покупок, збереження його до свого власного списку. Такий компонент можна перевикористати, а також керувати його змістом в залежності від ролі користувача.

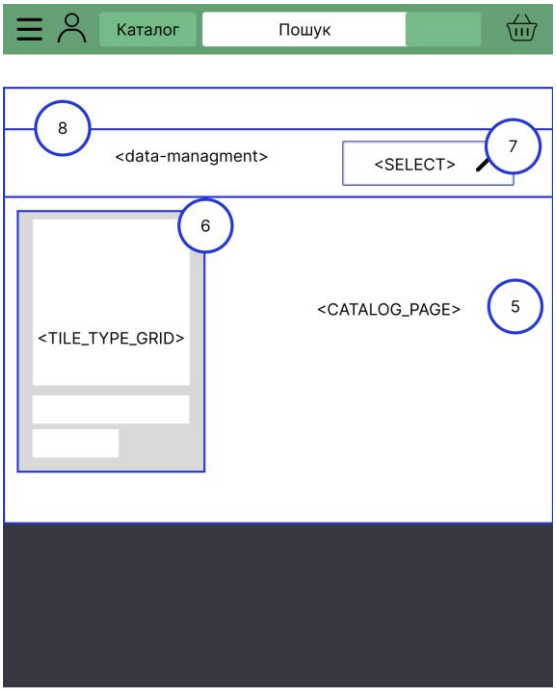

Рис. 3.3 – Макет каталогу товару

Для надання користувачу можливості фільтрації продуктів, або вибору необхідного каталогу, потрібним є наявність виборчого меню.

А для керуванням усією логікою і виглядом продуктів, корисним буде використання одного компонента, який і буде містити всі компоненти фільтрації і сортування. Макет каталогу товару представлений на рисунку 3.3

Використання спливаючих компонентів забезпечить динаміку і привернення уваги користувача. Також такі компоненти можуть містити невеликий але важливий зміст. Функціонал компонентів можна передавати до одного спливаючого діалогового меню і таким чином, уникати різновид таких компонентів.

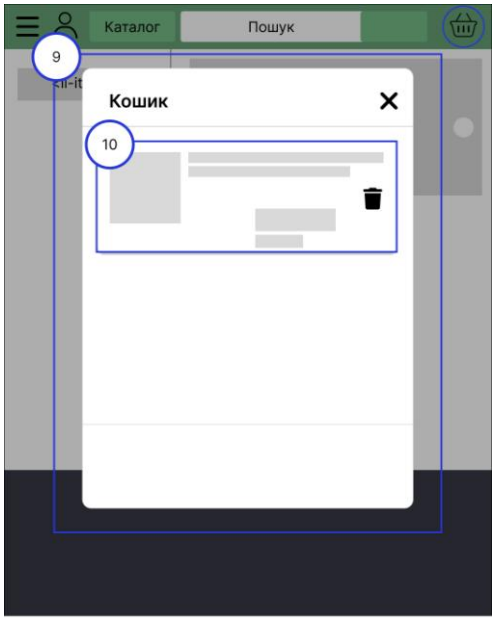

Рис. 3.4 – Макет спливаючих

Кошик, у вигляді спливаючого діалогового меню дозволяє не створювати надто багато сторінок, а також швидко повернутись до попереднього змісту.

Зміст кошику також можна виділити в окремий компонент, уникаючи завантаженість функціоналу. Компонент-картка для кошика повинна містити такий ж самий зміст для покупця. А також видалення його з кошику і зміну кількості товару.

компонентів

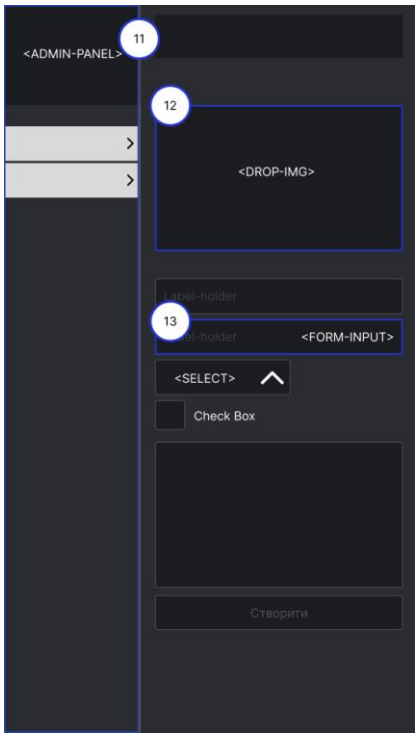

Рис. 3.5 – Макет панелі керування

Щоб відокремити роль покупців від керівного функціоналу, було вирішено змінити стилістику сайту на темну гаму рис. 3.5.

Панель навігації між різним функціоналом закріплюється на всю сторінку. Також в залежності від ролі керівників і їх повноважень, може змінюватися зміст.

Для завантаження зображень був розроблений відповідний компонент.

Для заповнення необхідних полей форми був розроблений відповідний компонент. Він також попереджає при внесенні помилок.

Для успішної реалізації користувацького дизайну веб-застосунку і здійснення ефективної стилізації його компонентів було використано сучасний стек фронтенд-технологій, в якому використовуються такі інструменти, як SCSS, Bootstrap та CSS. Використання цих технологій дозволяє втілити бажаний вигляд та функціональність нашого сайту, вирішуючи завдання щодо UX/UI [66, 68].

Маючи структуру веб-застосунку, макет головних сторінок і компонентів, розподіл на ролі, а також спроектовану систему, необхідним є приписання функціоналу для кожної ролі. У таблиці 3.3 визначені функціональні можливості системи зі сторони клієнта, у табл. 3.4 – для адміністратора.

