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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка серверної частини Web-застосунку для аптеки на мові Java з використанням фреймворку Angular»

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

_____ Іван ПОЛТАВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Іван ПОЛТАВСЬКИЙ

Керівник: Владислав ЯСКЕВИЧ
к.т.н.

Рецензент: _____

Київ 2024

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
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Полтавському Івану Андрійовичу

1. Тема кваліфікаційної роботи: «Розробка серверної частини Web-застосунку для аптеки на мові Java з використанням фреймворку Angular»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Владислав ЯСКЕВИЧ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література про розробку веб-додатків, технічні відомості з питань архітектури програмного забезпечення, технічна документація для засобів створення програмного забезпечення.

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

1. Аналіз предметної області.
2. Аналіз інструментів для розробки серверної частини веб-додатку.
3. Проектування, розробка та тестування серверної частини додатку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Схема представлення MVC архітектури.
5. Схема представлення REST API.
6. Схема представлення трьохрівневої архітектури.
7. Діаграма варіантів використання.
8. Діаграма класів.
9. Діаграма бази даних.
10. Демонстрація роботи API.
11. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих технічних рішень для реалізації програмного забезпечення	14.03-17.03.2024	
4	Проектування веб-застосунку	18.03-24.03.2024	
5	Реалізація веб-застосунку	25.03-21.04.2024	
6	Тестування веб-застосунку	22.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Іван ПОЛТАВСЬКИЙ

(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 1 табл., 49 рис., 15 джерел.

Мета роботи – підтримка процесу вибору, бронювання та покупки лікарських препаратів онлайн.

Об'єкт дослідження – процес вибору, бронювання та покупки лікарських препаратів онлайн.

Предмет дослідження – розробка серверної частини додатку "Farmakon" для покупки та бронювання лікарських препаратів.

Короткий зміст роботи: В цій бакалаврській роботі проведено аналіз існуючих рішень для замовлення та бронювання лікарських препаратів, виявлені їх переваги та недоліки. Проведено аналіз архітектурних підходів та інструментів для створення програмного забезпечення. Здійснено проектування архітектури додатку та, на основі цього, програмно реалізовані основні вимоги до застосунку, а саме: реєстрація, автентифікація та авторизація користувача, покупка та бронювання лікарських засобів, перевірка рецептів для рецептурних препаратів, можливість створювати нові та редагувати наявні препарати для адміністратора. В роботі використано мову програмування Java, фреймворк Spring Boot, СУБД PostgreSQL, бібліотеку для роботи з базою даних Hibernate та Postman для тестування додатку.

Сферою використання застосунку є онлайн продаж лікарських засобів.

КЛЮЧОВІ СЛОВА: JAVA, ВЕБ-ЗАСТОСУНОК, SPRING BOOT, ЛІКАРСЬКІ ПРЕПАРАТИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

REST API - Representational State Transfer Application Programming Interface

HTTP - HyperText Transfer Protocol

JSON - JavaScript Object Notation

URL - Uniform Resource Locator

MVC - Model-View-Controller

HTML - HyperText Markup Language

JS - JavaScript

JVM - Java Virtual Machine

JAR - Java ARchive

WAR - Web Application ARchive

БД - База Даних

СУБД - Система Управління Базами Даних

SQL - Structured Query Language

JDBC - Java Database Connectivity

ORM - Object-Relational Mapping

HQL - Hibernate Query Language

UML - Unified Modeling Language

DTO - Data Transfer Object

XML - EXtensible Markup Language

JWT - JSON Web Token

ВСТУП

Сучасний світ технологій дуже обширний та впливає на різні сфери життя, і фармацевтична галузь не є виключенням. Використання програмного забезпечення спрощує процес замовлення лікарських препаратів для клієнтів та допомагає бізнесу простіше та ефективніше надавати послуги. За допомогою таких програмних рішень користувачам стає набагато простіше оформлювати замовлення, оскільки це можна робити з будь-якої точки в будь-який зручний час, а власникам аптек стає куди зручніше вести облік препаратів, аналізувати ринок та, на основі цих аналізів, підлаштовувати під нього роботу бізнесу.

Тему цієї бакалаврської роботи можна вважати актуальною, оскільки популярність веб-застосунків для різних сфер з кожним роком все дедалі зростає і користувачі роблять все більше замовлень різної продукції саме в онлайн режимі. Конкурентоспроможність з іншими бізнесами без власного веб-застосунку стає більш важкою задачею, оскільки впізнаваність бренду завдяки подібним програмним рішенням розростається.

Метою роботи є підтримка процесу вибору, бронювання та покупки лікарських препаратів онлайн.

Об’єктом даного дослідження є бронювання та покупка лікарських препаратів онлайн.

Предметом дослідження є розробка серверної частини додатку “Farmakon” для покупки та бронювання лікарських препаратів.

