ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для навігації по магазину на основі списку товарів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Андрій МАКАРОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Андрій МАКАРОВ

Керівник: Ірина ЗАМРІЙ
д.т.н., професор

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Макарову Андрію Дмитровичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для навігації по магазину на основі списку товарів»
керівник кваліфікаційної роботи д.т.н., професор Ірина ЗАМРІЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про алгоритми пошуку шляху в графах, моделі представлення просторової структури магазину у вигляді графа, методи оптимізації маршруту на основі списку цільових об’єктів, технічна документація з описом фреймворку Spring та бібліотек для розробки вебзастосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз сучасних підходів і технологій побудови маршрутів у приміщеннях на основі списку товарів.
2. Проєктування програмного забезпечення для навігації по магазину на основі списку товарів.

3. Програмна реалізація та документування серверного застосунку для побудови маршруту по магазину.
4. Тестування роботи програмного забезпечення на основі типових сценаріїв використання.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма основних варіантів використання.
5. Діаграма модулів.
6. Діаграма класів доменного модуля.
7. Діаграма потоку аутентифікації.
8. ER-діаграма.
9. Апробація результатів дослідження

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз сучасних підходів і технологій побудови маршрутів у приміщеннях на основі списку товарів	14.03-20.03.2025	
4	Проєктування програмного забезпечення для навігації по магазину на основі списку товарів	21.03-31.03.2025	
5	Програмна реалізація та документування серверного застосунку для побудови маршруту по магазину	01.04-20.04.2025	
6	Тестування роботи програмного забезпечення на основі типових сценаріїв використання	21.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Андрій МАКАРОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 64 стор., 1 табл., 12 рис., 26 джерел.

Мета роботи – спрощення навігації покупців у магазині шляхом побудови маршруту на основі списку товарів.

Об'єкт дослідження – процес орієнтації покупця в просторі магазину під час здійснення покупок.

Предмет дослідження – програмне забезпечення для навігації по магазину.

Короткий зміст роботи: в роботі проаналізовано алгоритми та методи пошуку найкоротших маршрутів у графах у контексті задачі навігації по магазину на основі списку товарів. Проаналізовано інструментальні засоби для побудови серверних застосунків. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: створення структури магазину, додавання товарів, побудова графа з полицями, визначення найкоротшого маршруту між точками та генерація маршруту відповідно до списку покупок. Проведено функціональне та модульне тестування застосунку. У роботі використано Java, Spring Boot для реалізації прикладної логіки, а також відповідні бібліотеки для побудови REST-сервісу та серіалізації даних.

Сферою використання застосунку є автоматизація процесу навігації покупця в магазині на основі попередньо сформованого списку товарів.

КЛЮЧОВІ СЛОВА: НАВІГАЦІЯ, МАРШРУТ, ГРАФ, СПИСОК ТОВАРІВ, JAVA, SPRING BOOT, SPRING SECURITY, SPRING DATA, POSTGRESQL, LIQUIBASE, OAUTH2, JWT, DOCKER, HTML, CSS, JAVASCRIPT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GPS – Global Positioning System

BLE – Bluetooth Low Energy

API – Application Programming Interface

REST – Representational State Transfer

JWT – JSON Web Token

JSON – JavaScript Object Notation

AR – Augmented Reality

ERP – Enterprise Resource Planning

POS – Point of Sale

WMS – Warehouse Management System

JWM – Java Virtual Machine

JPA – Java Persistence API

HTTP – Hypertext Transfer Protocol

URL – Uniform Resource Locator

SQL – Structured Query Language

DTO – Data Transfer Object

CI – Continuous Integration

CD – Continuous Deployment

СКБД – Система Керування Базами Даних

ВСТУП

Актуальність: у великих торгівельних закладах покупці часто витрачають зайвий час на пошук потрібних товарів через складну навігацію. Це знижує зручність покупок та ефективність обслуговування. Впровадження програмного забезпечення для побудови оптимального маршруту за списком товарів дозволяє спростити процес орієнтації, скоротити час перебування в магазині та покращити якість обслуговування клієнтів. Тема є актуальною в умовах цифровізації торгівлі та зростання попиту на зручні інструменти для покупців.

Об'єктом дослідження є процес орієнтації покупця в просторі магазину під час здійснення покупок.

Предметом дослідження є програмне забезпечення для навігації по магазину.

Метою роботи є спрощення навігації покупців у магазині шляхом побудови маршруту на основі списку товарів.

Методи дослідження: першим кроком було проаналізовано існуючі рішення для навігації у фізичних просторах, зокрема системи внутрішньої навігації та мобільні застосунки супермаркетів, для формулювання початкових функціональних та нефункціональних вимог до власного програмного забезпечення.

Далі було проведено огляд технологічного стеку, здатного забезпечити ефективну реалізацію системи. Для розробки серверної частини було обрано фреймворк Spring Boot, як мову програмування – Java, а для зберігання даних – реляційну базу даних PostgreSQL. GitHub використовувався як система контролю версій, а Postman – для тестування REST API. Дослідження архітектури та реалізації застосунку здійснювалося безпосередньо під час етапів проєктування та розробки.

Наукова новизна роботи визначається наступними функціональними складовими: побудова внутрішньої структури магазину у вигляді графа; реалізація алгоритму пошуку найкоротшого маршруту до всіх товарів із врахуванням точки входу; автоматизоване формування шляху проходження через магазин за списком товарів. Застосунок виступає як ефективний інструмент для покращення досвіду покупців.

Практична значущість результатів полягає у можливості використання розробленого застосунку в роздрібній торгівлі для оптимізації маршруту покупця, зменшення часу перебування в магазині та підвищення рівня зручності при здійсненні покупок.

1 АНАЛІЗ ЗАДАЧІ НАВІГАЦІЇ У СУПЕРМАРКЕТІ

Організація ефективної навігації покупців у межах супермаркету є важливою складовою підвищення зручності та швидкості здійснення покупок. В умовах розширення асортименту та зростання розмірів торгівельних площ, покупці стикаються з труднощами пошуку необхідних товарів. Це створює потребу в спеціалізованих цифрових рішеннях, які можуть забезпечити інтелектуальний супровід процесу купівлі – зокрема, у вигляді побудови оптимального маршруту по магазину на основі списку товарів.

Проблематика полягає в тому, що більшість покупців змушені витрачати значний час на пошук необхідних товарів через складну або незрозумілу структуру розміщення полиць, відсутність чіткої схеми переміщення та загальну неінформованість про просторову організацію торгового залу. З огляду на зростання цифровізації торгівлі, актуальним є впровадження інструментів, які не лише надають перелік товарів, а й забезпечують інтелектуальну підтримку їхнього пошуку всередині приміщення магазину.

Метою даного розділу є вивчення поточного стану сфери програмного забезпечення для навігації по магазинах, виявлення ключових проблем, з якими стикаються користувачі та власники супермаркетів, а також аналіз наявних рішень та їхніх обмежень. Такий аналіз дозволить обґрунтувати необхідність створення нової системи та сформувати вимоги до її архітектури й функціональності.

1.1 Типовий процес здійснення покупок у супермаркеті

Процес здійснення покупок у супермаркеті складається з кількох послідовних етапів, які включають вхід у магазин, пошук і вибір товарів, формування кошика, проходження до каси та завершення покупки. Незважаючи на

уявну простоту, цей процес часто супроводжується труднощами, особливо у великих супермаркетах із заплутаним плануванням торгового залу.

У більшості магазинів товари розміщуються за категоріями: продукти харчування, побутова хімія, напої, засоби гігієни тощо. При цьому немає універсального стандарту щодо послідовності чи логіки їхнього розташування. Зміни у плануванні, переміщення товарів або поява нових категорій ще більше ускладнюють орієнтування.

Зазвичай покупець самостійно формує список товарів (в паперовому чи цифровому вигляді) і намагається знайти відповідні позиції в залі. При цьому виникає потреба в постійному поверненні до попередніх зон, якщо певний товар був пропущений або не знайдений одразу. У години пік скупчення людей у вузьких проходах також погіршує якість досвіду покупця.

Важливим чинником є й обмеження в часі: багато споживачів бажають зробити покупки швидко, без зайвих затримок. Особливо це актуально для щоденного або тижневого поповнення запасів, коли клієнти мають фіксований перелік потреб і очікують максимально зручного шляху їх реалізації.

Системи навігації покликані мінімізувати неефективні переміщення, зменшити кількість повторних проходжень повз ті самі полиці та сприяти швидшому виконанню завдання. Вони здатні враховувати просторову структуру залу, розміщення товарів, точки входу й виходу, а також динаміку змін у внутрішній логістиці супермаркету.

Таким чином, типовий сценарій здійснення покупок без цифрової навігаційної підтримки є неефективним, непередбачуваним і залежним від суб’єктивного досвіду покупця. Це створює передумови для впровадження спеціалізованого програмного забезпечення, яке б виконувало роль інтелектуального асистента у процесі проходження торговим залом.

1.2 Проблеми, що виникають без навігаційної підтримки

У супермаркетах із великою кількістю товарів та складним плануванням відсутність цифрової навігаційної підтримки створює низку суттєвих проблем для покупців. Однією з найпоширеніших є нераціональне проходження торговим залом. Покупці нерідко змушені повертатися до вже відвіданих зон, оскільки певні товари можуть бути пропущені або їх розташування виявилось неочікуваним. Це призводить до подовження часу покупки та виникнення додаткових фізичних навантажень, особливо у великих магазинах.

Ще однією проблемою є труднощі з пошуком рідковживаних товарів, які можуть знаходитись поза основними маршрутами або в нетипових місцях. Навіть наявність вивісок чи зонування не гарантує легкості пошуку, оскільки розміщення часто змінюється внаслідок внутрішніх ротацій асортименту.

Відсутність структурованого маршруту також спричиняє зниження загального рівня задоволеності процесом покупки. Покупці витрачають більше часу, відчувають роздратування, особливо під час годин пік, коли проходи можуть бути перевантажені. Це, у свою чергу, може негативно впливати на репутацію торгової точки та спонукати споживача шукати альтернативу.

З боку адміністрації магазину ситуація також ускладнюється: хаотичний рух клієнтів створює нерівномірне навантаження на зони магазину, ускладнює логістику персоналу, та підвищує ймовірність утворення заторів, які знижують пропускну здатність торгового залу.

Таким чином, відсутність цифрових інструментів навігації породжує як суб’єктивні труднощі для покупця, так і об’єктивні втрати для магазину. Це підкреслює доцільність розробки програмного забезпечення, яке забезпечує підтримку процесу переміщення користувача торговим залом.

1.3 Існуючі технологічні рішення

Сучасні технологічні рішення, що застосовуються у сфері внутрішньої навігації в магазинах, мають на меті полегшити орієнтацію користувача в просторі та забезпечити ефективну побудову маршруту до необхідних товарів. Ці рішення можна умовно поділити на дві основні категорії: апаратні та програмні.

До апаратних технологій належать такі підходи:

– GPS позиціонування – традиційна система глобального позиціонування, яка, однак, практично не застосовується у приміщеннях через слабкий сигнал. Тому її використання у супермаркетах є обмеженим;

– BLE-маячки – розміщуються у приміщеннях магазину та дозволяють визначати приблизну позицію користувача через взаємодію з мобільним пристроєм. Це дозволяє реалізовувати навігацію з точністю до кількох метрів;

– Wi-Fi трекінг – подібний до BLE, проте використовує сигнали бездротових мереж. Вимагає налаштування інфраструктури та не завжди дає достатню точність;

– візуальні маркери – розміщуються в торговому залі, а мобільний застосунок зчитує їх через камеру для визначення місцезнаходження.

Програмні рішення здебільшого реалізуються у вигляді:

– мобільних застосунків – забезпечують користувачів візуальним відображенням карти магазину, функцією формування списку товарів, підказками щодо їхнього розташування. Такі застосунки можуть також надавати інформацію про знижки, наявність товарів або альтернативи;

– інтерактивних терміналів – встановлюються на вході або у ключових точках магазину. Покупець може ввести потрібний товар і побачити його місце розташування на мапі;

– вебінтерфейсів – менш поширені у супермаркетах, але можуть застосовуватись у гіпермаркетах чи мережах, які пропонують попереднє планування покупок онлайн.

Попри широкий вибір технічних підходів, більшість із них мають низку обмежень. Зокрема, апаратні рішення вимагають додаткових витрат на закупівлю, налаштування та обслуговування обладнання. Крім того, їх застосування не завжди ефективне в умовах швидко змінюваного внутрішнього середовища супермаркету.

Програмні рішення, хоча й менш витратні, потребують точного відображення структури магазину, актуальної інформації про розташування товарів, а також підтримки з боку персоналу. Вони повинні бути максимально гнучкими, масштабованими та адаптивними до змін у плануванні торгового простору. Таким чином, вибір конкретного рішення залежить від технічних і фінансових можливостей магазину, а також очікувань його клієнтів.

Подальші підрозділи розглянуть приклади реалізації таких рішень у вигляді конкретних програмних продуктів, а також визначать їх переваги й недоліки в контексті завдань, поставлених у межах цієї роботи.

1.4 Аналіз аналогів

Розглянемо найбільш показові приклади програмних рішень, які так чи інакше реалізують або частково підтримують функції навігації в межах магазину, формування списків покупок та взаємодії з користувачем.

Walmart App – мобільний застосунок американської торгівельної мережі Walmart, який вважається одним з найрозвиненіших рішень у сфері внутрішньої навігації. Його функціонал дозволяє не лише додавати товари до списку покупок, а й отримувати точне розташування кожного товару у конкретному магазині. Система враховує наявність продукції в магазині та оновлює карту в режимі реального часу. Додатковою перевагою є глибока інтеграція з внутрішніми обліковими системами мережі [1]. На рисунку 1.1 проілюстровані екранні форми даного застосунку.

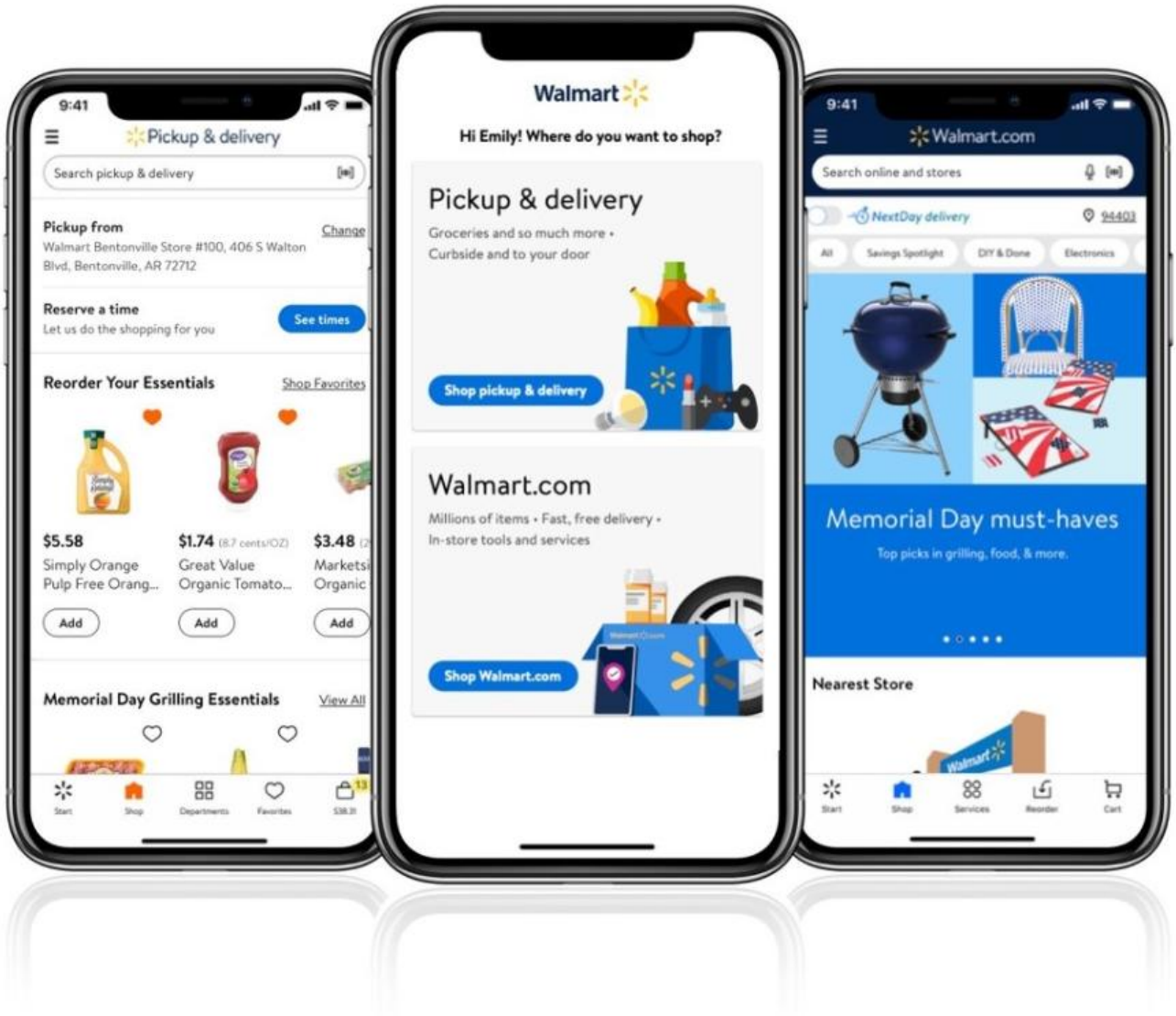


Рис. 1.1 Екранні форми Walmart App

Mapsted – рішення канадської компанії, яке вирізняється унікальною технологією позиціонування без використання GPS, маячків або Wi-Fi. Завдяки використанню власного алгоритму визначення координат, система забезпечує точне позиціювання користувача всередині приміщення навіть за відсутності спеціального обладнання. Основними сферами застосування є великі торгові центри, аеропорти, а також великі магазини. У межах таких приміщень Mapsted дозволяє будувати маршрут з високою точністю, не потребуючи зовнішніх сенсорів [2]. На рисунку 1.2 зображений приклад застосунку на основі рішення Mapsted.