Таблиця 3.3 – Функціонал веб-застосунку зі сторони клієнта

Назва	Опис
Авторизація	Авторизація будь-якого користувача
Реєстрація	Створення нового облікового аккаунту
Кошик	Додавання товару до списку покупок, можливість вибрати необхідну кількість лікарства
Каталог товарів	Перегляд всіх облікових ліків на складі веб-застосунку
Блок керування ліками	Включає фільтрацію, сортування і пошук по імені продукту, зміна виду продукта
Перегляд обраних ліків	Детальний опис продукта, з усіма характеристиками
Оформлення замовлення	Обрані продукти користувача можно оформити і купити

Таблиця 3.4 – Функціонал веб-застосунку зі сторони адміністратора

Назва	Опис
Додати нову позицію на склад	Форма продукта, з завантаженням зображень, додаванням характеристики продукту, назва, ціни, кількості на складі
Створення нового облікового обліку для модератора	Створення нового облікового аккаунту з певними повноваженнями
Керування каталогом	Можливість додавати, видаляти, редагувати, а також створювати нові підкатегорії
Створення нової аптеки і складу	Форма створення нової облікової аптеки, а також налаштування її
Перегляд і керування замовленнями	Можливість змінити стан замовлення, або відхилити його

Функціональні можливості системи зі сторони модератора визначені у таблиці 3.5.

Таблиця 3.5 – Функціонал веб-застосунку зі сторони модератора

Назва	Опис
Створення замовлення	Можливість обирати товар та його кількість для подальшого замовлення
Підтвердження замовлення	Підтвердження замовлення
Перегляд товарів	Перегляд товарів на складі аптеки

3.4 Експлуатація та тестування веб-застосунку для управління запасами ліків

Розроблена програмна система відповідає усім основним вимогам до управління та обліку ліків в аптеці, забезпечує увесь необхідний функціонал зі сторони безпеки: авторизація користувачів, реєстрація, перевірка валідності токена, за ролями користувачів визначається доступний функціонал програми; функціонал за клієнтом: створення облікового запису, перегляд каталогу, сортування і фільтрація, пошук за назвою ліків, додавання обраного товару до кошику; функціонал за адмінов системи: створення нових керівників системи, створення аптек і закріплення керівника-фармацевта за ним, додавання нових позицій на складі, динамічне управління каталогом, перегляд замовлень; і згідно з таблицею функціонал фармацевтика-керівника / модератора.

При відкритті програми користувач за замовчуванням попадає на головну сторінку сайту рис. 3.6. Як зазначено вище, головній сторінці властиві швидка навігація і основний функціонал програми, швидке входження у систему.

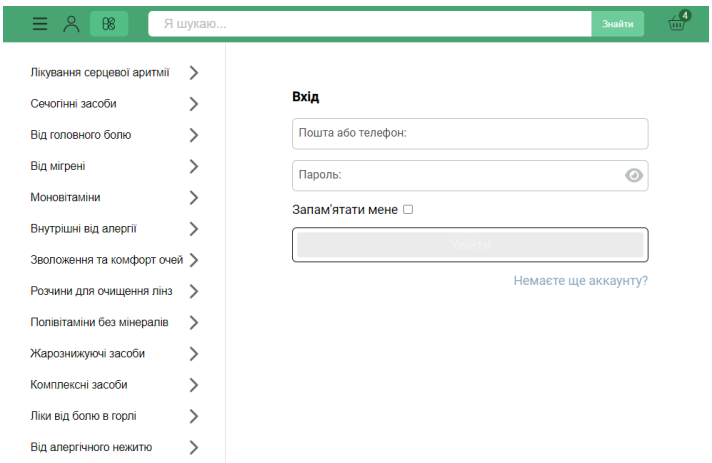

Рис. 3.6 – Головна сторінка веб-застосунку, вхід до системи

При успішній верифікації та автентифікації користувач у залежності від ролі входить до системи. На рисунку 3.7 представлена головна сторінка адмін-панелі веб-застосунку, коли користувач увійшов з роллю адміна. Початковою сторінкою є додавання нового продукту.

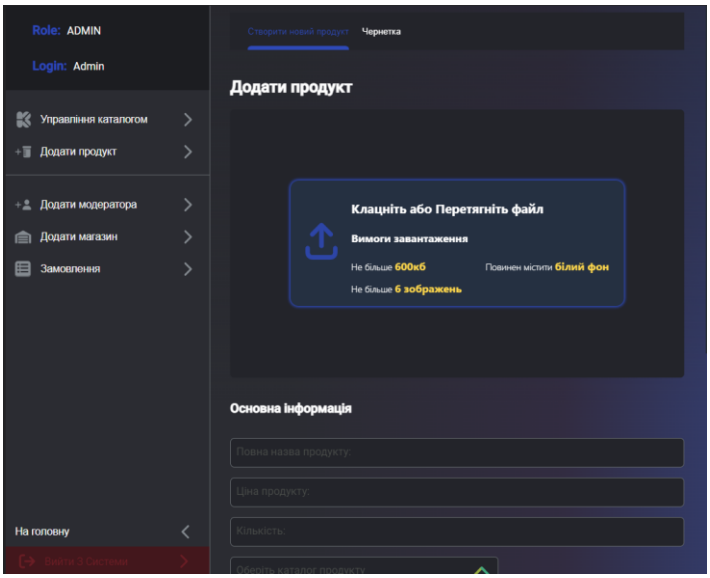

Рисунок 3.7 – Головна сторінка адмін-панелі веб-застосунку

При натисканні “додати модератора” здійснюється перехід на сторінку add-person, адміністратору відображається нова форма для створення обліку користувача з правами модератора. Як правило, потім необхідні дані для входа передаються керівнику аптеки. На рис. 3.8 представлена адмін-панель, додавання нового обліку користувача, на рис. 3.9 – успішне додавання нового обліку користувача.