Рис. 1.2 Приклад навігації за допомогою Mapsted

Target App – мобільний застосунок великої американської мережі Target. Він дозволяє користувачам створювати списки покупок, а також надає доступ до інтерактивної карти магазину. Карта містить позначення зон із відповідними товарами. Користувачі можуть перевірити наявність товару, ціни, а також ознайомитися з діючими знижками та акціями [3]. Однак система позиціонування та побудова маршруту реалізована менш гнучко порівняно з Walmart або Mapsted. На рисунку 1.3 зображено екранні форми цього застосунку.

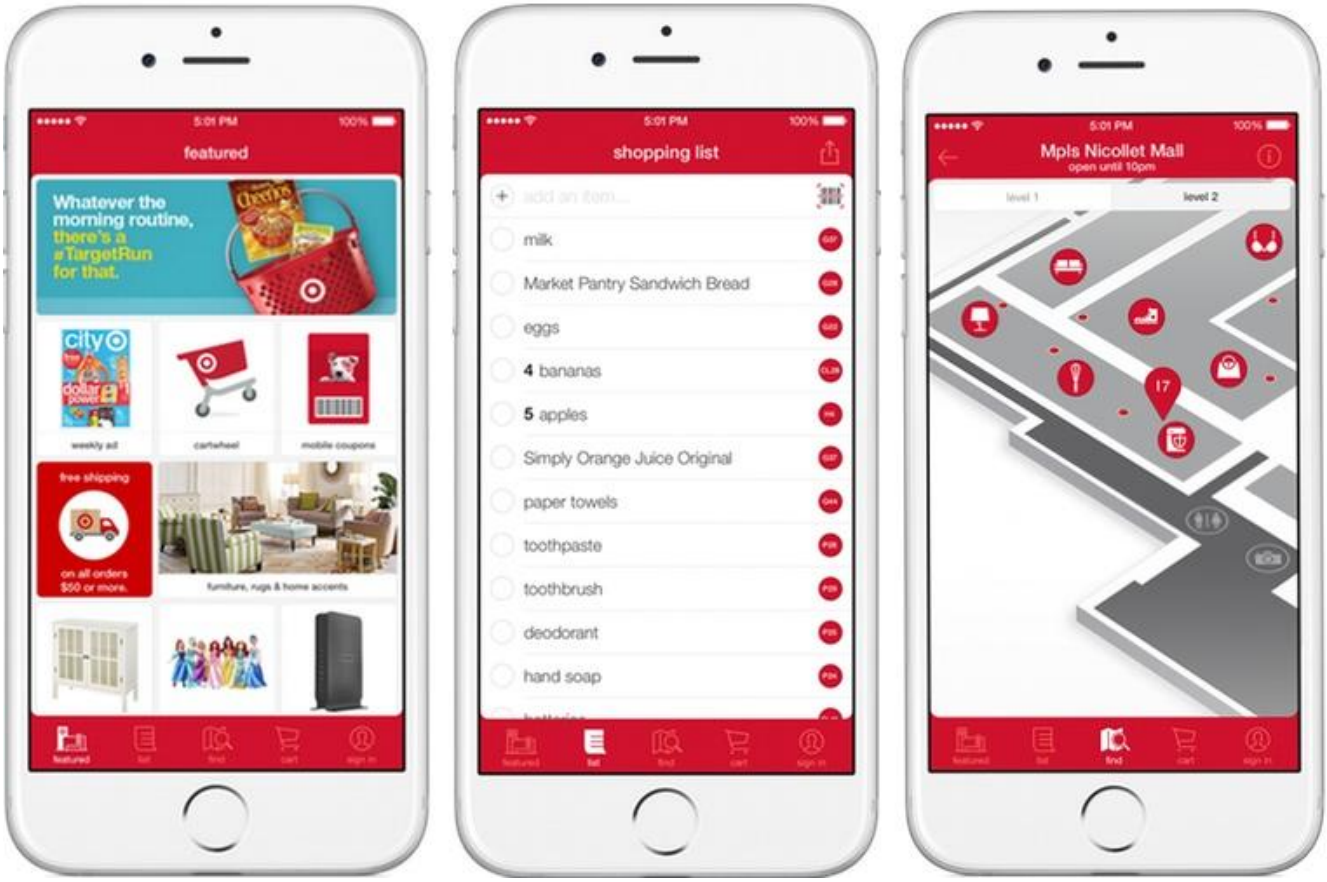


Рис. 1.3 Екранні форми Target App

Bring! Shopping List – європейський мобільний застосунок, що фокусується на створенні та синхронізації списків покупок для кількох користувачів. Основна функціональність стосується роботи зі списками, інтеграції з брендами та нагадуваннями. Навігаційні функції відсутні, однак застосунок є поширеним інструментом в європейських країнах і активно підтримується. Його перевага — мультиплатформеність і простота використання [4]. На рисунку 1.4 продемонстровано екранні форми даного застосунку.

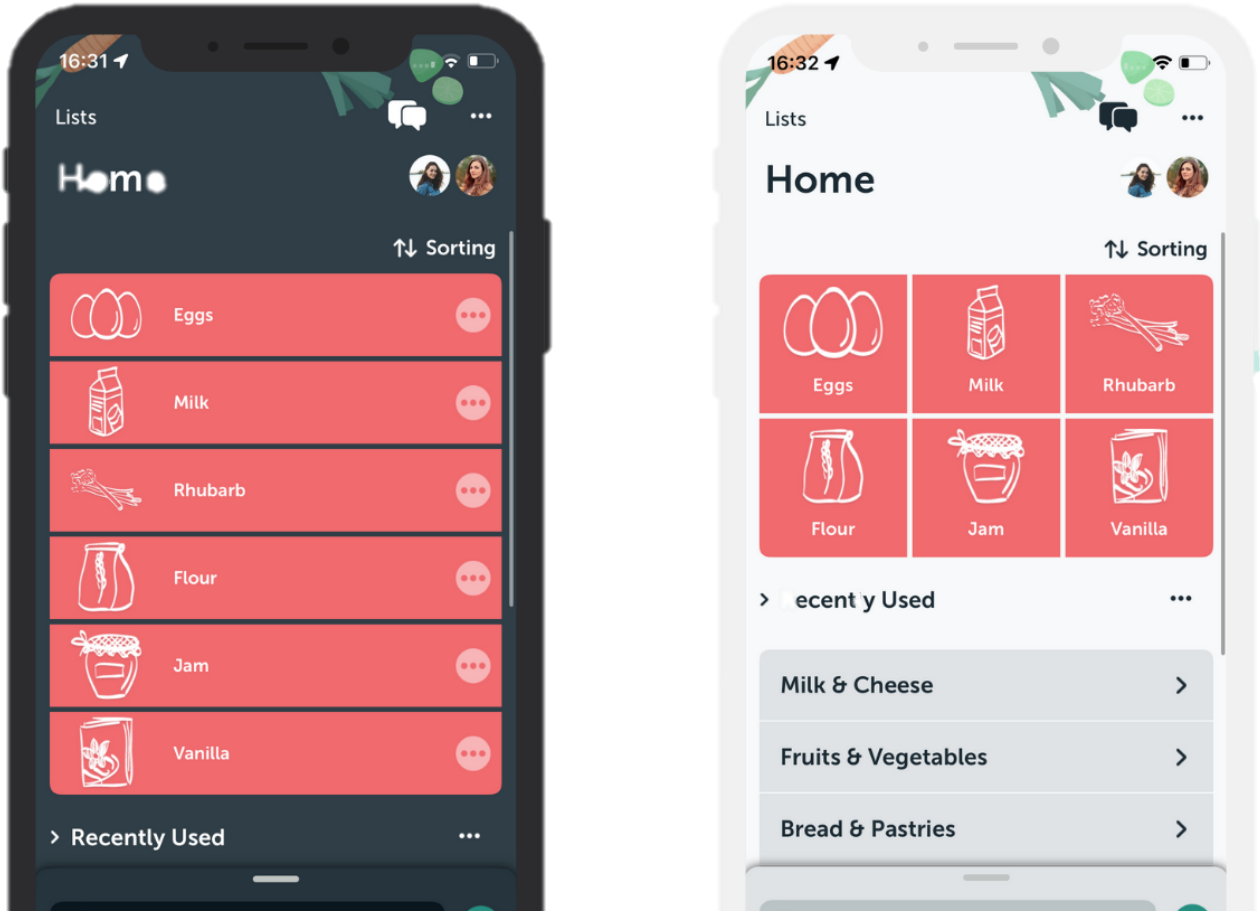


Рис. 1.4 Екранні форми Bring! Shopping List

Загальний аналіз вказує на наявність спільних обмежень. Більшість програм не враховує фізичну структуру магазину як граф чи координатну модель. Деякі з них не мають відкритого API або SDK, що ускладнює можливість інтеграції або кастомізації під конкретну торгівельну мережу. Лише окремі рішення надають можливість побудови точного маршруту на основі фактичного розташування товарів.

Зведені результати аналізу аналогів наведено у таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця аналізу програмних засобів

Параметр	**Walmart App**	**Mapsted SDK**	**Target App**	**Bring! Shopping List**
Побудова маршруту	+	+	+	-
Локалізація товарів	+	+	+	-
Доступність в Україні	-	-	-	+
Підтримка списків	+	+	+	+
Платформа	Android/iOS	Android/iOS	Android/iOS	Android/iOS/Web

1.5 Роль супермаркету як цифрового простору

Супермаркет у сучасному світі перестає бути лише фізичним місцем для купівлі товарів. Він трансформується у цифровий простір, який поєднує фізичну інфраструктуру із інформаційними системами, що забезпечують збір, обробку та використання даних для оптимізації роботи магазину і покращення досвіду покупців.

Цифровий супермаркет включає в себе електронні каталоги товарів, системи обліку запасів, платформи для взаємодії з клієнтами, а також технології навігації та аналітики переміщень у межах торгового залу.

Однією з ключових складових цифрового супермаркету є структурована інформаційна модель, що відображає просторову організацію приміщення у вигляді графа або інших моделей даних. Ця модель включає полиці, входи, каси та проходи між ними, що дозволяє здійснювати ефективне маршрутування та динамічне оновлення інформації про розташування товарів.

Інтеграція інформаційної моделі з системами обліку дає змогу відслідковувати наявність товарів у режимі реального часу та швидко оновлювати маршрути, враховуючи зміни у викладенні товарів або розміщенні полиць.

Також цифровий простір відкриває нові можливості для аналітики поведінки покупців, що дозволяє оптимізувати планування торгових зон, маркетингові стратегії та персоналізовані пропозиції.

Загалом, цифровий супермаркет — це інтегрована система, яка поєднує фізичні ресурси та інформаційні технології для створення зручного, ефективного та адаптивного середовища для покупців і адміністрації магазину.

1.6 Представлення карти магазину в цифровій формі

Ефективна навігація по супермаркету потребує точного й структурованого представлення простору магазину у цифровому форматі. Карта магазину слугує основою для побудови маршрутів, пошуку товарів і управління інформацією про розташування полиць та продуктів. Існують різні підходи до представлення карти, кожен із яких має свої переваги та обмеження.

1.6.1 Види моделей для представлення простору магазину

1.6.1.1 Графова модель

Графова модель є найбільш поширеним і ефективним способом представлення внутрішніх просторів магазинів. У такій моделі магазину за вершини графа відповідають полиці, входи у магазин, каси або перехрестя проходів. Ребра графа представляють проходи або зв’язки між цими точками.

Переваги:

– чітка структура для алгоритмів пошуку найкоротшого шляху;

– можливість легко додавати нові елементи або змінювати структуру;

– зручність для відображення складних конфігурацій магазинів.

Обмеження:

– потрібно підтримувати актуальність даних при зміні розташування полиць;

– необхідність точного задання координат або ваг ребер для коректної роботи маршрутизації.

1.6.1.2 Матриця суміжності

Матриця суміжності — це двовимірний масив, де кожен елемент позначає наявність або вагу ребра між двома вершинами графа.

Переваги:

– простота реалізації та швидкий доступ до інформації про наявність ребер між вершинами;

– зручність для невеликих мереж із стабільною кількістю вузлів.

Недоліки:

– неефективність для великих магазинів через квадратичний розмір матриці;

– високе споживання пам’яті при збільшенні кількості полиць;

– складність масштабування при динамічних змінах структури магазину.

1.6.1.3 Координатна сітка

Поділ торгового залу на координатну сітку, де кожна клітинка відповідає певній зоні або частині приміщення.

Переваги:

– простота обробки руху та орієнтації в просторі;

– застосування в алгоритмах пошуку шляху з урахуванням перешкод;

– легкість у відображенні на картах із регулярною розбивкою.

Недоліки:

– втрата точності при моделюванні дрібних деталей розташування товарів;

– велика кількість клітинок призводить до зростання обчислювальних ресурсів;

– не завжди ефективна для нестандартних планувань або складної архітектури магазину.

1.6.2 Особливості збереження карти магазину

Цифрове подання простору супермаркету вимагає не лише ефективної моделі представлення, але й належної організації збереження цієї моделі у базі даних. Для підтримки навігаційної логіки система повинна зберігати детальну інформацію про структуру магазину, зв’язки між його елементами, а також про розташування товарів.

У рамках проєктування інформаційної моделі слід передбачити збереження наступних типів даних:

– координоване представлення об’єктів – основними просторовими елементами магазину є полиці, вхідні та вихідні зони, а також касові зони. Кожен із цих об’єктів повинен мати унікальний ідентифікатор та координати, що дозволяють розмістити його на цифровій карті. Таке координатне представлення є універсальним і дозволяє побудувати гнучку карту з можливістю масштабування;

– модель зв’язків між об’єктами – для побудови маршрутів необхідно зберігати інформацію про зв’язки між просторовими точками. Кожне ребро має з’єднувати дві точки та мати вагу, наприклад, відстань або умовний час проходження. Це забезпечує можливість реалізації алгоритмів пошуку найкоротших шляхів між вузлами;

– відображення структури магазину – магазин як логічна одиниця повинен мати власні характеристики: назву, координати входу та кас, а також зв’язок із відповідними полицями та іншими зонами. Це дозволить відокремити карти різних торгових точок і реалізувати підтримку кількох магазинів у межах одного застосунку;

– зв’язок товарів із просторовими зонами – кожен товар повинен бути прив’язаний до конкретної полиці або зони. У базі даних це може бути реалізовано через окрему таблицю відповідностей між товарами та полицями. У перспективі це дозволить здійснювати фільтрацію товарів за їхнім розташуванням та обчислювати маршрут через пункти, де вони знаходяться;

– можливість масштабування та оновлення – модель зберігання повинна бути гнучкою до змін. Зокрема, необхідно передбачити можливість: додавання та видалення полиць, зміни топології проходів, оновлення координат.