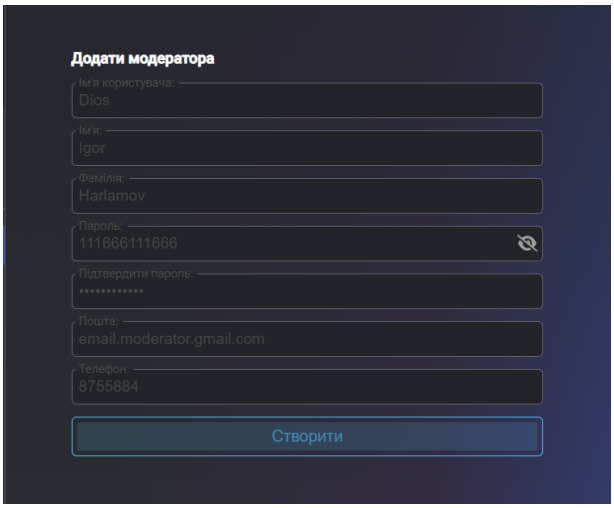

Рис. 3.8 – Адмін-панель, додавання нового обліку користувача

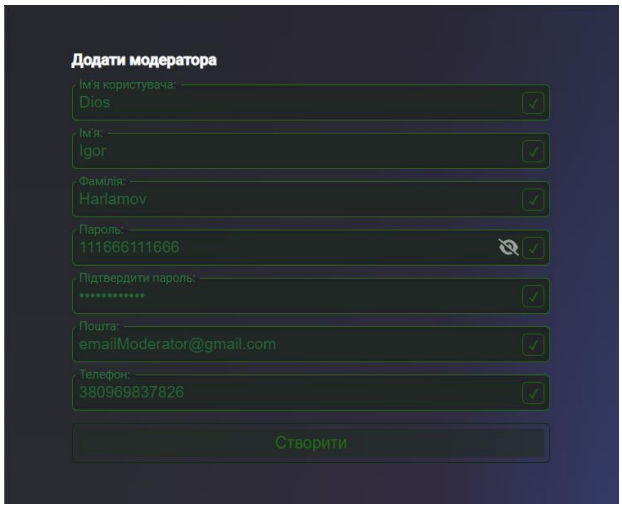

Рис. 3.9 – Адмін-панель, успішне додавання нового обліку користувача

У наступній формі створюється обліковий запис аптеки, із закріпленням керівника аптеки рис. 3.10.

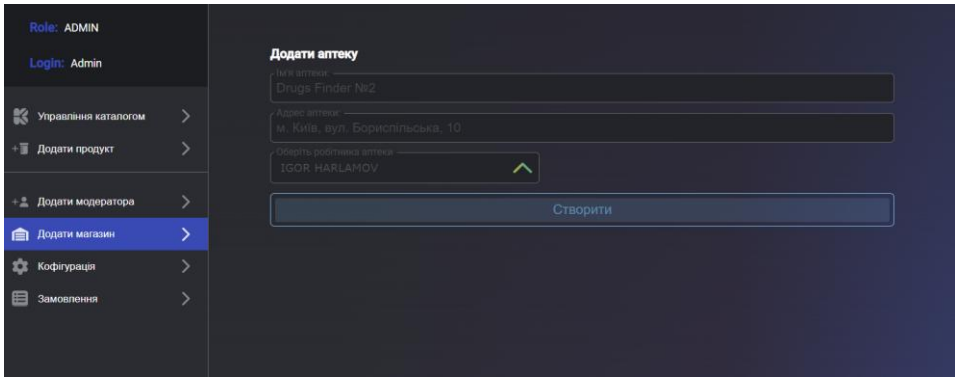

Рис. 3.10 – Адмін-панель, додавання нового обліку аптеки із закріпленням фармацевта-керівника

Після успішного створення нового обліку запису користувача і закріплення керівника за аптекою можна виконати вхід до системи рис. 3.11.

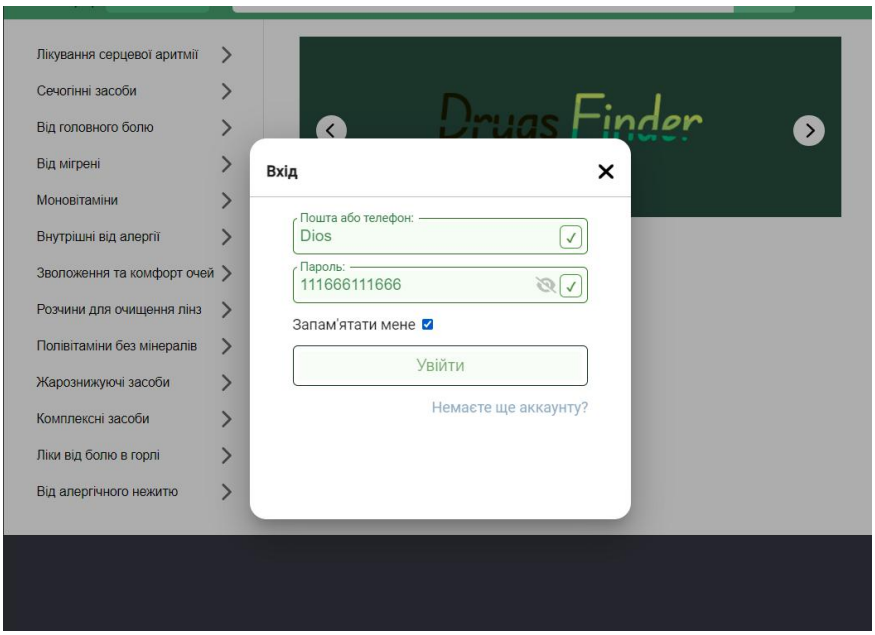

Рис. 3.11 – Головна сторінка веб-застосунку: спливаюче меню входу до системи

На рис. 3.12 представлений вхід при успішній верифікації та автентифікації користувача з правами модератора аптеки. На початковій сторінці відображено здійснення замовлення модератором відповідної аптеки для поповнення запасів ліків.