Таким чином, зберігання карти магазину має ґрунтуватися на реляційній структурі з чітко визначеними зв’язками між сутностями. Це дозволить ефективно управляти даними про простір, підтримувати актуальність навігаційної інформації та

забезпечити коректну роботу системи побудови маршрутів. У наступних розділах буде запропонована конкретна реалізація цієї моделі у вигляді структури бази даних.

1.6.3 Виклики і перспективи

Цифрове представлення простору магазину є критично важливим етапом для побудови навігаційних рішень, проте його реалізація супроводжується низкою технічних, організаційних і концептуальних викликів. Успішне подолання цих викликів дозволить створити систему, що буде не лише зручною для користувача, але й гнучкою, адаптивною та масштабованою.

1.6.3.1 Основні виклики

У процесі розробки системи цифрової навігації по супермаркету виникають низка серйозних технічних та концептуальних викликів. Вони пов’язані як із характером фізичного середовища, так і з вимогами до точності, актуальності та продуктивності системи. Нижче наведено ключові проблеми, які необхідно враховувати під час проєктування та масштабування такого рішення:

– необхідність високої точності даних – для коректної роботи алгоритмів побудови маршрутів потрібна висока точність у визначенні координат елементів магазину: полиць, входу, кас, проходів тощо. Навіть незначна похибка у розміщенні об’єкта на цифровій мапі може призвести до некоректного маршруту або недоступності певного товару;

– часті зміни фізичного планування магазину – у супермаркетах періодично відбуваються зміни в розташуванні товарів або полиць. Система повинна мати можливість оперативно відображати ці зміни, інакше користувачі отримуватимуть застарілі або хибні маршрути. Це створює вимоги до інструментів оновлення карти та її синхронізації з внутрішніми процесами магазину;

– баланс між деталізацією і продуктивністю – занадто детальна карта з великою кількістю координат, зон і зв’язків збільшує навантаження на систему, зокрема при виконанні маршрутних запитів або візуалізації шляху. З іншого боку, надто узагальнена модель не забезпечить достатньої точності. Важливо знайти баланс між обсягом інформації та швидкістю її обробки, щоб система залишалася ефективною;

– масштабування та підтримка кількох магазинів – у разі підтримки мережі супермаркетів система повинна враховувати відмінності у плануванні кожного окремого магазину. Це передбачає наявність незалежних карт, але з уніфікованою логікою обробки, що ускладнює архітектуру програмного рішення.

1.6.3.2 Перспективи розвитку

Розвиток технологій відкриває широкі можливості для удосконалення системи навігації у магазинах. Багато з них вже знаходяться на стадії досліджень або тестування в роздрібному середовищі. Подальший розвиток програмного забезпечення може враховувати такі напрямки:

– технології доповненої реальності – AR відкриває нові можливості для зручної навігації. Замість статичної схеми користувач може бачити маршрут прямо на екрані смартфона поверх реального зображення магазину, що підвищує інтуїтивність і залученість;

– штучний інтелект для автоматичного оновлення карти – алгоритми комп'ютерного зору або аналізу переміщень персоналу можуть автоматично виявляти зміни у розміщенні товарів і пропонувати оновлення карти. Це значно скорочує навантаження на адміністраторів системи;

– розпізнавання поведінки покупців – інтеграція з системами моніторингу дозволяє збирати дані про маршрути клієнтів. Ця інформація може використовуватись для оптимізації структури магазину, підвищення ефективності викладки товарів і створення персоналізованих маршрутів;

– інтеграція з внутрішніми системами управління – синхронізація з ERP, POS та WMS-системами дозволяє автоматично оновлювати інформацію про наявність товарів, блокування проходів, зміну викладки. Це забезпечує цілісність і актуальність карти магазину у цифровому середовищі.

Підсумовуючи, створення ефективної цифрової карти магазину — це не лише питання технічної реалізації, а й системна задача, що вимагає адаптивності, підтримки змін і інтеграції з іншими компонентами цифрового середовища супермаркету.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ НАВІГАЦІЇ ПО МАГАЗИНУ НА ОСНОВІ СПИСКУ ТОВАРІВ

Розробка програмного забезпечення для навігації по магазину передбачає використання сучасного стеку технологій, який забезпечує ефективну реалізацію функціональності, гнучкість архітектури та надійність роботи застосунку. У цьому розділі розглянуто інструменти, які були використані під час розробки системи, а також обґрунтовано їх вибір залежно від вимог до проекту.

До основних категорій засобів, що використовувалися, належать: мова програмування та середовище розробки, фреймворки для побудови бекенд-застосунку, засоби захисту та авторизації, системи керування базами даних, інструменти для мапінгу даних, засоби для тестування API, а також рішення для контейнеризації та розгортання.

Вибір кожного інструмента зумовлений прагненням забезпечити модульність, масштабованість, зручність розгортання та подальшу підтримку програмного продукту. У наступних підрозділах описано функціональне призначення кожного компонента, його роль у системі, а також переваги, які він надає під час реалізації задачі.

2.1 Середовище розробки та мова програмування

У процесі створення програмного забезпечення для навігації по магазину було використано сучасне та зручне середовище розробки, а також мову програмування, яка забезпечує високу продуктивність, безпеку та підтримку великої екосистеми бібліотек і фреймворків.

2.1.1 IntelliJ IDEA

IntelliJ IDEA – це інтегроване середовище розробки, яке було обране для написання, тестування та налагодження коду. Воно підтримує повний стек Java

технологій і має розширену інтеграцію з системами керування версіями, збірки проєкту, запуску сервісів і автодоповненням коду [5].

Основні переваги використання IntelliJ IDEA у рамках проєкту:

– інтелектуальний аналіз коду – автоматичне виявлення помилок та рекомендації щодо рефакторингу;

– інтеграція зі Spring – зручне автозаповнення, конфігурація залежностей та навігація по контекстах Spring-застосунку;

– вбудована підтримка роботи з базами даних – можливість переглядати структуру таблиць, виконувати SQL-запити безпосередньо з IDE;

– інтеграція з Git – спрощене керування версіями та командами розробників.

Вибір цього середовища був обумовлений також широкою документаційною та спільнотною підтримкою, що полегшує вирішення нетипових технічних проблем.

2.1.2 Java

Для реалізації застосунку обрано мову програмування Java. Це мова з багаторічною історією розвитку, яка добре зарекомендувала себе в розробці корпоративних, веб та серверних застосунків [6].

Серед причин вибору Java для реалізації серверної логіки проєкту:

– об'єктно-орієнтована модель – дозволяє чітко структурувати бізнес-логіку у вигляді модулів, сервісів і моделей;

– підтримка багатопотоковості – важлива для роботи з обчисленнями маршруту та обробкою одночасних запитів;

– сумісність із Spring екосистемою – велика кількість інструментів і бібліотек доступна саме для Java;

– мова зі строгою типізацією – дозволяє зменшити кількість помилок під час компіляції;

– портативність – застосунок може запускатися на будь-якій платформі, де доступна JVM.

Java також добре підходить для реалізації складних алгоритмів, зокрема пошуку найкоротших шляхів, роботи з графами та обробки запитів у реальному часі. Крім того, велика кількість навчальних матеріалів і спільнот підтримки значно полегшує подальшу підтримку та розвиток проєкту.

2.2 Фреймворки та серверна основа

Для реалізації бекенд частини програмного забезпечення було використано перевірені часом фреймворки та архітектурні підходи, які забезпечують стабільну роботу, масштабованість та модульність системи. Основу серверної частини становить стек Spring, що включає Spring Boot і Spring Security, а логіка взаємодії з клієнтом реалізована у вигляді REST API.

2.2.1 Spring Boot

Spring Boot – це фреймворк, що базується на Spring Framework і призначений для швидкої розробки готових до запуску Java застосунків з мінімальним конфігуруванням. Його використання дозволило зосередитися на бізнес-логіці проєкту без необхідності вручну налаштовувати сервер, контейнери сервлетів, залежності тощо [7].

Основні переваги Spring Boot для цього проєкту:

– автоматична конфігурація – система самостійно визначає налаштування за замовчуванням для більшості компонентів;

– вбудований вебсервер – дозволяє запускати застосунок як звичайний jar-файл без додаткового контейнера;

– інтеграція з JPA, REST, Security, Web – забезпечує швидке підключення необхідних модулів;

– підтримка профілів конфігурації – дозволяє створювати різні середовища.

Spring Boot є ідеальним вибором для створення мікросервісної архітектури або модульних застосунків з чітким розподілом відповідальностей між компонентами.

2.2.2 Spring Security

Для захисту застосунку та реалізації механізмів автентифікації й авторизації було використано Spring Security, потужний модуль безпеки у складі Spring [8].

Цей фреймворк надає:

– гнучку систему авторизації з можливістю визначення доступу до ресурсів на основі ролей, прав або умов;

– підтримку JWT та OAuth2, що дозволяє реалізовувати сучасні протоколи безпеки у вебзастосунках;

– фільтри безпеки для обробки HTTP-запитів на ранніх етапах життєвого циклу;

– інтеграцію зі Spring Boot – конфігурація здійснюється зручно, з мінімумом коду.

Spring Security є надійним фундаментом для реалізації доступу на основі ролей, захисту REST-ендпоін та безпечного зберігання облікових даних.

2.2.3 REST

Комунікація між клієнтом і сервером реалізована у вигляді REST API – архітектурного стилю, що базується на використанні HTTP-запитів для доступу до ресурсів. Кожен ресурс має свій унікальний URL-ідентифікатор, і доступ до нього здійснюється через методи GET, POST, PUT, DELETE тощо [9].

Переваги REST під час реалізації системи:

– легкість у реалізації та тестуванні – запити можна перевіряти вручну за допомогою Postman;

– масштабованість – REST-сервіси легко масштабуються та кешуються;

– взаємодія з мобільними застосунками – формат JSON зручно обробляється на клієнтській стороні;

– незалежність клієнта і сервера – обидві частини можуть розвиватися незалежно.

REST забезпечує гнучкий спосіб зв’язку між клієнтом і сервером, що особливо актуально для системи, яка в майбутньому може бути розширена до повноцінної клієнтської платформи.

2.3 Аутентифікація та авторизація

Захист доступу до ресурсів – один із ключових аспектів будь-якого вебзастосунку, особливо у випадках, коли система обробляє персональні дані користувачів або виконує дії на основі їхніх прав. У розробленому програмному забезпеченні використовуються сучасні підходи до аутентифікації та авторизації, що базуються на відкритих протоколах і технологіях JWT та OAuth2.

2.3.1 JWT

JWT – це компактний, безпечний формат передачі інформації між учасниками системи у вигляді токена, який містить закодовану інформацію про користувача та його права доступу [10].

У застосунку JWT використовується для реалізації безстанової аутентифікації, тобто без потреби зберігати сесію на сервері. Процес працює наступним чином:

– користувач надсилає свої облікові дані на спеціальний ендпоінт;

– після успішної перевірки система видає токен, підписаний секретним ключем;

– токен додається до кожного подальшого запиту у заголовку Authorization;

– сервер перевіряє токен, дістає з нього інформацію про користувача і надає доступ до ресурсів.

Основні переваги JWT:

– відсутність стану сесії – покращує масштабованість застосунку;

– простота використання – токен зберігається на клієнті, як правило, у локальному сховищі або cookies;

– можливість контролю доступу – у токен можна додавати ролі, ідентифікатори та інші claims.

2.3.2 OAuth2

OAuth2 – це відкритий протокол авторизації, що дозволяє безпечно надавати стороннім застосункам доступ до обмежених ресурсів без розкриття облікових даних користувача [11].

У рамках розробки ця технологія використовується для:

– розмежування прав доступу між адміністраторами та звичайними користувачами;

– забезпечення авторизаційного потоку через токени;

– підтримки стандартизованого підходу до безпеки.

OAuth2 підтримує кілька авторизаційних схем. У даному випадку реалізовано сценарій Resource Owner Password Credentials, який підходить для застосунків з власним інтерфейсом автентифікації.

Spring Security надає повну інтеграцію з OAuth2, що дозволяє централізовано налаштувати всі механізми безпеки, включаючи:

– фільтрацію запитів на основі ролей;

– валідацію токенів;

– обробку помилок доступу.

Таким чином, комбінація JWT і OAuth2 дозволяє створити сучасну, гнучку та безпечну модель доступу до ресурсоємної системи навігації, яка враховує контекст дій користувача і дозволяє ефективно керувати правами доступу.

2.4 Робота з базами даних

Оскільки розроблене програмне забезпечення оперує великою кількістю структурованих даних, таких як інформація про магазини, полиці, товари, зв’язки між об’єктами, маршрути, користувачів та їхні списки покупок, необхідним є

використання надійної системи керування базами даних. У даному проєкті було обрано PostgreSQL як основну СКБД, а для автоматизації управління змінами в схемі бази – Liquibase.

2.4.1 PostgreSQL

PostgreSQL – це потужна об'єктно-реляційна система керування базами даних з відкритим вихідним кодом. Вона підтримує складні типи даних, транзакції, цілісність даних і є однією з найнадійніших систем для зберігання структурованої інформації [12].

Основні переваги використання PostgreSQL у цьому проєкті:

– підтримка складних зв’язків між таблицями – важливо для збереження просторових зв’язків між полицями, магазинами, товарами;

– розширені типи даних – зручно для зберігання координат, JSON-структур, часових міток;

– можливість ефективної індексації – критично важливо при роботі з великими таблицями та частому пошуку;

– сумісність із Spring Data JPA – дозволяє зручно реалізовувати запити через репозиторії, без написання SQL-коду вручну;

– масштабованість і стабільність – PostgreSQL добре себе зарекомендувала в багатокористувацьких системах.

У базі зберігаються основні сутності:

– магазини;

– полиці;

– просторові зв’язки між координатами;

– товари;

– кошики і їхній вміст;

– користувачі.

Кожна таблиця має чітко визначені зв’язки, первинні та зовнішні ключі, що гарантує цілісність і логічну послідовність даних.

2.4.2 Liquibase

Liquibase – це інструмент керування версіями схеми бази даних, який дозволяє зберігати всі зміни у вигляді файлів, що можуть бути автоматично застосовані під час запуску застосунку [13].

Його використання дає змогу:

– синхронізувати структуру бази даних у всіх середовищах;

– відстежувати історію змін – кожна зміна зберігається у вигляді кроку з описом, автором і міткою часу;

– безпечно оновлювати базу – всі зміни перевіряються перед виконанням;

– інтегрувати міграції в CI/CD процеси – забезпечується автоматизація розгортання оновлень.

Для проєкту це особливо актуально, оскільки структура даних є складною, а зміни до неї можуть вноситися поступово в ході розробки або підтримки проєкту.

Таким чином, комбінація PostgreSQL і Liquibase забезпечує не лише стабільне зберігання інформації, а й контрольоване розгортання схеми даних, що критично важливо для командної розробки та безпечної еволюції застосунку.

2.5 Робота з моделями даних

У сучасних застосунках, які дотримуються принципів багаторівневої архітектури, особливу увагу приділяють чіткому розділенню даних між шарами: базою даних, доменом та інтерфейсами API. Для забезпечення зручного й безпечного перетворення об'єктів між цими шарами часто використовуються спеціалізовані інструменти, зокрема, MapStruct і Lombok.

2.5.1 MapStruct

MapStruct – це Java бібліотека для автоматичного створення маперів, тобто класів, що виконують перетворення об’єктів з одного типу в інший. Її ключова особливість полягає у тому, що вона генерує код під час компіляції, що забезпечує високу продуктивність і типобезпечність [14].

Зазвичай MapStruct використовується для мапінгу між:

– DTO – об'єктами, які передаються через API;

– доменними моделями – об'єктами, з якими працює бізнес-логіка;

– сутностями бази даних – об'єктами, які зберігаються у СКБД.