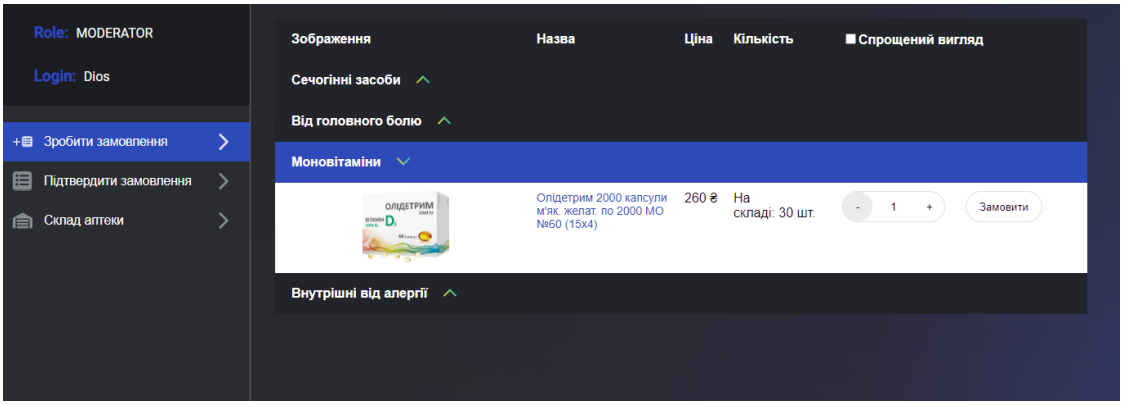

Рис. 3.12 – Адмін-панель, здійснення замовлення модератором відповідної аптеки для поповнення запасів ліків

Для забезпечення належної роботи функціональності розробленого веб-застосунку був проведений аудит системи за допомогою процесу мануального тестування, що являє собою процес перевірки функціональності та відповідності системи заданим вимогам, та передбачає ручну перевірку активності ресурсу, де тестувальник виступає у ролі користувача та взаємодіє з системою так, як це робив би справжній працівник або клієнт системи.

Основним завданням тестування є виявлення слабких місць у системі та документування помилок у звіті. Результатом тестування є успішне виконання очікуваних функцій або виявлення системної помилки.

Для перевірки правильності роботи системи було проведено кілька тестових випадків, результати яких були наведені у Додатку А. Тестування включає перевірку модулів системи, таких як вхід у систему за паролем і поштою тестувальника і фільтрацію відображених ліків в каталозі за під-категорію.

Тест ТС1 включає перевірку процесу фільтрування товарів у застосунку; Тест ТС2 - процес входу до системи модератора. Для обох тестів були описані кроки, що були виконані під час тестування, початкові умови, вхідні дані, очікувані та фактичні результати.

Отже, на підставі результатів тестування можна стверджувати про правильну роботу програмної системи.

Таким чином, в результаті виконання третього розділу кваліфікаційної роботи було проведено детальне проектування, планування, реалізація та тестування веб-застосунку. Результатом є розроблений веб-застосунок для управління запасами ліків.

ВИСНОВОК

У результаті виконання кваліфікаційної роботи бакалавра розроблено веб-застосунок для управління ліками, а саме:

- досліджено особливості побудови, функціональні можливості систем керування і обліку продукції.
- здійснено аналіз програмно-технологічних рішень для створення веб-орієнтованої системи керування ліками.
- спроектовано, реалізовано, впроваджено веб-застосунок для управління запасами ліків з урахуванням технічних вимог.

В результаті проведених досліджень були визначені основні поняття сучасних веб-застосунків, їх технічна характеристика та окреслені їх переваги. Проведено загальний аналіз рівневої архітектури веб-застосунку та приведена їх класифікація відповідно до сфери призначення. Здійснено огляд провідних вітчизняних та закордонних існуючих ERP-систем.

Проведено аналіз програмно-технічних рішень для створення веб-застосунків, зокрема визначені архітектурні підходи щодо організації програмного забезпечення і взаємодії компонентів. Надана порівняльна характеристика архітектурних рішень веб-застосунків, розглянуто основні характеристики моделей керування доступом, як основного елементу кібербезпеки. Проведений порівняльний аналіз патернів MVC та MVVM. Розглянуто використання технологій та сервісів для веб-розробки, таких як хмарні сервіси, серверний рендеринг; асинхронність подій та обробка потоків даних. Здійснено огляд та аналіз основних фреймворків для розробки серверної інфраструктури та клієнтської частини веб-застосунків. Визначені принципи роботи та основні функції БД PostgreSQL та MongoDB, проведена їх порівняльна характеристика.

На підставі проведеного аналізу програмно-технічних рішень для створення веб-застосунків для обліку ліків визначено комплекс функціональних та технічних вимог, відповідно до яких була спроектована клієнт-серверна

архітектура веб-застосунку.