Основні переваги використання MapStruct:

– продуктивність – код генерується на етапі компіляції, тому немає втрат через рефлексію;

– безпека типів – помилки мапінгу виявляються на рівні компіляції;

– зменшення шаблонного коду – немає потреби вручну реалізовувати десятки однакових перетворень;

– гнучкість налаштувань – підтримка складних мапінгів, об’єднань, виключень полів тощо.

Цей інструмент добре інтегрується з іншими бібліотеками Java та не вимагає додаткового навантаження, що робить його популярним вибором у проєктах із чітким розмежуванням рівнів логіки.

2.5.2 Lombok

Lombok – це бібліотека, що призначена для зменшення обсягу шаблонного коду в Java застосунках. Вона використовує анотації, які інструктують компілятор автоматично генерувати часто використовувані елементи класів: гетери, сетери, конструктори, методи equals, hashCode, toString тощо [15].

Ключові анотації, які надає Lombok:

– @Getter, @Setter – автоматичне створення методів доступу до полів;

– @NoArgsConstructor, @AllArgsConstructor, @RequiredArgsConstructor – генерація різних типів конструкторів;

– @Builder – реалізація шаблону побудовника для зручного створення об'єктів;

– @EqualsAndHashCode, @ToString – перевизначення базових методів;

– @Data – комплексна анотація, яка об’єднує кілька інших для швидкої декларації класу даних.

Використання Lombok дозволяє підвищити читабельність коду, зменшити ризик людських помилок при дублюванні логіки та спростити підтримку проєкту.

Загалом, застосування таких інструментів, як MapStruct і Lombok, суттєво покращує процес роботи з даними, зменшує обсяг рутинної реалізації та дозволяє зосередитися на логіці системи, а не на технічних деталях перетворення об’єктів.

2.6 Тестування і взаємодія з API

У системах, побудованих за архітектурою REST, API виступає основним інтерфейсом взаємодії між клієнтом і сервером. Для забезпечення стабільності, передбачуваності та відповідності поведінки системи очікуванням користувачів, необхідне регулярне тестування HTTP ендпоінтів. Для цих цілей широко використовуються інструменти, які дозволяють тестувати API як вручну, так і в автоматизованому режимі. Одним з найбільш популярних засобів є Postman.

Postman – це безкоштовний інструмент для тестування API, який забезпечує зручний графічний інтерфейс для створення, надсилання та аналізу HTTP запитів і відповідей [16].

Основні можливості Postman:

– створення та збереження колекцій запитів – дозволяє групувати запити за сценаріями використання;

– редагування параметрів запиту – метод, заголовки, тіло, параметри шляху або рядка запиту;

– аналіз відповідей – виведення статусу, часу відповіді, тіла, заголовків і відформатованого JSON;

– тестування безпеки – перевірка доступності захищених ресурсів за допомогою токенів автентифікації;

– автоматизація тестів – підтримка скриптів для перевірки результатів відповідей;

– імпорт та експорт колекцій – зручний обмін запитами між розробниками, тестувальниками та технічною документацією.

Postman є інструментом, який не лише полегшує перевірку коректності API, але й слугує засобом комунікації між різними членами команди. Він дозволяє імітувати роботу кінцевого користувача, наочно протестувати сценарії, виявити помилки, провести регресійне тестування при внесенні змін у бізнес-логіку.

Застосування такого засобу є особливо корисним на ранніх етапах розробки, коли ще відсутній фронтенд, а також у фазі системного та модульного тестування для валідації відповідей сервера.

2.7 Контейнеризація і розгортання

Після завершення розробки програмного забезпечення постає завдання його стабільного, ізольованого та передбачуваного розгортання в різних середовищах — як локально, так і на сервері. Для вирішення цих задач у сучасній практиці активно застосовується підхід контейнеризації, який передбачає пакування застосунку разом із усіма залежностями в окреме середовище виконання. У межах цього проєкту для контейнеризації було обрано Docker, провідну платформу для створення, керування та запуску контейнерів.

Docker – це інструмент, який дозволяє створювати стандартизовані середовища для запуску застосунків у вигляді ізольованих контейнерів. Контейнер містить усе необхідне для роботи застосунку: виконуваний код, бібліотеки, середовище запуску, системні залежності тощо [17].

Ключові переваги Docker:

– уніфікованість середовищ – незалежно від того, де розгортається контейнер, його поведінка залишається ідентичною;

– ізольованість – кожен контейнер працює незалежно від інших процесів і не впливає на основну операційну систему;

– простота розгортання – один Docker образ може запускатися на будь-якому хості з встановленим Docker Engine;

– швидкий старт і масштабування – запуск контейнерів займає лічені секунди, що спрощує CI/CD процеси;

– інтеграція з Docker Compose – дозволяє одночасно запускати кілька взаємопов’язаних сервісів.

У типових сценаріях розгортання вебзастосунків Docker дозволяє:

– автоматизувати підняття серверного середовища;

– створити окремі контейнери для застосунку, бази даних, кешу та інших сервісів;

– описати конфігурацію у вигляді Dockerfile та docker-compose.yml, що зручно для командної роботи.

Використання Docker забезпечує гнучкість під час розробки, тестування та продакшен розгортання, дозволяє уникати проблем із сумісністю середовищ та значно пришвидшує цикл деплойменту.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ НАВІГАЦІЇ ПО МАГАЗИНУ НА ОСНОВІ СПИСКУ ТОВАРІВ

На етапі проєктування було визначено вимоги до системи, обрана архітектура застосунку, а також здійснено розподіл логіки за окремими рівнями для досягнення високого рівня модульності, супроводжуваності та масштабованості. Розробка орієнтувалась на принципи чистої архітектури та забезпечення відокремлення бізнес-логіки від технічної реалізації.

3.1 Вимоги до системи

На етапі проєктування системи навігації по магазину на основі списку товарів важливо чітко визначити вимоги, які впливають як на функціональну поведінку системи, так і на її архітектуру, якість коду та зручність використання. Усі вимоги поділяються на функціональні та нефункціональні.

3.1.1 Функціональні вимоги

– система повинна надавати користувачу список товарів магазину;

– користувач має можливість додавати товари до персонального списку;

– система повинна визначати розташування кожного товару у магазині;

– система має будувати оптимальний маршрут по магазину з урахуванням розміщення товарів;

– користувач має змогу переглянути згенерований маршрут у вигляді схеми;

3.1.2 Нефункціональні вимоги

– архітектура повинна забезпечувати модульність та гнучкість системи;

– інтерфейс API має бути RESTful та задокументований;

– код повинен бути написаний з урахуванням принципів чистого коду [18].

3.2 Архітектура системи

Для забезпечення модульності, масштабованості та легкості супроводу програмного забезпечення було обрано багаторівневу архітектуру, яка ґрунтується на принципах розділення відповідальностей. Такий підхід дозволяє ізолювати бізнес-логіку від зовнішніх систем, забезпечити гнучкість інтеграцій і полегшити тестування окремих компонентів [19-20].

Архітектура системи умовно поділена на три основні рівні:

– доменний рівень;

– рівень застосунку;

– інфраструктурний рівень.

3.2.1 Багаторівнева архітектура: розділення на шари

Доменний рівень – центральний шар системи, який містить бізнес-логіку і предметні сутності. Він не залежить від жодних технічних засобів або зовнішніх систем. Саме тут реалізуються правила побудови маршруту, визначення розташування товарів та інші ключові функції [19-20].

Основні компоненти:

– предметні сутності;

– сервіси для обробки маршрутів, структури магазину, списку товарів;

– інтерфейси для взаємодії з базою даних або зовнішніми джерелами.

На рисунку 3.1 зображено діаграму класів доменного рівня. На ній представлено основні класи домену, їхню структуру та взаємозв’язки. Центральною частиною є клас Store, який агрегує об’єкти StoreGraph, ShelfIndex, CoordinateShelfIndex і StoreDetails. Побудова маршруту здійснюється через сервіс GreedyRouteFinder, який реалізує інтерфейс RouteFinder і використовує графову модель із класів StoreGraph, Edge та Coordinate.

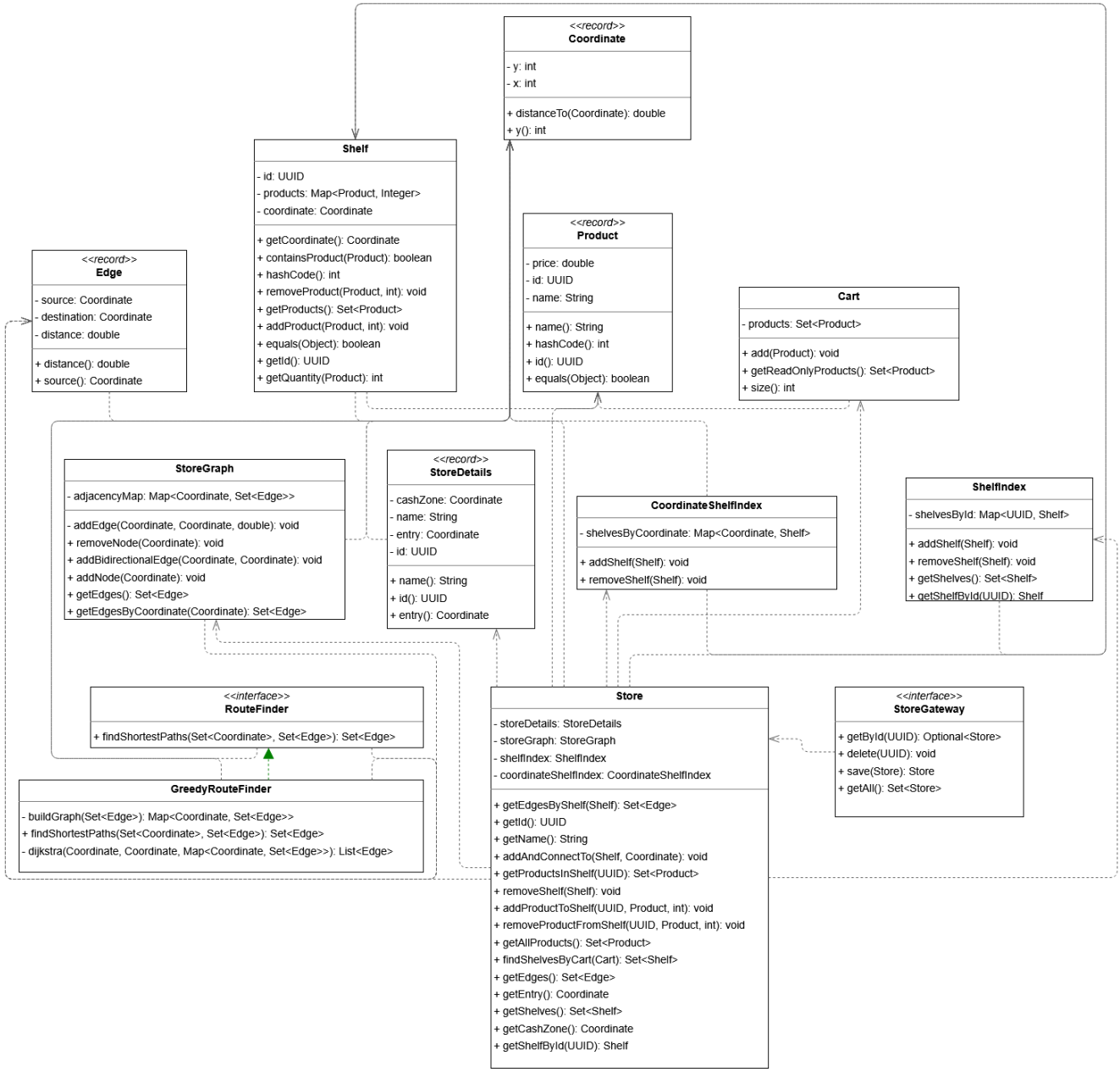


Рис. 3.1 Діаграма класів доменного модуля

Діаграма також демонструє, як класи взаємодіють між собою в межах бізнес-логіки без залежності від бази даних, що відповідає принципам чистої архітектури. Кожен клас відповідає за конкретну відповідальність, що полегшує тестування та розширення системи.

Рівень застосунку – рівень, що відповідає за координацію виконання бізнес-логіки. Він обробляє вхідні запити, викликає відповідні сервіси домену, перетворює DTO у внутрішні об’єкти та навпаки [19-20].

Його функції:

– управління сценаріями використання;

– мапінг моделей між шарами;

– обробка валідації, авторизації, логування;

– взаємодія з сервісами домену через інтерфейси.

Інфраструктурний рівень – рівень, що відповідає за комунікацію із зовнішніми сервісами. Він залежить від технологічних рішень і виконує роль плагіна до логіки, описаної в домені [19-20].

Його елементи:

– репозиторії;

– компоненти авторизації;

– конфігураційні файли.

3.3 Модель предметної області

Модель предметної області – це концептуальна основа програмного забезпечення, яка відображає ключові об’єкти реального світу та їхні зв’язки у контексті задачі навігації по магазину. Вона визначає, які сутності існують у системі, як вони пов’язані між собою та яку роль відіграють у процесі формування списку товарів, побудови маршруту та взаємодії користувача з магазином.

3.3.1 Основні сутності

У межах предметної області виділяються такі основні об’єкти:

– магазин – представляє торговий зал, у межах якого здійснюється навігація;

– полиця – одиниця внутрішньої структури магазину, що розміщується у конкретному місці згідно з координатною сіткою;

– товар – об’єкт, який може бути розміщений на полицях і доданий до списку покупок користувача;

– кошик – віртуальний список, який формується користувачем із вибраних товарів.

3.3.2 Розширення моделі в інших контекстах

Деякі сутності, необхідні для повноцінної роботи системи, не входять до предметної області навігації, але реалізовані в інфраструктурному рівні:

– користувач – входить до контексту автентифікації та авторизації. У домені навігації відсутній;

– зв’язки між координатами – зберігаються у базі даних як таблиця зв’язків між полицями. У домені ці дані агрегуються у StoreGraph.

Сутність маршруту не виділена ані в домені, ані в базі, оскільки результатом алгоритму пошуку шляху є список точок, що передається у вигляді відповіді клієнту, без потреби в окремому класі або таблиці.

Такий підхід забезпечує гнучке розділення відповідальностей, дозволяє масштабувати систему без дублювання логіки між рівнями й підтримує чистоту доменної моделі [18].

3.4 Механізм побудови маршруту

Однією з ключових функцій розробленого програмного забезпечення є побудова оптимального маршруту проходження покупця по магазину на основі сформованого списку товарів. Цей процес вимагає внутрішнього представлення простору магазину у вигляді структури, придатної для навігації, а також застосування алгоритму пошуку найкоротшого шляху з урахуванням усіх заданих точок.

3.4.1 Формалізація карти магазину як графа

Для вирішення задачі навігації простір магазину моделюється у вигляді орієнтованого зваженого графа, де:

– вершинами є полиці, вхід, каса або будь-які інші ключові координати магазину;

– ребрами є можливості переміщення між сусідніми полицями або зонами;

– ваги ребер відповідають за реальну відстані або час проходження.

У доменній моделі граф представлений об'єктом StoreGraph, який інкапсулює логіку додавання поличок та побудови зв’язків між ними.

3.4.2 Алгоритм пошуку маршруту

Після того як користувач формує кошик із переліком товарів, магазин формує список полиці згідно цього кошику, на яких ці товари знаходяться. Далі цей список полиць передається сервісу, де виконується побудова найкоротшого маршруту, який:

– починається від точки входу до магазину;

– проходить через усі полиці, де розташовані потрібні товари;

– закінчується в зоні кас.

Для реалізації цього процесу було використано гібридний підхід, що поєднує:

– алгоритм Дейкстри [21] – для знаходження найкоротших шляхів між будь-якими двома точками на карті магазину, представленій у вигляді графа;

– жадібний алгоритм [22] – для вибору наступної полиці для відвідування. На кожному кроці алгоритм обирає найближчу ще не відвідану полицю.

Такий підхід дозволяє швидко формувати маршрут із прийнятною оптимальністю без повного перебору всіх перестановок. Він добре масштабується при збільшенні кількості товарів і адаптується до зміни структури магазину або розташування полиць.

Результатом роботи алгоритму є список координат полиць, який представляє маршрут переміщення покупця магазином. Цей список повертається на клієнтську частину для візуалізації.

3.5 Сценарії взаємодії з системою

Сценарії взаємодії описують основні дії, які можуть виконувати користувачі системи в межах реалізованої функціональності. У програмному забезпеченні для навігації по магазину передбачено два основних типи акторів: адміністратор і користувач. Кожен з них має власні цілі й набір дозволених дій. Опис сценаріїв дозволяє конкретизувати функціональні вимоги до системи, візуалізувати процеси та покращити розуміння логіки взаємодії з програмним інтерфейсом.

3.5.1 Сценарії для адміністратора

Адміністратор відповідає за конфігурацію магазину та наповнення його структурними елементами. Основні сценарії включають:

– створення магазину – адміністратор ініціалізує новий об’єкт магазину, задаючи йому унікальну назву, координати точки входу та розташування зони кас;

– додавання полиць до магазину – у межах створеного магазину адміністратор розміщує полиці, кожна з яких має координатну позицію та може бути з’єднана з іншими полицями через ребра;

– зв’язування полиць між собою – формуються ребра графа, які описують можливість переміщення між полицями. Це забезпечує навігаційну карту для майбутніх розрахунків маршруту;

– створення товарів і їх розміщення – адміністратор додає товари до системи, а потім прив’язує їх до конкретних полиць магазину. Один товар може бути присутній на кількох полицях.

Ці дії, як правило, реалізовані через закриті ендпоінти, доступ до яких обмежено роллю адміністратора.

3.5.2 Сценарії для користувача

Кінцевий користувач взаємодіє із системою для полегшення процесу здійснення покупок. Його сценарії виглядають так:

– вибір магазину – користувач переглядає список доступних магазинів і обирає той, у якому планує здійснити покупки;

– ознайомлення з асортиментом товарів – після вибору магазину користувач отримує список товарів, які є в наявності, з можливістю фільтрації та пошуку;

– формування списку покупок – користувач додає товари до персонального кошика;

– отримання оптимального маршруту – на основі сформованого кошика користувач надсилає запит до системи, яка повертає оптимальний маршрут по магазину;

– перегляд схеми маршруту – відповідь на запит містить список координат, які можуть бути візуалізовані на стороні клієнта як інтерактивна або статична схема проходження.

3.5.3 Діаграма варіантів використання

На рисунку 3.1 зображена діаграма варіантів використання. Вона дозволяє чітко візуалізувати розподіл ролей і функціональних можливостей у системі, а також визначити межі відповідальності для кожного типу користувача.

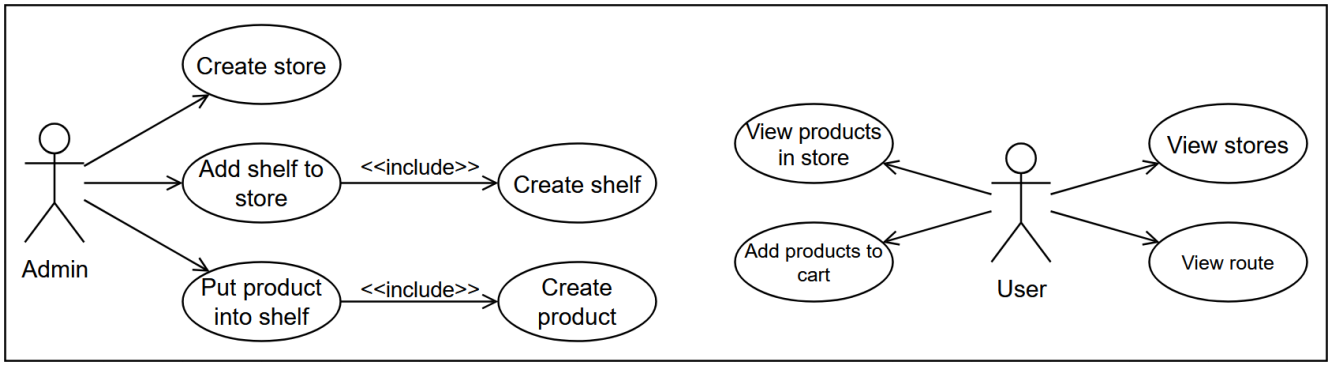


Рис. 3.2 Діаграма варіантів використання

3.6 Модель бази даних

Для збереження структурованої інформації про магазини, полиці, товари, користувачів, списки покупок та просторові зв’язки між елементами

використовується реляційна база даних, спроєктована відповідно до логіки предметної області та вимог системи. Дані організовані у вигляді таблиць із чітко визначеними зв’язками. Особливу увагу приділено збереженню просторової структури магазину та взаємозв’язку між товарами й полицями.

3.6.1 Структура таблиць

У системі виділено дві основні групи таблиць: таблиці структур магазину і товарів, а також таблиці взаємодії користувача з системою.

Таблиці структури магазину:

– stores – зберігає магазини;

– shelves – описує полиці з координатами, які належать певному магазину;

– edges – зберігає зв’язки між полицями у вигляді ребер графа;

– products – містить інформацію про товари;

– shelf_product – проміжна таблиця для зв’язку "багато-до-багатьох" між полицями та товарами.

Таблиці користувацької взаємодії:

– users – облікові записи користувачів, включаючи логін, зашифрований пароль та роль;

– carts – список товарів, який створює користувач;

– cart_product – зв’язує конкретний кошик з обраними товарами.

3.6.2 Особливості збереження просторової структури

Карта магазину зберігається у вигляді орієнтованого зваженого графа, що представлений двома таблицями:

– shelf – вершини графа, де кожна полиця має унікальні координати;

– edge – ребра графа, які визначають можливість переміщення між полицями та містять вагу.

Такий підхід дозволяє будувати маршрути динамічно, з урахуванням змін у плануванні магазину.

3.6.3 Діаграма бази даних

На рисунку 3.2 зображена ER-діаграма, яка демонструє усі перераховані таблиці та їхні атрибути. Дана структура дозволяє ефективно реалізовувати логіку побудови маршруту на основі списку товарів, зберігати інформацію про просторову структуру магазину та забезпечувати персоналізацію взаємодії з кожним користувачем.

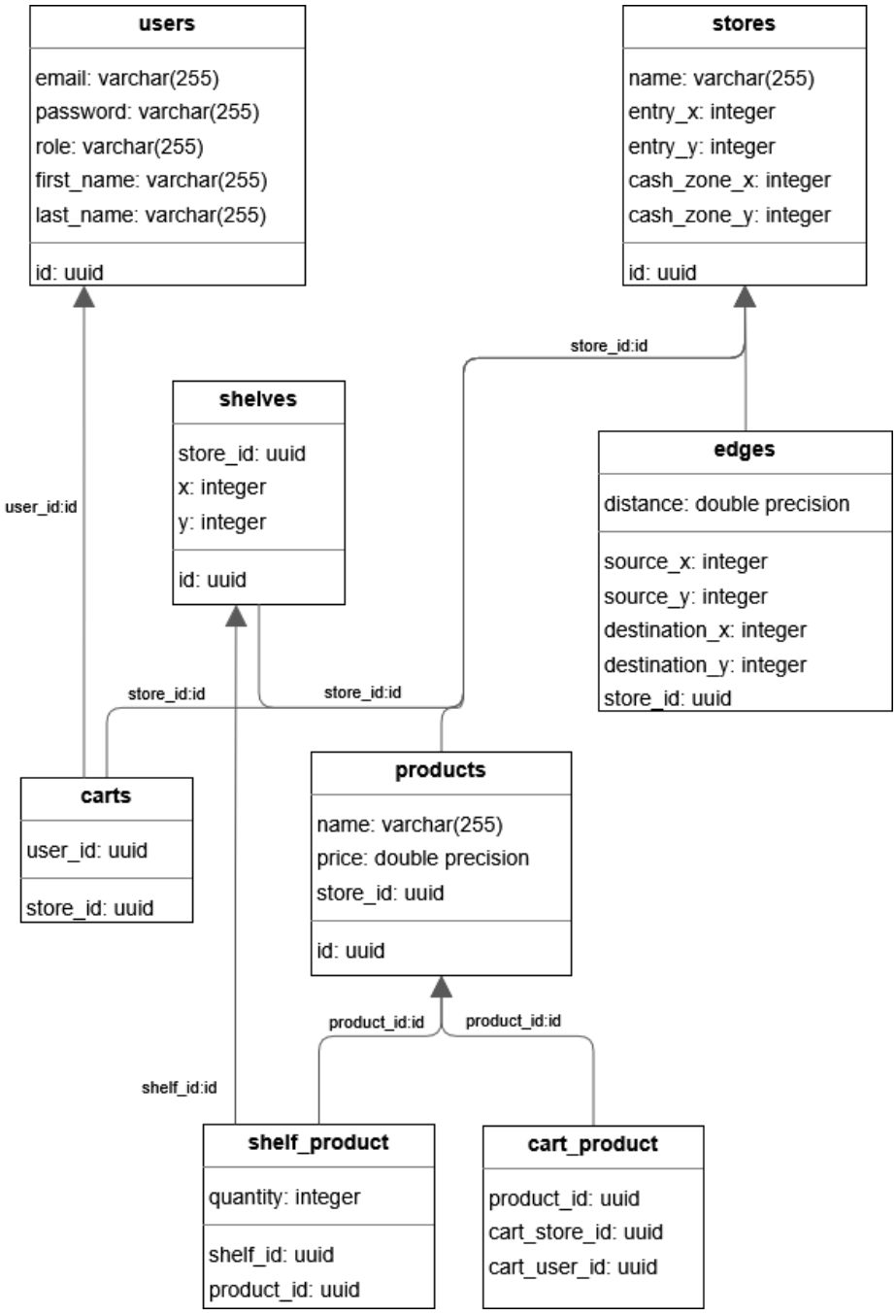


Рис. 3.3 ER-діаграма

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Після етапу проєктування архітектури системи було здійснено безпосередню реалізацію програмного забезпечення для навігації по магазину на основі списку товарів. Основна мета цього етапу – втілення спроєктованої логіки у вигляді робочого застосунку, здатного надавати користувачеві повноцінний сервіс з пошуку товарів та побудови оптимального маршруту.

Цей розділ описує процес реалізації ключових компонентів системи, а саме структури проєкту, логіки взаємодії між рівнями, алгоритмів обробки запитів, захисту доступу, а також підходів до тестування. Реалізацію здійснено на основі попередньо визначених функціональних і нефункціональних вимог з дотриманням принципів чистої архітектури, що забезпечує гнучкість і масштабованість рішення.

Особливу увагу приділено перевірці працездатності компонентів. Для валідації правильності побудови маршруту використано конкретні сценарії на основі сформованих списків товарів і відповідної структури магазину.

4.1 Організація проєкту

Для забезпечення структурованої, масштабованої та підтримуваної реалізації програмного забезпечення було застосовано модульний підхід до організації проєкту. Архітектура побудована з чітким розділенням відповідальностей між логічними компонентами, що реалізують бізнес-правила, спілкування з користувачами та сторонніми сервісами.

4.1.1 Структура проєкту

Проєкт складається з кількох логічних модулів, кожен із яких виконує окрему роль у системі:

– domain – містить сутності предметної області, а також сервіси, що інкапсулюють бізнес-логіку, не залежачи від зовнішніх бібліотек, баз даних чи фреймворків;

– application – реалізує сценарії використання — управління створенням магазинів, наповненням полиць, обробкою списку товарів, запуском алгоритму пошуку маршруту тощо;

– infrastructure – містить реалізації інтерфейсів, роботу з базою даних та конфігураційні файли.

Такий поділ дозволяє змінювати внутрішню логіку, не зачіпаючи зовнішні залежності, та навпаки.

4.1.2 Інтеграція з системою керування версіями

Розробка системи здійснювалась з використанням Git як системи контролю версій. У репозиторії реалізовано ізольоване ведення гілок:

– main – стабільна версія;

– dev – поточна розробка;

– окремі гілки для окремих задач.

Це забезпечує надійність збереження коду та можливість повернення до стабільних версій.

4.1.3 Налагодження середовища

Для спрощення розгортання та забезпечення ізольованості середовища виконання було використано Docker. Усі компоненти системи запускаються у вигляді окремих контейнерів, що дозволяє швидко розгорнути застосунок у будь-якому середовищі без складного ручного налаштування.

Управління контейнерами здійснюється за допомогою файлу docker-compose.yml, який описує конфігурацію сервісів, мережі та змінні середовища. Для зберігання даних використовується PostgreSQL, а ініціалізація структури бази даних виконується автоматично за допомогою Liquibase.

Конфігураційні параметри виносяться у файл .env, що спрощує адаптацію до різних середовищ без необхідності зміни коду.

4.2 Реалізація основної бізнес-логіки

Бізнес-логіка системи реалізована на доменному рівні, що забезпечує її незалежність від способів зберігання даних або протоколів взаємодії. Саме доменні класи відповідають за формування структури магазину, обробку списку товарів, керування просторовими зв’язками та побудову маршруту. Рівень аплікації у свою чергу виконує роль координатора між користувачем за доменним рівнем. Він отримує запити, викликає відповідні методи доменних сервісів та готує результат для передачі у зовнішній світ.

4.2.2 Створення магазину та наповнення його структурою

Однією з перших дій адміністратора є створення нового магазину. Для цього система приймає вхідні дані, а саме назву магазину і координати входу та касової зони. У відповідь створюється доменний об’єкт Store, який інкапсулює карту магазину у вигляді графа.

Після створення магазину до нього можуть бути додані полиці. Кожна полиця має унікальні координати та додається до графа магазину як вузол. Для побудови повноцінного графа адміністратор також створює зв’язки між полицями.

4.2.3 Додавання товарів та їх прив’язка до полиць

Окремим кроком є додавання товарів. Після створення товару система дозволяє асоціювати його полицею. Це забезпечує можливість гнучкого розміщення товарів та оптимізації маршруті.

4.2.4 Обробка списку покупок

Кінцевий користувач формує список покупок, додаючи до кошика потрібні товари з доступного асортименту магазину. Цей кошик передається системі як джерело даних для побудови маршруту.

На основі переліку товарів з кошика застосунок знаходить відповідні полиці, на яких вони знаходяться. Далі ці полиці формуються у список та повертаються для подальшого опрацювання.

4.2.5 Побудова маршруту

Після підготовки вхідних даних система викликає сервіс маршрутизації, який виконує пошук оптимального маршруту. Цей маршрут обов’язково включає вхід у магазин, усі полиці, які ми передали алгоритму, та касову зону.

Маршрут будується з використанням гібридного алгоритму, який поєднує:

– алгоритм Дейкстри [21] – для знаходження найкоротших шляхів між точками графа;

– жадібний алгоритм [22] – для вибору наступної точки маршруту на основі поточної позиції користувача та відстані до найближчої полиці з товаром.

У результаті формується список координат, які відповідають маршруту, і повертається на клієнтську частину для візуалізації.

4.3 Реалізація API

Для взаємодії з програмним забезпеченням реалізовано набір RESTful API ендпоінтів, які надають доступ до основної функціональності як для адміністратора, так і для кінцевого користувача. API спроєктовано відповідно до принципів REST [9], що дозволяє забезпечити просту, передбачувану та масштабовану взаємодію з клієнтськими застосунками або іншими сервісами.

Використано HTTP-методи відповідно до їх призначення:

– GET – отримання даних;

– POST – створення нових ресурсів;

– PUT / PATCH – оновлення існуючих об'єктів;

– DELETE – видалення ресурсів.

4.3.1 Ендпоінти адміністратора

Адміністратор має доступ до операцій із налаштування структури магазину. Основні запити:

– POST /api/stores – створення магазину;

– POST /api/stores/{store_id}/shelves – додавання полиці до магазину;

– POST /api/stores/{store_id}/shelves/{shelf_id}/products – створення нового товару.

Ці запити захищені роллю ADMIN і недоступні пересічним користувачам.

4.3.2 Ендпоінти користувача

Користувач працює з застосунком у контексті покупок. Йому доступні:

– GET /api/stores – перегляд списку доступних магазинів;

– GET /api/stores/{store_id}/products – отримання асортименту товарів магазину;

– POST /api/stores/{store_id}/cart – додавання товару до кошика;

– GET /api/stores/{store_id}/route – отримання маршруту на основі кошика.