Для проектування та розробки системи кваліфікаційної роботи був застосований стек технологій: клієнтська частина із застосуванням Angular MVC на мові програмування Typescript, для розмітки інтерфейсу і дизайну програми використовувався HTML і SCSS відповідно, для серверної частини програми застосовувався Spring Boot на основі Spring, для роботи з БД був використаний JPA, всі тестування API проводились в Swagger2, для DBMS був обраний PostgreSQL. Передбачено захист персональних даних користувачів системи за допомоги використання сторонніх бібліотек Spring Boot для роботи з JWT. Angular допомагає маніпулювати з відображенням компонентів за допомогою анотації директив і класу CanActivate.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a Web App? - Web Application Explained - AWS : [Електронний ресурс]. Режим доступу URL https://aws.amazon.com/ what-is/web-application/ Дата звернення (08.11.2022)
2. What is a Web Application? : [Електронний ресурс]. Режим доступу URL https://www.stackpath.com/edge-academy/what-is-a-web-application/ Дата звернення (12.11.2022)
3. What is a REST API? | IBM [Електронний ресурс]. Режим доступу URL https://www.ibm.com/topics/rest-apis Дата звернення (13.11.2023)
4. Common client-side web technologies | Microsoft Learn : [Електронний ресурс]. Режим доступу URL https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies Дата звернення (14.11.2022)
5. LinkedIn characteristics Web-application : [Електронний ресурс]. Режим доступу URL https://www.linkedin.com/pulse/characteristics-web-application-nripesh-sonavane/?trk=public_profile_article_view Дата звернення (14.11.2022)
6. Client Server Architecture: Components, Types, Benefits : [Електронний ресурс]. Режим доступу URL https://www.knowledgehut.com/blog /cloud-computing/client-server-architecture Дата звернення (18.11.2022)
7. What is Three-Tier Architecture | IBM : [Електронний ресурс]. Режим доступу URL https://www.ibm.com/topics/three-tier-architecture Дата звернення (18.11.2023)
8. N-Tier Architecture: Tier 2, Tier 3, and Multi-Tier Explained – BMC Software | Blogs : [Електронний ресурс]. Режим доступу URL https://www.bmc.com/blogs/n-tier-architecture-tier-2-tier-3-and-multi-tier-explained/ Дата звернення (19.11.2022)

9. Software Architecture: N Tier, 3 Tier, 1 Tier, 2 Tier Architecture : [Електронний ресурс]. Режим доступу URL https://www.appsierra.com blog/tiers-in-software-architecture (19.11.2022)
10. Two-tier Vs Three-tier Architecture | by Gacheru Evans | Medium : [Електронний ресурс]. Режим доступу URL https://medium.com/@gacheruevans0/2-tier-vs-3-tier-architecture-26db56fe7e9c Дата звернення (17.11.2022)
11. Different Types of Web Applications : [Електронний ресурс]. Режим доступу URL https://www.gurutechnolabs.com/types-of-web-applications/ Дата звернення (25.11.2022)
12. Плескач В. Л. Електронна комерція: Підручник / В. Л. Плескач, Т. Г. Затонацька. К.: Знання, 2007 – 535 с. Дата звернення (04.12.2022)
13. The 11 Biggest Benefits of Using a CRM System - businessnewsdaily.com : [Електронний ресурс]. Режим доступу URL https://www.business newsdaily.com/15963-benefits-of-crm.html Дата звернення (06.12.2022)
14. What is ERP? | Oracle : [Електронний ресурс]. Режим доступу URL https://www.oracle.com/erp/what-is-erp/ Дата звернення (07.12.2022)
15. What is a Web Portal? [A Complete Guide] : [Електронний ресурс]. Режим доступу URL https://www.monocubed.com/blog/what-is-web-portal/ Дата звернення (08.12.2022)
16. IT-Enterprise : [Електронний ресурс]. Режим доступу URL https://www.it.ua/ Дата звернення (12.12.2022)
17. Oracle E-Business Suite Applications : [Електронний ресурс]. Режим доступу URL https://www.oracle.com/applications/ebusiness/ Дата звернення (14.12.2022)
18. Oracle E-Business Suite Architecture : [Електронний ресурс]. Режим доступу URL https://docs.oracle.com/cd/E18727-01/doc.121/e12841/T120505T120508.htm Дата звернення (14.12.2022)
19. Open Source ERP and CRM | Odoo : [Електронний ресурс]. Режим доступу URL https://www.odoo.com/uk_UA Дата звернення (16.12.2022)

20. ERP Ukraine : [Електронний ресурс]. Режим доступу URL https://erp.co.ua/ Дата звернення (16.12.2022)
21. PWA vs. MPA vs. SPA - What's the Best Choice for Your App? | Neoteric [Електронний ресурс]. Режим доступу URL https://neoteric.eu/blog /pwa-vs-mpa-vs-spa-what's-the-best-choice-for-your-app/ Дата звернення (20.12.2022)
22. Single-page application vs Multi-page application: What Is Better for Your Project? [Електронний ресурс]. Режим доступу URL https://www.linkedin.com/pulse/single-page-application-vs-multi-page-what-better-your-?trk=organization_guest_main-feed-card_feed-article-content Дата звернення (20.12.2022)
23. Angular SPA: Why Single Page [Електронний ресурс]. – Режим доступу URL Applications?https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/ Дата звернення (25.12.2022)
24. SPA SEO: Mission Impossible? | Magnolia Headless CMS https://www.magnolia-cms.com/blog/spa-seo-mission-impossible.html Дата звернення (25.12.2023)
25. Ways of Using Vue [Електронний ресурс]. Режим доступу URL https://vuejs.org/guide/extras/ ways-of-using-vue.html Дата звернення (26.12.2023)
26. PWA vs. Native App - Which is better in 2023? [Електронний ресурс]. – Режим доступу URL https://www.sommo.io/blog/pwa-vs-native -app-which-is-better-in-2023 Дата звернення (6.01.2023)
27. Що таке керування доступом? | Захисний комплекс Microsoft [Електронний ресурс]. Режим доступу URL: https://www.microsoft.com/uk-ua/security/business/security-101/ what-is-access-control Дата звернення (08.01.2023)
28. Role-based access control with JWT| Netlify : [Електронний ресурс]. – Режим доступу URL: https://docs.netlify.com/visitor-access/ role-based-access-control/ Дата звернення (08.01.2023)