Ці запити дозволяють повноцінно виконати сценарій: вибір магазину, формування списку товарів, побудова маршруту.

4.4 Безпека та авторизація

Оскільки програмне забезпечення реалізує як адміністративні функції, так і дії кінцевих користувачів, важливо забезпечити контроль доступу до ресурсів, захист даних та ідентифікацію запитів. У системі реалізовано ролеву модель доступу, авторизацію на основі JWT, а також підтримку OAuth2 для взаємодії з ресурсами [23].

4.4.1 Ролі користувачів

У системі передбачено два типи ролей:

– ADMIN – має доступ до адміністративних ендпоінтів: створення магазинів, додавання полиць, товарів, побудова структури магазину;

– USER – має доступ до функціональності перегляду магазинів, формування списку товарів, отримання маршруту.

Кожен ендпоінт API конфігурується з урахуванням дозволеної ролі. Доступ перевіряється на рівні фреймворку Spring Security.

4.4.2 Авторизація за допомогою JWT

Система автентифікації реалізована на основі JWT – безпечного способу передачі інформації про користувача між клієнтом і сервером [10]. Алгоритм роботи:

– користувач проходить автентифікацію;

– у відповідь отримує JWT, який містить інформацію про роль та ідентифікатор;

– усі наступні запити містять токен у заголовку Authorization;

– сервер перевіряє токен і надає або обмежує доступ до відповідних ресурсів.

JWT забезпечує відмову від серверної сесії, масштабованість та простоту розподіленої авторизації.

4.4.3 Підтримка OAuth2

Для уніфікованої інтеграції з системами автентифікації використовується підтримка OAuth2 [11]. Налаштування реалізовано на базі Spring Security з наступними можливостями [24]:

– розширення механізмів автентифікації;

– підключення сторонніх постачальників авторизації;

– обробка токенів у форматі JWT.

Це створює основу для майбутньої інтеграції з зовнішніми сервісами без зміни основної бізнес-логіки системи.

На рисунку 4.2 проілюстровано процес аутентифікації користувача за JWT. Сервер вилучає токен з заголовка Authorization, перевіряє його справжність, термін дії та відповідність цифрового підпису. Якщо токен валідний, користувач аутентифікується і запит передається далі у контексті його ролі. У разі помилки потік виконання переходить до відповідного обробника виключення, який постачається фреймворком.

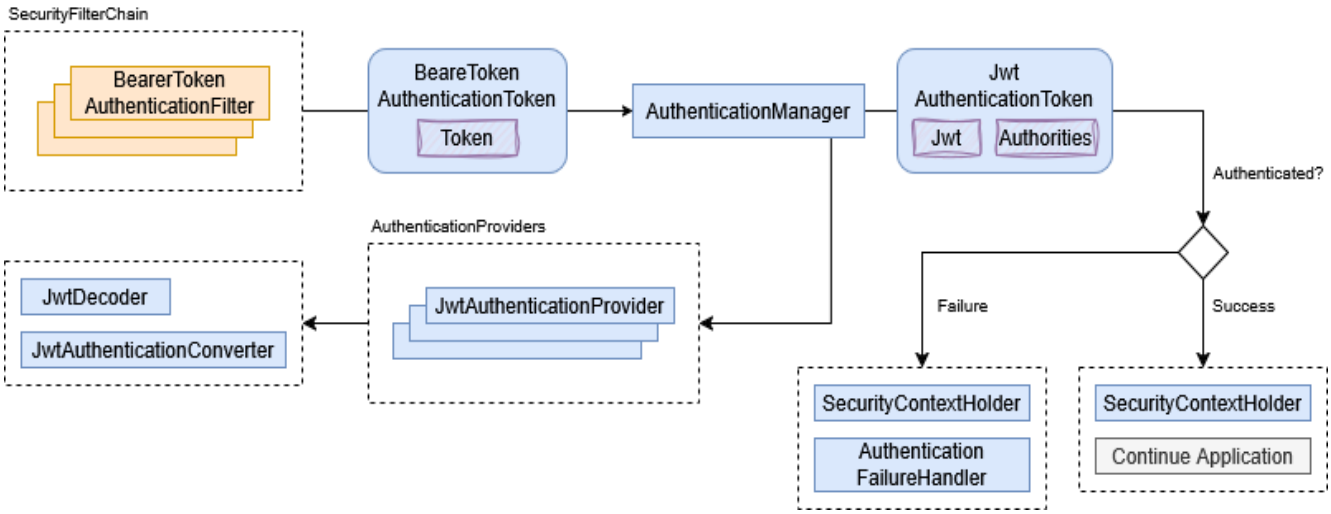


Рис. 4.1 Діаграма потоку аутентифікації

4.5 Тестування

З метою перевірки коректності реалізації бізнес-логіки системи та забезпечення її надійної роботи було проведено модульне тестування основних компонентів, а також ручне тестування REST API. Особливу увагу приділено перевірці функціональності побудови маршруту, що є критичним елементом застосунку.

4.5.1 Модульне тестування

Модульні тести зосереджені на перевірці доменного рівня, оскільки саме там реалізована основна бізнес-логіка. Для цього було написано окремі тести для таких сценаріїв:

– створення магазину з початковими параметрами;

– додавання полиць до магазину та перевірка координат;

– побудова структури графа;

– розміщення товарів на полицях;

– підбір полиць, що містять задані товари;

– обробка кошика та передача його в навігаційний модуль.

Тести реалізовано з використанням JUnit 5 [25]. У випадках, де потрібна емуляція залежностей, використовується Mockito [26].

4.5.2 Тестування алгоритму побудови маршруту

Оскільки обчислення шляху є центральною функцією системи, для алгоритму побудови маршруту були реалізовані окремі тести, що перевіряють:

– правильність маршруту між точками;

– коректну обробку початкової точки та кінцевої;

– поведінку алгоритму на спрощених графах.

Такі тести дозволили переконатися в правильності реалізації гібридного алгоритму пошуку шляху у більшості типових сценаріїв.

4.5.3 Ручне тестування API

REST API було перевірено вручну за допомогою Postman. Було проведено:

– тестування всіх основних запитів, а саме створення магазину, полиць, товарів, кошика, генерація маршруту;

– перевірку поведінки системи при помилкових даних;

– перевірку доступу до ендпоінтів відповідно до ролей;

– тестування обробки JWT у заголовках запитів.

Цей етап дозволив переконатися, що API відповідає очікуваній поведінці та успішно взаємодіє з бізнес-логікою.

4.6 Результати та демонстрація

У результаті реалізації програмного забезпечення для навігації по магазину на основі списку товарів було отримано повнофункціональну систему, що відповідає поставленим функціональним та архітектурним вимогам. У цьому підрозділі наведено аналіз досягнутих результатів, проведено оцінку коректності реалізації та ефективності запропонованих технічних рішень.

4.6.1 Оцінка функціонального покриття

Система реалізує повний функціональний цикл, що був визначений на етапі проєктування. Всі ключові сценарії, включно з автентифікацією користувача, вибором магазину, роботою з товарами, формуванням кошика та побудовою маршруту, були втілені. У цьому підрозділі наведено ілюстрації, які демонструють, як ці функції реалізовані на практиці.

Першим кроком взаємодії з системою є реєстрація нового користувача. Для цього створено форму, яка дозволяє ввести необхідні дані: ім’я, прізвище, адресу електронної пошти та пароль. Цей процес є обов’язковим для доступу до функціоналу сервісу. На рисунку 4.2 зображено форму реєстрації

Після успішної реєстрації або при повторному використанні система надає можливість виконати вхід у систему. Для цього реалізовано форму аутентифікації, яка зображена на рисунку 4.3. Вона приймає адресу електронної пошти та пароль користувача. У випадку успішного входу видається JWT, який забезпечує доступ до захищених ресурсів.

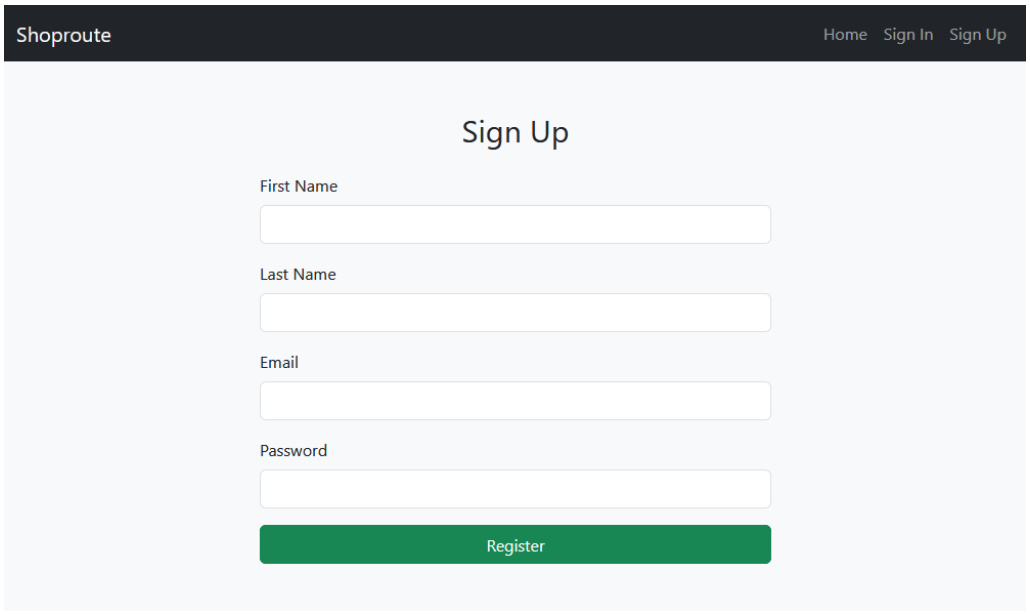


Рис. 4.2 Форма реєстрації

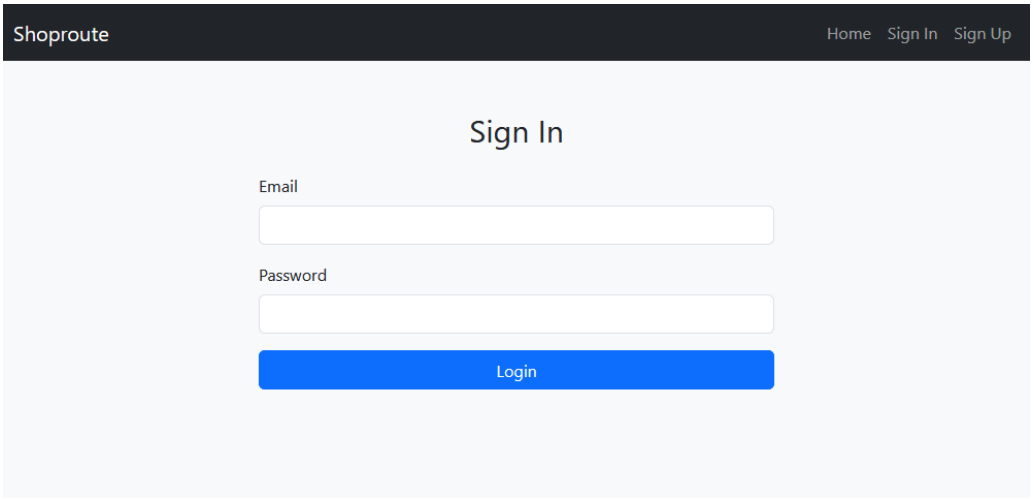


Рис. 4.3 Форма аутентифікації

Після аутентифікації користувача перенаправляє на головну сторінку, де відображається список доступних магазинів. На рисунку 4.4 проілюстровано дану головну сторінку. Кожен магазин має унікальну мапу з координатною сіткою, до яких прив’язано полиці, товари, вхід і касову зону. Користувач може обрати один із доступних магазинів для перегляду асортименту та побудови маршруту.

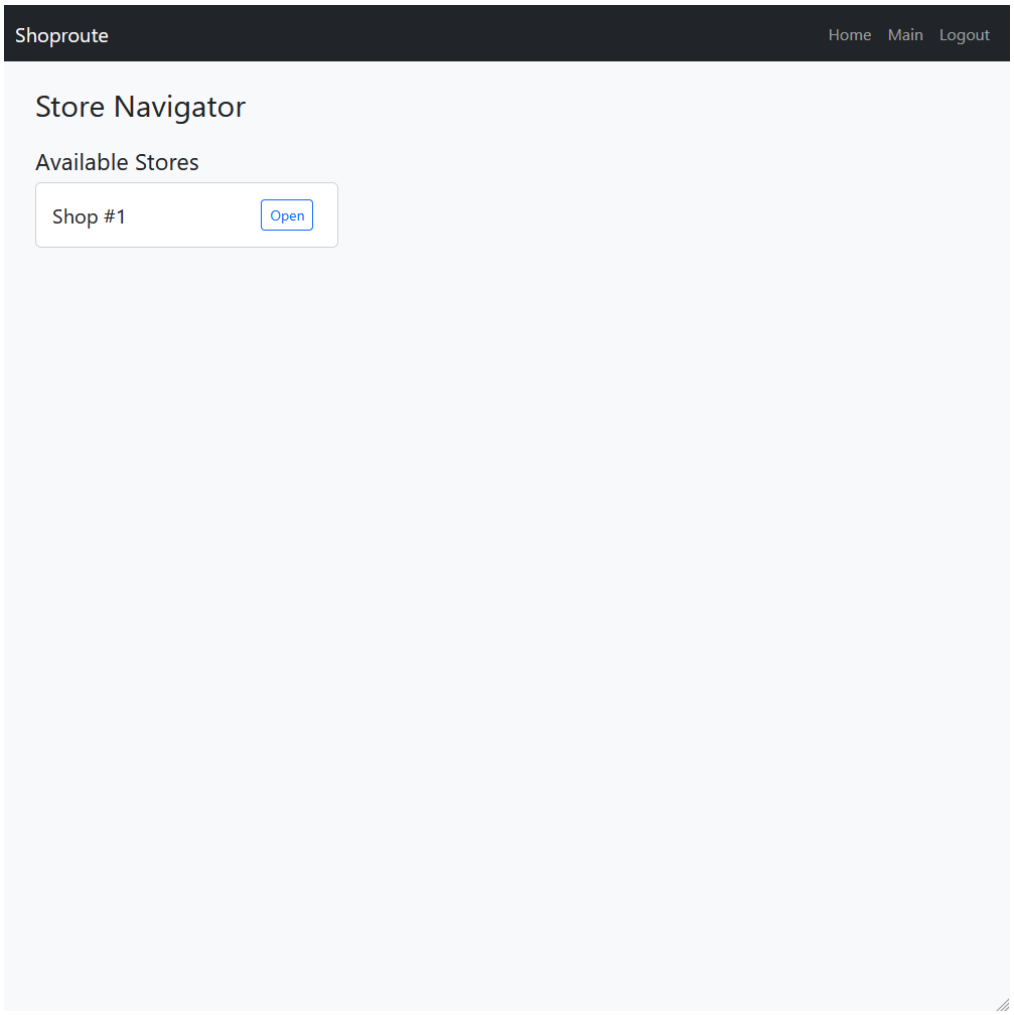


Рис. 4.4 Головна сторінка

Після вибору магазину користувач бачить список доступних товарів, які розміщені на полицях у залі. Він може додати товари до кошика, після чого система пропонує побудувати оптимальний маршрут. Результат побудови відображається у вигляді візуалізації, а саме послідовності точок, з'єднаних лініями, що показують шлях від входу через усі потрібні полиці до каси. На рисунку 4.5 проілюстровано даний принцип роботи.