29. Access Control Models: MAC, DAC, RBAC, & PAM [Електронний ресурс]. Режим доступу URL: https://www.twingate.com/blog /other/access-control-models Дата звернення (09.01.2023)
30. What Is Attribute-Based Access Control (ABAC)? | Okta : [Електронний ресурс]. – Режим доступу URL https://www.okta.com/blog/2020/09/attribute-based-access-control-abac/ Дата звернення (09.01.202)
31. What Is Role-Based Access Control (RBAC)? A Complete Guide | Frontegg : [Електронний ресурс]. Режим доступу URL https://frontegg.com/guides/rbac Дата звернення (10.01.2023)
32. IaaS, PaaS, SaaS: Вибираємо найбільш релевантні рішення для вашого бізнесу [Електронний ресурс]. – Режим доступу: cloudfresh.com (дата звернення: 12.01.2023)
33. Cloud Computing Trends: Flexera 2022 State of the Cloud Report [Електронний ресурс]. Режим доступу: flexera.com (дата звернення: 12.01.2023)
34. What is Cloud Security? [Електронний ресурс]. – Режим доступу: kaspersky (дата звернення: 13.01.2023)
35. Azure безпека вступ [Електронний ресурс]. Режим доступу: learn.microsoft.com (дата звернення: 13.01.2023)
36. Хмарна безпека: ключові поняття, загрози та рішення [Електронний ресурс]. – Режим доступу: sgs4business.com (дата звернення: 13.01.2023)
37. Server-Side Rendering (SSR) | Vue.js : [Електронний ресурс]. Режим доступу URL https://vuejs.org/guide/scaling-up/ssr.html#what-is-ssr Дата звернення (14.01.2023)
38. Server-side rendering (SSR) with Angular Universal : [Електронний ресурс]. Режим доступу URL https://angular.io/guide/universal Дата звернення (14.01.2023)
39. Ajax - MDN Web Docs Glossary: Definitions of Web-related terms [Електронний ресурс]. – Режим доступу URL https://developer.mozilla.org/en-US/docs/Glossary/AJAX?retiredLocale=uk Дата звернення (15.01.2023)

40. RxJS [Електронний ресурс]. Режим доступу URL https://rxjs.dev Дата звернення (16.01.2023)
41. Multicasted Observables [Електронний ресурс]. Режим доступу URL: http://reactivex.io/rxjs/manual/overview.html Дата звернення (16.01.2023)
42. Angular - The RxJS library [Електронний ресурс]. Режим доступу URL: https://angular.io/guide/rx-library Дата звернення (17.01.2023)
43. Розробка на Vue.js https://avada-media.ua/ua/services/vue-js/ Дата звернення (17.01.2023)
44. Use Java for Backend Development [Електронний ресурс]. Режим доступу URL https://www.freecodecamp.org/news/java-for-backend-web-development/ Дата звернення (18.01.2023)
45. Why You Should Use Java Backend Infrastructure [Електронний ресурс]. Режим доступу URL: https://www.devteam.space/blog/why-should-you- use-java-for-your-backend-infrastructure/ Дата звернення (18.01.2023)
46. Spring Boot : [Електронний ресурс]. Режим доступу URL https://spring.io/projects/spring-boot Дата звернення (18.01.2023)
47. Advantages of Spring Boot: [Електронний ресурс]. Режим доступу URL https://www.adservi./advantages-of-spring-boot Дата звернення (18.01.2023)
48. Getting Started | Accessing Data with JPA [Електронний ресурс]. Режим доступу URL https://spring.io/guides/gs/accessing-data-jpa/ Дата звернення (19.01.2023)
49. Setting Up Swagger 2 with a Spring REST API Using Springfox [Електронний ресурс]. Режим доступу URL https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api Дата звернення (20.01.2023)
50. Angular vs React vs Vue: Which Framework to Choose in 2023 [Електронний ресурс]. Режим доступу URL: https://www.codeinwp.com/blog /angular-vs-vue-vs-react/#gref Дата звернення (21.01.2023)
51. AngularJS: Developer Guide [Електронний ресурс]. Режим доступу URL: https://docs.angularjs.org/guide Дата звернення (21.01.2023)

52. Angular Universal - How to Build SEO Friendly SPA : [Електронний ресурс]. Режим доступу URL https://blog.angular-university.io/angular-2-universal-meet-the-internet-of-the-future-seo-friendly-single-page-web-apps/ Дата звернення (21.01.2023)
53. Single-File Components | Vue.js : [Електронний ресурс]. Режим доступу URL https://vuejs.org/guide/scaling-up/sfc.html#how-it-works Дата звернення (22.01.2023)
54. БАЗИ ДАНИХ ТА ЗАСОБИ УПРАВЛІННЯ : [Електронний документ]. – Режим доступу URL https://ela.kpi.ua/bitstream/123456789/46193/1/Bazy_danykh_ta_zasoby_upravlinnia_Praktykum.pdf Дата звернення (23.01.2023)
55. PostgreSQL : [Електронний ресурс]. Режим доступу URL https://www.postgresql.org/ Дата звернення (23.01.2023)
56. PostgreSQL 15.3 Documentation : [Електронний ресурс]. Режим доступу URL https://www.postgresql.org/docs/15/index.html Дата звернення (23.01.2023)
57. MongoDB : [Електронний ресурс]. Режим доступу URL https://www.mongodb.com/ Дата звернення (24.01.2023)
58. Comparing MongoDB vs PostgreSQL : [Електронний ресурс]. Режим доступу URL https://www.mongodb.com/compare/mongodb-postgresql Дата звернення (25.01.2023)
59. Spring in Action, Sixth Edition [Електронна книга]. Режим доступу URL https://www.manning.com/books/spring-in-action-sixth-edition Дата звернення (03.02.2023)
60. Spring Boot API Security with JWT and Role-Based Authorization | by Akhilesh Anand | Medium : [Електронний ресурс]. Режим доступу URL https://medium.com/@akhileshanand/spring-boot-api-security-with-jwt-and-role-based-authorization-fea1fd7c9e32 Дата звернення (06.02.2023)
61. Auth0 Spring Boot API SDK Quickstarts: Authorization : [Електронний ресурс]. Режим доступу URL https://auth0.com/docs/quickstart/backend/java-spring-security5/01-authorization Дата звернення (06.02.2023)

62. Spring Boot security expressions for Auth0 JWT | by Aivaras Prudnikovas | Medium : [Електронний ресурс]. Режим доступу URL https://medium.com/@ivarprudnikov/spring-boot-security-expressions-for-auth0-jwt-30ac616a09f0 Дата звернення (08.02.2023)
63. JWT.io : [Електронний ресурс]. – Режим доступу URL https://jwt.io/ Дата звернення (14.02.2023)
64. Angular MVC - A Primer | DigitalOcean : [Електронний ресурс]. – Режим доступу URL https://www.digitalocean.com/community /tutorials/angular-angular-mvc-primer Дата звернення (15.02.2023)
65. Components | Angular Material : [Електронний ресурс]. Режим доступу URL https://material.angular.io/components/categories Дата звернення (20.02.2023)
66. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability [Електронна книга]. Режим доступу URL chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://eng317hannah.wordpress.ncsu.edu/files/2020/01/Krug_Steve_Dont_make_me_think_revisited___a_cz-lib.org_.pdf Дата звернення (26.02.2023)
67. MVC vs MVP vs MVVM. What's different between them? | by Anh T. Dang | Level Up Coding : [Електронний ресурс]. Режим доступу URL https://levelup.gitconnected.com/mvc-vs-mvp-vs-mvvm-35e0d4b933b4 Дата звернення (26.02.2023)
68. UX design for e-commerce: principles and strategies | Qubstudio : [Електронний ресурс]. Режим доступу URL https://qubstudio.com/blog/10-ux-design-for-e-commerce-principles-and-strategies/ Дата звернення (06.03.2023)
69. Sass: Documentation [Електронний ресурс]. Режим доступу URL: https://sass-lang.com/documentation/ Дата звернення (10.03.2023)
70. Form controls · Bootstrap v5.0 [Електронний ресурс]. Режим доступу URL: https://getbootstrap.com/docs/5.0/forms/form-control/ Дата звернення (15.03.2023)

ДОДАТКИ

ДОДАТОК А

ID Тесту: TC1

Назва: Автентифікація модератора

Опис: Перевірити роботу входу до системи

Пріоритет тесту (Низький/Середній/Високий): Високий

Розробив: Харламов І. Є.

Дата розробки: 16.02.2023

Провів: Харламов І. Є.

Дата виконання: 16.02.2023

Попередня умова: Виконавець тесту повинен бути зареєстрованим у системі. Вдала перевірка валідності форми автентифікації модератора. Всі компоненти в працюючому стані

Кроки тесту	Очікувані результати	Фактичні результати	Статус
Модератор натискає кнопку автентифікації	Відкривається форма авторизації	Очікування справдилися.	Пройдено
Модератор вводить дані для входу	Модератор вводить дані email і password форми та після натискання на кнопку “Увійти”. Отримує JWT з ролью “ROLE_MODERATOR” та його логіном. Випливає повідомлення із успішним входження системи. Перенаправлення до адмін-панелі	Очікування справдилися.	Пройдено

Стан системи після виконання тесту: Користувач отримує доступ до програми з правами модератора.

Статус: Вдалий

ID Тесту: TC2 **Розробив**: Харламов І. Є.

Назва: Фільтрація товару **Дата розробки**: 19.02.2023

Опис: Перевірити фільтрацію товару по під-категорії **Провів**: Харламов І. Є.

Пріоритет тесту (Низький/Середній/Високий): Низький **Дата виконання**: 19.02.2023

Попередня умова: Користувач перейшов до сторінки каталогу. Всі ліки успішно отримані з БД. Всі під-категорії успішно отримані з БД. Всі компоненти в працюючому стані

Кроки тесту	Очікувані результати	Фактичні результати	Статус
Користувач обирає під-категорію	Спадне меню відкрилося, вдалося вибрати бажану під-категорію	Очікування справдилися.	Пройдено
Перевірка, що ліки відфільтровані правильно	Оновився url сторінки. Успішно зчитан id каталогу. Успішно був здійснен перехід до підкаталогу.	Очікування справдилися.	Пройдено

Стан системи після виконання тесту: Користувач отримує відфільтровані ліки.