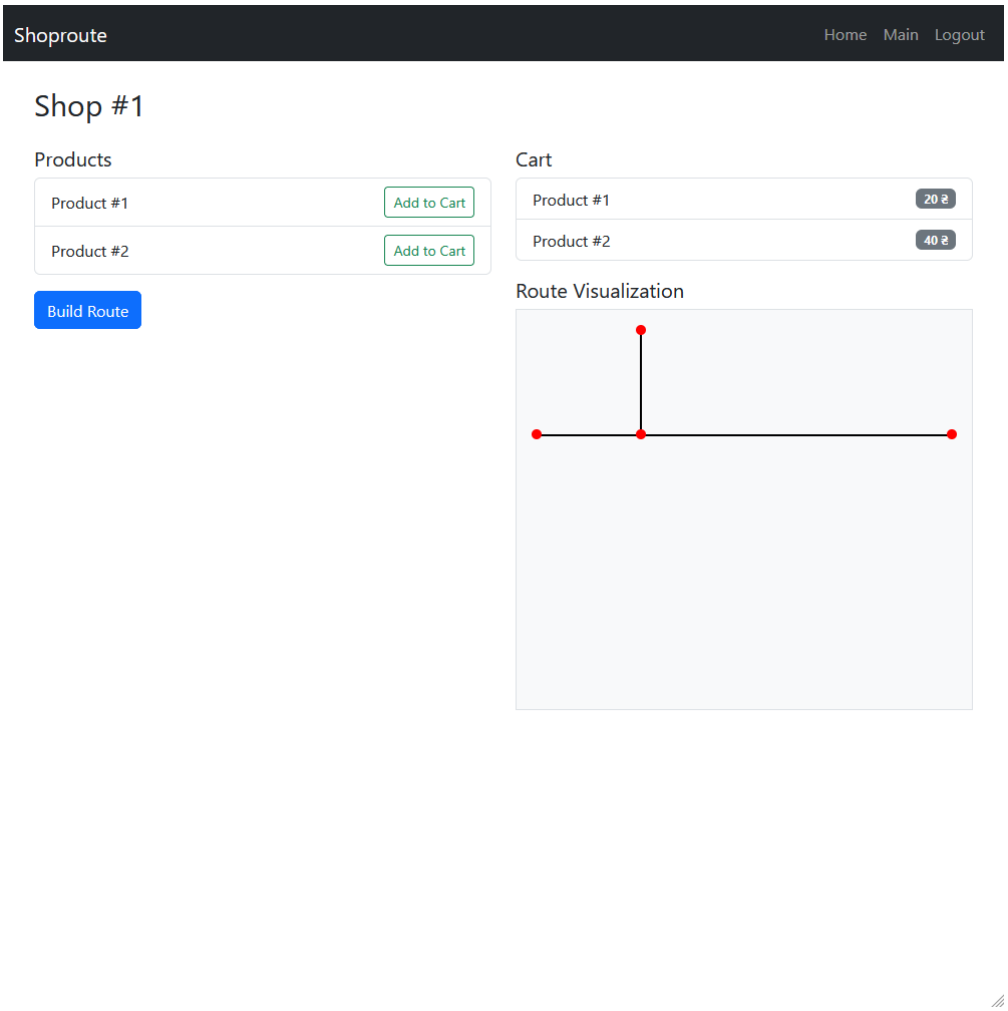


Рис. 4.5 Форма з побудованим шляхом

Таким чином, усі передбачені функціональні можливості – автентифікація, вибір торгової точки, формування кошика та навігація по магазину – реалізовані в інтерактивній формі, що підтверджується представленими екранними формами. Інтерфейс підтримує базову взаємодію з системою і демонструє практичну реалізацію основних сценаріїв роботи користувача.

4.6.2 Тестування та стабільність

Покриття бізнес-логіки модульними тестами дозволило виявити й усунути помилки на ранніх етапах. Алгоритм побудови маршруту показав стабільну поведінку у типовому та граничному випадках. Вручну протестовано основні

сценарії використання API – система передбачувано реагує як на валідні, так і на некоректні запити.

4.6.3 Гнучкість та розширюваність

Архітектура реалізованої системи побудована з урахуванням принципів модульності, слабкого зв'язування між компонентами та суворого поділу обов’язків. Завдяки цьому програмне забезпечення є легко розширюваним як на рівні функціональності, так і на рівні технічної інтеграції з іншими системами. Усі критичні бізнес-процеси – зокрема побудова маршруту, керування структурою магазину та обробка запитів користувачів – зосереджені в доменному рівні та реалізовані у вигляді незалежних, тестованих сервісів.

Це створює широкі можливості для подальшого розвитку системи:

– підтримка нових типів торгових точок – завдяки загальній моделі магазину та її реалізації через параметризовані координати, система може адаптуватися не лише до супермаркетів, а й до інших типів торгових приміщень — наприклад, аптек, складів, гіпермаркетів або торгових центрів з багатьма рівнями.

– можливість впровадження складніших алгоритмів побудови маршруту – поточна реалізація використовує комбінацію алгоритму Дейкстри та жадібного підходу, однак структура дозволяє безболісно замінити або розширити модуль маршрутизації іншими алгоритмами, наприклад, алгоритмами з урахуванням пріоритетів товарів, часових обмежень або потоків відвідувачів.

– інтеграція зі сторонніми сервісами – оскільки система взаємодіє через REST API, вона легко піддається інтеграції з зовнішніми системами обліку товарів, мобільними застосунками, CRM або платформами аналітики. Це відкриває можливість автоматичного оновлення товарного асортименту, синхронізації з наявністю на складі та підключення аналітичних інструментів.

– підтримка кількох планувань одного магазину – у майбутньому можна реалізувати можливість створення кількох конфігурацій однієї й тієї ж торгової

точки. Це дозволить обирати активне планування залежно від дати або навантаження.

– інтеграція з клієнтськими застосунками – реалізована логіка готова до підключення фронтенду у вигляді мобільного застосунку або вебінтерфейсу. Через доступне API можна реалізувати візуалізацію маршруту на карті магазину, інтерактивний список товарів, відстеження прогресу покупки, push-нагадування тощо.

Завдяки гнучкій структурі системи та відсутності жорсткої прив'язки до конкретних фреймворків або реалізацій, реалізоване програмне забезпечення має потенціал для використання як основа складніших, масштабованих рішень у сфері роздрібної торгівлі.

ВИСНОВКИ

Результатом виконаної роботи стало серверне програмне забезпечення, призначене для навігації по супермаркету на основі списку товарів. Система дозволяє користувачам формувати перелік необхідних продуктів, після чого забезпечує побудову оптимального маршруту торговим залом із урахуванням просторової структури магазину. Це рішення спрямоване на скорочення часу перебування у супермаркеті, підвищення зручності здійснення покупок та покращення загального користувацького досвіду.

У ході виконання кваліфікаційної роботи було досягнуто наступних результатів:

1. Проведено аналіз предметної області, виявлено ключові проблеми навігації в супермаркетах, розглянуто сучасні підходи до внутрішньої маршрутизації, порівняно існуючі програмні рішення, визначено їх обмеження та обґрунтовано доцільність створення нового застосунку. Досліджено особливості представлення карти магазину у вигляді графа та способи зберігання такої структури в базі даних.

2. Розроблено вимоги до програмного забезпечення, зокрема функціональні сценарії для адміністратора і користувача, а також нефункціональні вимоги до архітектури. Спроєктовано модель предметної області, визначено доменні сутності та їхні зв’язки, описано архітектуру застосунку з чітким поділом на доменний, аплікаційний та інфраструктурний рівні.

3. Реалізовано серверний застосунок на основі стеку Java та Spring Boot. Увесь функціонал побудови маршруту, управління магазином і товарами реалізовано в доменному шарі. Забезпечено збереження даних у PostgreSQL з автоматичним мігруванням схем через Liquibase. Реалізовано безпечну автентифікацію та авторизацію з використанням JWT і OAuth2.

4. Проведено модульне тестування бізнес-логіки, зокрема побудови маршруту, обробки кошика, роботи графової структури магазину. Для REST API виконано ручне тестування через Postman. Результати перевірок підтверджують коректність реалізації алгоритмів і відповідність системи функціональним вимогам.

Таким чином, поставлені в межах кваліфікаційної роботи завдання виконано в повному обсязі, а розроблена система може слугувати основою для подальшого розвитку та впровадження в реальні торговельні середовища.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Макаров А.Д., Замрій І.В. Побудова гнучкої архітектури через ізоляцію доменної логіки веб-застосунках. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, С. 69-71.

2. Макаров А.Д., Замрій І.В. Застосування oauth2 і jwt для підвищення безпеки у веб-сервісі. // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025». 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, С. 293-295.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Walmart | Save Money. Live better. Walmart.com. URL: https://www.walmart.com/.

2. Indoor Positioning System Company with Advanced Navigation Solutions | Mapsted. Mapsted Technology. URL: https://www.mapsted.com/.

3. Target : Expect More. Pay Less. Target : Expect More. Pay Less. URL: https://www.target.com/.

4. Bring! Shopping List App for iOS & Android. Bring!. URL: https://www.getbring.com/en/home.

5. IntelliJ IDEA | Features. JetBrains. URL: https://www.jetbrains.com/idea/features/.

6. JDK 21 Documentation - Home. Oracle Help Center. URL: https://docs.oracle.com/en/java/javase/21/.

7. Spring Boot :: Spring Boot. Spring | Home. URL: https://docs.spring.io/spring-boot/index.html.

8. Spring Security :: Spring Security. Spring | Home. URL: https://docs.spring.io/spring-security/reference/index.html.

9. RFC 6690: Constrained RESTful Environments (CoRE) Link Format. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc6690.

10. RFC 7519: JSON Web Token (JWT). IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7519.

11. RFC 6749: The OAuth 2.0 Authorization Framework. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc6749.

12. PostgreSQL. PostgreSQL. URL: https://www.postgresql.org/.

13. Liquibase Community. Liquibase Community. URL: https://www.liquibase.org/.

14. MapStruct – Java bean mappings, the easy way!. MapStruct – Java bean mappings, the easy way!. URL: https://mapstruct.org/.

15. Project Lombok. Project Lombok. URL: https://projectlombok.org/.

16. Postman: The World's Leading API Platform. Postman API Platform. URL: https://www.postman.com/.

17. Home. Docker Documentation. URL: https://docs.docker.com/.

18. Мартін Р. Чистий код: створення і рефакторинг за допомогою Agile / пер. з англ. І. Бондар-Терещенко. Харків : "Ранок" : Фабула, 2023. 448 с.

19. Мартін Р. Чиста архітектура: Мистецтво створення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. 2-ге вид. Харків : "Ранок" : Фабула, 2023. 368 с.

20. Ornelas G. Clean Architecture with Spring Boot. Baeldung. URL: https://www.baeldung.com/spring-boot-clean-architecture.

21. Find Shortest Paths from Source to all Vertices using Dijkstra’s Algorithm - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/.

22. Greedy Algorithms - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/greedy-algorithms/.

23. RFC 7523: JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7523.

24. OAuth 2.0 resource server JWT: spring security. Spring Documentation. URL: https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html

25. JUnit 5. JUnit. URL: https://junit.org/junit5/.

26. Mockito framework site. Mockito framework site. URL: https://site.mockito.org/.

ДОДАТОК А. ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНОКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для навігації по магазину на основі списку товарів

Виконавстудент 4 курсу
групи ПД-41
МакаровАндрій Дмитрович
Керівник роботи
д.т.н, проф., завідувач кафедриІПЗ Замрій Ірина Вікторівна

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи –** спрощення навігації покупців у магазині шляхом побудови маршруту на основі списку товарів.
- **Об’єкт дослідження –** процес орієнтації покупця в просторі магазину під час здійснення покупок.
- **Предмет дослідження –** програмне забезпечення для навігації по магазину.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. *Огляд та аналіз сучасних підходів і технологій побудови маршрутів у приміщеннях на основі списку товарів*
2. *Проєктування програмного забезпечення для навігації по магазину на основі списку товарів.*
3. *Програмна реалізація та документування серверного застосунку для побудовимаршруту по магазину.*
4. *Тестування роботи програмного забезпечення на основі типових сценаріїв використання.*

3

АНАЛІЗ АНАЛОГІВ

Параметр	Walmart App	Mapsted SDK	Target App	Bring! Shopping List	Shoproute
Побудова маршруту	+	+	+	-	+
Локалізація товарів	+	+	+	-	+
Доступність в Україні	-	-	-	+	+
Підтримка списків	+	+	+	+	+
Платформа	Android/iOS	Android/iOS	Android/iOS	Android/iOS/Web	Web

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні:

1. Системаповинна надавати користувачу список товарів магазину.
2. Користувач має можливість додавати товари до персонального списку.
3. Система повинна визначати розташування кожного товару у магазині.
4. Система має будувати оптимальний маршрут по магазину з урахуванням розміщення товарів.
5. Користувач маєзмогу переглянути згенерований маршрут у вигляді схеми.

Нефункціональні:

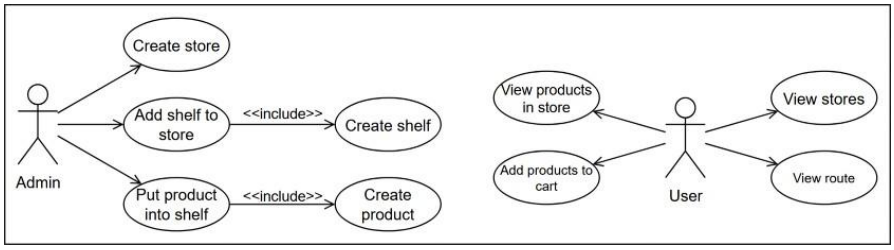
1. Архітектура повинна забезпечувати модульністьта гнучкість системи.
2. Інтерфейс API має бути RESTfulта задокументований.
3. Код повинен бути написаний з урахуванням принципів чистого коду.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

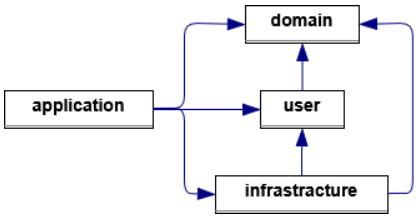
6

ДІАГРАМА ОСНОВНИХ ВАРІАНТІВ ВИКОРИСТАННЯ



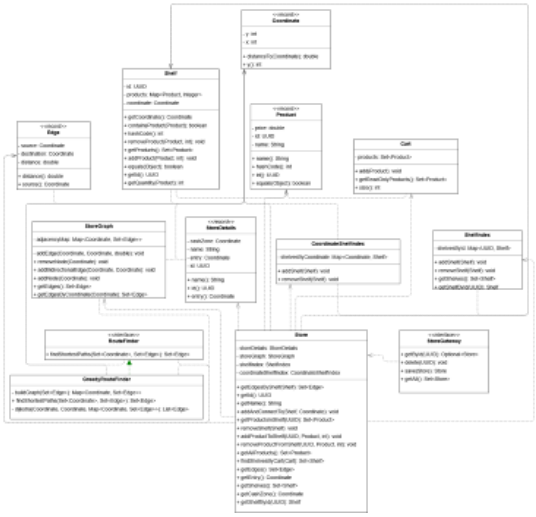
7

ДІАГРАМА МОДУЛІВ



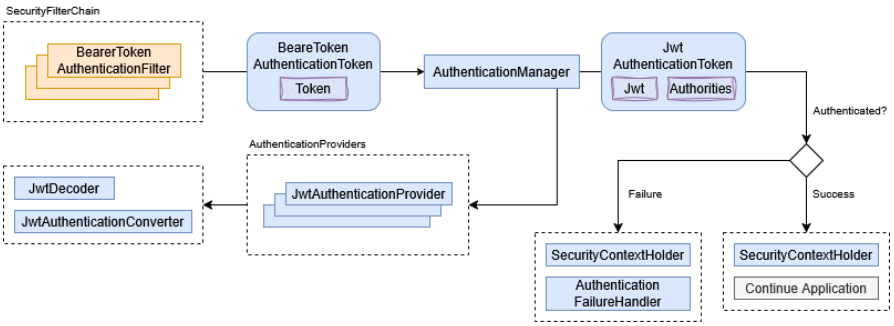
8

ДІАГРАМА КЛАСІВ ДОМЕННОГО МОДУЛЯ



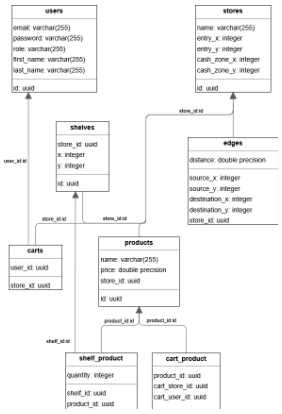
9

ДІАГРАМА ПОТОКУ АУТЕНТИФІКАЦІЇ



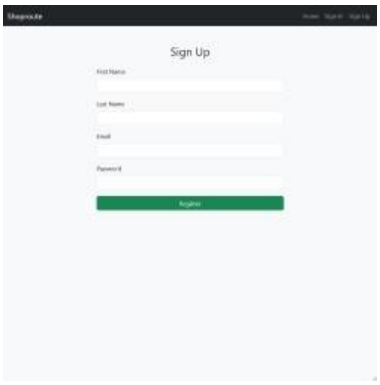
10

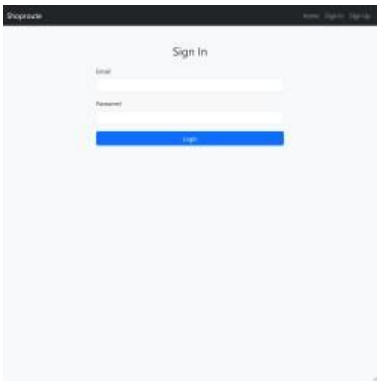
ER-ДІАГРАМА



11

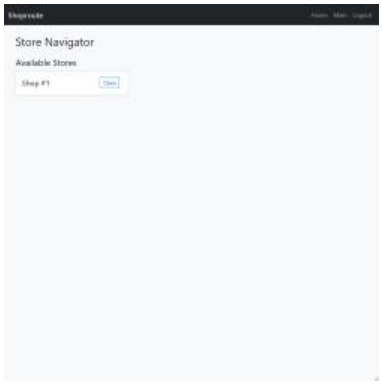
ЕКРАННІ ФОРМИ





12

ЕКРАННІ ФОРМИ



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. *Макаров А.Д., Замрій І.В. ПОБУДОВА ГНУЧКОЇ АРХІТЕКТУРИ ЧЕРЕЗ ІЗОЛЯЦІЮ ДОМЕННОЇ ЛОГІКИ ВЕБ-ЗАСТОСУНКАХ. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, С. 69-71.*
2. *Макаров А.Д., Замрій І.В. ЗАСТОСУВАННЯ OAUTH2 І JWT ДЛЯ ПІДВИЩЕННЯ БЕЗПЕКИ У ВЕБ-СЕРВІСІ. // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025». 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна , С. 293-295.*