Статус: Вдалий

ДОДАТОК Б

```
@RestController
@RequestMapping("/categories")
public class CategoriesController {
    private final ParentCategoryRepository parentCategoryRepository;
    private final CategoryRepository categoryRepository;
    private final SubCategoryRepository subCategoryRepository;
    public CategoriesController(ParentCategoryRepository parentCategoryRepository,
                                CategoryRepository categoryRepository,
                                SubCategoryRepository subCategoryRepository) {
        this.parentCategoryRepository = parentCategoryRepository;
        this.categoryRepository = categoryRepository;
        this.subCategoryRepository = subCategoryRepository;
    }
    @GetMapping("/get/parent/all")
    public List<RootCategoriesDTO> getAllParentCategories() {
        List<ParentCategory> parentCategory = parentCategoryRepository.findAll();
        List<RootCategoriesDTO> rootCategoriesDTOS = parentCategory.stream()
                .map(object -> {
                    List<CategoriesDTO> categories = object.getCategories().stream()
                            .map(category -> {
                                CategoriesDTO.Builder builder = new
CategoriesDTO.Builder();
                                List<SubCategoryDTO> subCategories =
category.getSubCategories().stream().map(subCategory -> {
                                    SubCategoryDTO.Builder subCategoryBuilder = new
SubCategoryDTO.Builder()
                                            .setId(subCategory.getId())
```

```
                                        .setName(subCategory.getName());
                                return subCategoryBuilder.build();
                            }).collect(Collectors.toList());

                            builder.setName(category.getName())
                                    .setId(category.getId())
                                    .setCategories(subCategories);
                            return builder.build();
                        }).collect(Collectors.toList());
                return new RootCategoriesDTO.Builder()
                        .setName(object.getName())
                        .setSvg(object.getSvg())
                        .setId(object.getId())
                        .setCategories(categories)
                        .build();
            }).collect(Collectors.toList());
        return rootCategoriesDTOS;}
    @PostMapping("/create/parent")
    public ResponseEntity createParentCategory(@RequestBody ParentCategoryDTO
categoryDTO) {
        ParentCategory parentCategory = new ParentCategory(categoryDTO.getName(),
categoryDTO.getSvg());
        ParentCategory savedParentCategory =
parentCategoryRepository.save(parentCategory);
        return ResponseEntity.status(HttpStatus.CREATED).build();}
    @PutMapping("/update/parent/{parentId}")
    public ResponseEntity updateParentCategory(@PathVariable Long parentId,
@RequestBody ParentCategoryDTO categoryDTO) {
```

```
        Optional<ParentCategory> optionalParentCategory =
parentCategoryRepository.findById(parentId);
        if (optionalParentCategory.isPresent()) {
            ParentCategory parentCategory = optionalParentCategory.get();
            parentCategory.setName(categoryDTO.getName());
            parentCategory.setSvg(categoryDTO.getSvg());
            parentCategoryRepository.save(parentCategory);
            return ResponseEntity.ok().build();
        } else {return ResponseEntity.notFound().build();}
    }
    @DeleteMapping("/delete/parent/{parentId}")
    public ResponseEntity deleteParentCategory(@PathVariable Long parentId) {
        Optional<ParentCategory> optionalParentCategory =
parentCategoryRepository.findById(parentId);
        if (optionalParentCategory.isPresent()) {
            ParentCategory parentCategory = optionalParentCategory.get();
            parentCategoryRepository.delete(parentCategory);
            return ResponseEntity.ok().build();
        } else {return ResponseEntity.notFound().build();}
    }
    @GetMapping("/get/categories/all")
    public List<Category> getAllCategories() {return categoryRepository.findAll();}
    @PostMapping("/create/category")
    public ResponseEntity createCategory(@RequestBody ParentCategoryDTO categoryDTO) {
        Category category = new Category();
        category.setName(categoryDTO.getName());
        ParentCategory parentCategory =
parentCategoryRepository.findById(categoryDTO.getParent_category_id()).orElse(null);
        if (parentCategory == null) {return ResponseEntity.badRequest().build();}
```

```
        category.setParentCategory(parentCategory);
        Category savedCategory = categoryRepository.save(category);
        return ResponseEntity.status(HttpStatus.CREATED).build(); }

    @PutMapping("/update/category/{id}")
    public ResponseEntity updateCategory(@PathVariable("id") Long id, @RequestBody
ParentCategoryDTO categoryDTO) {
        Optional<Category> optionalCategory = categoryRepository.findById(id);
        if (optionalCategory.isPresent()) {
            Category category = optionalCategory.get();
            category.setName(categoryDTO.getName());
            ParentCategory parentCategory =
parentCategoryRepository.findById(categoryDTO.getParent_category_id()).orElse(null);
            if (parentCategory == null) {return ResponseEntity.badRequest().build();}
            category.setParentCategory(parentCategory);
            categoryRepository.save(category);
            return ResponseEntity.ok().build();
        } else {return ResponseEntity.notFound().build();}
    }
    @DeleteMapping("/delete/category/{id}")
    public ResponseEntity deleteCategory(@PathVariable("id") Long id) {
        Optional<Category> optionalCategory = categoryRepository.findById(id);
        if (optionalCategory.isPresent()) {
            categoryRepository.deleteById(id);
            return ResponseEntity.ok().build();
        } else {return ResponseEntity.notFound().build();}
    }
    @GetMapping("/get/subcategories/all")
    public List<SubCategoryDTO> getAllSubCategories() {
```

```
    List<SubCategory> subCategories = subCategoryRepository.findAll();
    List<SubCategoryDTO> subCategoryDTOS = subCategories.stream()
            .map(subCategory -> {
                SubCategoryDTO.Builder builder = new SubCategoryDTO.Builder()
                        .setId(subCategory.getId())
                        .setName(subCategory.getName());
                return builder.build();
            })
            .collect(Collectors.toList());
    return subCategoryDTOS;
}
```