ВИСНОВКИ

1. *Оглянуто та проаналізовано сучасні підходи і технології побудови маршрутів у приміщенні на основі списку товарів, включаючи графові моделі, алгоритми навігації та прикладні реалізації у комерційних рішеннях. Вивчено особливості роботи таких систем у реальних супермаркетах та виокремлено їх ключові переваги і недоліки*
2. *Спроєктовано програмне забезпечення для навігації по магазину на основі списку, з урахуванням функціональних і нефункціональних вимог. Визначено архітектуру системи з поділом на доменну, аплікаційну та інфраструктурну частини, а також розроблено модель предметної області.*
3. *Реалізовано серверний застосунок для побудови маршруту по магазину, який забезпечує роботу з координатною картою магазину, обробку кошика користувача та побудову оптимального маршруту на основі гібридного алгоритму. Реалізовано REST API з підтримкою авторизації на основі JWT.*
4. *Протестовано роботу програмного забезпечення на основі типових сценаріїв використання, включаючи створення магазину, додавання полиць і товарів, формування кошика та побудову маршруту. Проведено модульне тестування бізнес-логіки та ручову перевірку взаємодії через API.*

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
public final class Cart {
    private final Set<Product> products;

    public Cart(Set<Product> products) {
        this.products = products;
    }

    public int size() {
        return products.size();
    }

    public void add(Product product) {
        products.add(Objects.requireNonNull(product,
"Product cannot be null"));
    }

    public Set<Product> getReadOnlyProducts() {
        return Collections.unmodifiableSet(products);
    }
}
final class CoordinateShelfIndex {
    private final Map<Coordinate, Shelf>
shelvesByCoordinate = new HashMap<>();

    public void addShelf(Shelf shelf) {
        if
(shelvesByCoordinate.containsKey(shelf.getCoordinate()
)) {
            throw new
ShelfAlreadyExistsException(ExceptionMessages.Shelf.
ALREADY_EXISTS_COORDINATE);
        }
        shelvesByCoordinate.put(shelf.getCoordinate(),
shelf);
    }

    public void removeShelf(Shelf shelf) {
        if
(!shelf.equals(shelvesByCoordinate.get(shelf.getCoordin
ate()))) {
            throw new
ShelfNotFoundException(ExceptionMessages.Shelf.NO
T_FOUND_COORDINATE);
        }

shelvesByCoordinate.remove(shelf.getCoordinate());
    }
}
public record Product(UUID id, String name, double
price) {
    @Override
    public boolean equals(Object o) {
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Product product = (Product) o;
        return Objects.equals(id, product.id);
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(id);
    }
}
public final class Shelf {
    private final UUID id;
    private final Coordinate coordinate;
    private final Map<Product, Integer> products;

    public Shelf(UUID id, Map<Product, Integer>
products, Coordinate coordinate) {
        this.id = Objects.requireNonNull(id, "Cart ID cannot
be null");
        this.coordinate =
Objects.requireNonNull(coordinate, "Coordinate cannot
be null");
        this.products = Objects.requireNonNull(products,
"Products cannot be null");
    }

    public Shelf(UUID id, Coordinate coordinate) {
        this(id, new HashMap<>(), coordinate);
    }

    public UUID getId() {
        return id;
    }

    public Coordinate getCoordinate() {
        return coordinate;
    }

    public void addProduct(Product product, int quantity) {
        if (quantity <= 0) {
            throw new
IllegalArgumentException(ExceptionMessages.Validatio
n.POSITIVE_QUANTITY);
        }
        products.merge(Objects.requireNonNull(product,
"Product cannot be null"), quantity, Integer::sum);
    }

    public Set<Product> getProducts() {
        return
Collections.unmodifiableSet(products.keySet());
    }

    public boolean containsProduct(Product product) {
        return products.containsKey(product);
    }
```

```
    public void removeProduct(Product product, int
quantity) {
        if (getQuantity(product) <= quantity) {
            products.remove(product);
        }
        products.computeIfPresent(product, (key, value) ->
value - quantity);
    }

    public int getQuantity(Product product) {
        return products.getOrDefault(product, 0);
    }

    @Override
    public boolean equals(Object o) {
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Shelf shelf = (Shelf) o;
        return Objects.equals(id, shelf.id);
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(id);
    }
}
final class ShelfIndex {
    private final Map<UUID, Shelf> shelvesById = new
HashMap<>();

    public void addShelf(Shelf shelf) {
        if (shelvesById.containsKey(shelf.getId())) {
            throw                                    new
ShelfAlreadyExistsException(ExceptionMessages.Shelf.
ALREADY_EXISTS_ID);
        }
        shelvesById.put(shelf.getId(), shelf);
    }

    public void removeShelf(Shelf shelf) {
        if (!shelf.equals(shelvesById.get(shelf.getId()))) {
            throw                                    new
ShelfNotFoundException(ExceptionMessages.Shelf.NO
T_FOUND_WITH_ID);
        }
        shelvesById.remove(shelf.getId());
    }

    public Shelf getShelfById(UUID id) {
        Shelf shelf = shelvesById.get(id);
        if (shelf == null) {
            throw                                    new
ShelfNotFoundException(ExceptionMessages.Shelf.NO
T_FOUND_ID.formatted(id));
        }
        return shelf;
    }

    public Set<Shelf> getShelves() {
        return Set.copyOf(shelvesById.values());
    }
}
public final class Store {
    private final StoreDetails storeDetails;
    private final StoreGraph storeGraph;
    private final ShelfIndex shelfIndex = new ShelfIndex();
    private                final                CoordinateShelfIndex
coordinateShelfIndex = new CoordinateShelfIndex();

    public Store(StoreDetails storeDetails) {
        this.storeDetails = storeDetails;
        this.storeGraph                =                new
StoreGraph(storeDetails.entry(),
storeDetails.cashZone());
    }

    public Store(StoreDetails storeDetails, Set<Shelf>
shelves, Set<Edge> edges) {
        this(storeDetails);
        shelves.forEach(shelf -> {
            storeGraph.addNode(shelf.getCoordinate());
            shelfIndex.addShelf(shelf);
        });
        edges.forEach(edge                               ->
storeGraph.addBidirectionalEdge(edge.source(),
edge.destination()));
    }

    public UUID getId() {
        return storeDetails.id();
    }

    public String getName() {
        return storeDetails.name();
    }

    public Coordinate getEntry() {
        return storeDetails.entry();
    }

    public Coordinate getCashZone() {
        return storeDetails.cashZone();
    }

    public void addAndConnectTo(Shelf shelf, Coordinate
connectTo) {
        coordinateShelfIndex.addShelf(shelf);

storeGraph.addBidirectionalEdge(shelf.getCoordinate(),
connectTo);
        shelfIndex.addShelf(shelf);
    }

    public void removeShelf(Shelf shelf) {
        coordinateShelfIndex.removeShelf(shelf);
        storeGraph.removeNode(shelf.getCoordinate());
        shelfIndex.removeShelf(shelf);
    }

    public Set<Shelf> getShelves() {
        return shelfIndex.getShelves();
    }
```

```
    public Shelf getShelfById(UUID shelfId) {
        return shelfIndex.getShelfById(shelfId);
    }

    public Set<Edge> getEdges() {
        return storeGraph.getEdges();
    }

    public Set<Edge> getEdgesByShelf(Shelf shelf) {
        return
storeGraph.getEdgesByCoordinate(shelf.getCoordinate()
);
    }

    public void addProductToShelf(UUID shelfId, Product
product, int quantity) {
        Shelf shelf = getShelfById(shelfId);
        shelf.addProduct(product, quantity);
    }

    public void removeProductFromShelf(UUID shelfId,
Product product, int quantity) {
        Shelf shelf = getShelfById(shelfId);
        shelf.removeProduct(product, quantity);
    }

    public Set<Product> getAllProducts() {
        return getShelves().stream()
            .flatMap(shelf -> shelf.getProducts().stream())
            .collect(Collectors.toUnmodifiableSet());
    }

    public Set<Product> getProductsInShelf(UUID
shelfId) {
        return getShelfById(shelfId).getProducts();
    }

    public Set<Shelf> findShelvesByCart(Cart cart) {
        return shelfIndex.getShelves().stream()
            .filter(shelf ->
cart.getReadOnlyProducts().stream().anyMatch(shelf::co
ntainsProduct))
            .collect(Collectors.toUnmodifiableSet());
    }
}
public record StoreDetails(
    UUID id,
    String name,
    Coordinate entry,
    Coordinate cashZone
) {
}
final class StoreGraph {
    private final Map<Coordinate, Set<Edge>>
adjacencyMap = new HashMap<>();

    public StoreGraph(Coordinate entry, Coordinate
cashZone) {
        addNode(entry);
        addNode(cashZone);
    }

    public void addNode(Coordinate coordinate) {
        if (adjacencyMap.containsKey(coordinate)) {
            throw new
NodeAlreadyExistsException(ExceptionMessages.Node.
ALREADY_EXISTS);
        }
        adjacencyMap.put(coordinate, new HashSet<>());
    }

    public void removeNode(Coordinate coordinate) {
        adjacencyMap.remove(coordinate);
        adjacencyMap.values()
            .forEach(edges -> edges.removeIf(edge ->
edge.destination().equals(coordinate)));
    }

    public void addBidirectionalEdge(Coordinate
newCoordinate, Coordinate connectTo) {
        if (!adjacencyMap.containsKey(connectTo)) {
            throw new
NodeNotFoundException(ExceptionMessages.Node.NO
T_FOUND);
        }

        double distance =
newCoordinate.distanceTo(connectTo);
        if (distance <= 0) {
            throw new
InvalidDistanceException(ExceptionMessages.Validatio
n.POSITIVE_DISTANCE);
        }

        addEdge(newCoordinate, connectTo, distance);
        addEdge(connectTo, newCoordinate, distance);
    }

    private void addEdge(Coordinate from, Coordinate to,
double distance) {
        adjacencyMap.computeIfAbsent(from, k -> new
HashSet<>())
            .add(new Edge(from, to, distance));
    }

    public Set<Edge> getEdges() {
        return Set.copyOf(adjacencyMap.values().stream()
            .reduce((acc, edges) -> {
                acc.addAll(edges);
                return acc;
            })
            .orElseGet(Set::of));
    }

    public Set<Edge> getEdgesByCoordinate(Coordinate
coordinate) {
        return
Set.copyOf(adjacencyMap.getOrDefault(coordinate,
Set.of()));
    }
}
public interface StoreGateway {
    Set<Store> getAll();
```

```
    Optional<Store> getById(UUID id);

    Store save(Store store);

    void delete(UUID id);
}
public interface RouteFinder {
    Set<Edge> findShortestPaths(Set<Coordinate>
keyPoints, Set<Edge> edges);
}
public class GreedyRouteFinder implements RouteFinder
{
    @Override
    public Set<Edge> findShortestPaths(Set<Coordinate>
keyPoints, Set<Edge> edges) {
        Map<Coordinate, Set<Edge>> graph =
buildGraph(edges);

        Set<Edge> result = new HashSet<>();
        Set<Coordinate> remaining = new
HashSet<>(keyPoints);

        Coordinate current = remaining.iterator().next();
        remaining.remove(current);

        while (!remaining.isEmpty()) {
            Coordinate next = null;
            List<Edge> shortestPath = null;
            double minDistance =
Double.POSITIVE_INFINITY;

            for (Coordinate target : remaining) {
                List<Edge> path = dijkstra(current, target,
graph);
                double distance =
path.stream().mapToDouble(Edge::distance).sum();

                if (distance < minDistance) {
                    minDistance = distance;
                    next = target;
                    shortestPath = path;
                }
            }

            if (shortestPath == null) {
                break;
            }

            result.addAll(shortestPath);
            current = next;
            remaining.remove(current);
        }
        return result;
    }

    private Map<Coordinate, Set<Edge>>
buildGraph(Set<Edge> edges) {
        return edges.stream()
            .collect(Collectors.toMap(
                Edge::source,
                edge -> new HashSet<>(Set.of(edge)),
                (existing, newEdges) -> {
                    existing.addAll(newEdges);
                    return existing;
                }
            ));
    }

    private List<Edge> dijkstra(Coordinate start,
Coordinate end, Map<Coordinate, Set<Edge>> graph) {
        Map<Coordinate, Double> distances = new
HashMap<>();
        PriorityQueue<Coordinate> queue = new
PriorityQueue<>(Comparator.comparingDouble(distance
s::get));
        Map<Coordinate, Edge> previousEdges = new
HashMap<>();
        Set<Coordinate> visited = new HashSet<>();

        distances.put(start, 0.0);
        queue.add(start);

        while (!queue.isEmpty()) {
            Coordinate current = queue.poll();

            if (current.equals(end)) {
                break;
            }

            visited.add(current);

            for (Edge edge : graph.getOrDefault(current,
Collections.emptySet())) {
                Coordinate neighbor = edge.destination();

                if (visited.contains(neighbor)) {
                    continue;
                }

                double newDist = distances.get(current) +
edge.distance();
                if (newDist < distances.getOrDefault(neighbor,
Double.POSITIVE_INFINITY)) {
                    distances.put(neighbor, newDist);
                    previousEdges.put(neighbor, edge);
                    queue.add(neighbor);
                }
            }
        }

        List<Edge> path = new ArrayList<>();
        Coordinate current = end;
        while (previousEdges.containsKey(current)) {
            Edge edge = previousEdges.get(current);
            path.add(edge);
            current = edge.source().equals(current) ?
edge.destination() : edge.source();
        }

        Collections.reverse(path);
        return path;
    }
}
public record Coordinate(int x, int y) {
```

```
    public Coordinate {
        if (x < 0 || y < 0) {
            throw new IllegalArgumentException("Coordinates cannot be negative");
        }
    }

    public double distanceTo(Coordinate other) {
        return Math.sqrt(Math.pow((double) x - other.x, 2) + Math.pow((double) y - other.y, 2));
    }
}
public record Edge(Coordinate source, Coordinate destination, double distance) {
}
```