Завданням роботи є:

1. Провести аналіз існуючих засобів для замовлення та бронювання ліків онлайн, визначити їх переваги та недоліки.
2. Визначити вимоги до програмного забезпечення на основі результатів дослідження існуючих рішень.
3. Провести огляд технічних рішень для побудови веб-застосунку для замовлення ліків, включаючи архітектурні підходи, використання різних API, взаємодію з базами даних, засоби забезпечення безпеки та інші аспекти розробки.

4. Розробити архітектуру додатку на основі проведеного огляду архітектурних підходів.

5. Провести огляд IT-засобів, які можуть бути використані при розробці програмного забезпечення кваліфікаційної роботи бакалавра.

6. На основі розробленої архітектури розробити моделі та сервіси для реалізації бізнес-логіки.

7. Налаштувати базу даних та розробити DAO рівень для роботи з нею.

8. Провести тестування серверної частини додатку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості веб-додатків для замовлення продукції

У сучасному світі важко уявити успішний бізнес без власного веб-додатку, оскільки це стає не лише зручним способом взаємодії з клієнтами, але і необхідністю для підтримки конкурентоспроможності. Веб-додатки дозволяють здійснювати продажі, надавати послуги та взаємодіяти зі споживачами за будь-яких зручних для них умов. Це відкриває нові можливості для бізнесу, оскільки він може обслуговувати клієнтів по всьому світу, розширюючи свій ринок та збільшуючи потенційний прибуток. [1]

Веб-додатки значно підвищують рівень зручності для користувачів. Клієнти можуть переглядати каталоги товарів, порівнювати ціни, читати відгуки інших клієнтів і робити замовлення, не виходячи з дому. Це скорочує час і зусилля, які вони витрачають на здійснення покупок. Зручний інтерфейс і можливість швидко знайти потрібний товар сприяють позитивному користувальницькому досвіду, сприяючи підвищенню лояльності клієнтів. [1], [2]

Веб-додатки дозволяють підприємствам ефективніше керувати своїми операціями. Вони можуть автоматизувати багато процесів, такі як обробка замовлень, управління запасами та логістика, зменшуючи витрати на робочу силу та ризик помилок. Автоматизація дозволяє швидко реагувати на зміни ринку та попиту і забезпечити безперебійну доставку продукції клієнтам. [2]

Програмне забезпечення дозволяє підприємствам збирати та аналізувати дані про клієнтів та їхні замовлення. Цю інформацію можна використовувати для вдосконалення маркетингових стратегій, персоналізації пропозицій і покращення послуг. Аналітика даних допомагає зрозуміти потреби клієнтів, що, у свою чергу, дозволяє компаніям надавати більш відповідні продукти та послуги, тим самим підвищуючи задоволеність клієнтів.

Важливим аспектом також є конкурентоспроможність.[2] У сучасному світі, де цифровізація це звичайне явища для будь-чого, наявність веб-додатку для замовлення товарів онлайн стає нормою. Підприємства, які не адаптуються до нових умов, ризикують втратити клієнтів на користь конкурентів із більш передовими технологіями. Веб-додатки дозволяють не відставати від ринку і навіть випереджати своїх конкурентів, надаючи клієнтам більш зручний і сучасний спосіб здійснення покупок.

Веб-додатки також допомагають підприємствам підвищити видимість і впізнаваність бренду.[2] Інтернет є основним джерелом інформації для багатьох людей, і наявність гарного веб-додатку може значно підвищити довіру до бренду. Він також пропонує можливість інтегрувати різні маркетингові інструменти, такі як соціальні мережі, електронна пошта та пошукова оптимізація, щоб залучати нових клієнтів і утримувати існуючих.

Розробка веб-додатків для замовлення продукції є досить популярним завданням, але, незважаючи на це, все одно підхід до кожного додатку різний, в залежності від потреб бізнесу, його задач та його платоспроможності.

Проаналізувавши всі аспекти, можемо виокремити загальні вимоги до веб-додатків для замовлення продукції онлайн : [2]

1. Реєстрація користувача: можливість створення облікового запису юзера та авторизація цього юзера на ресурсі.
2. Керування обліковим записом користувача: можливість змінювати акаунт користувача за його побажанням, а саме: зміна паролю, зміна email, зміна даних про користувача (ім’я, прізвище, дата народження).
3. Перегляд продукції: можливість користувачів переглядати списки продукції. Також потреба в фільтрації цієї продукції за певними критеріями.
4. Оформлення замовлень: можливість для користувача зробити замовлення продукції. Наявність функціоналу додавання в кошик, зміни замовлення та його оплату.

5. Наявність адміністративної панелі (сторінки): можливість для адміністраторів додатку робити зміни в ньому: додавати продукцію, видаляти, змінювати її, тощо.

1.2 Особливості веб-додатків для замовлення лікарських препаратів

Однією з головних переваг веб-додатків для замовлення ліків, як і з випадком з будь-якою іншою продукцією, є доступність. Вони дозволяють користувачам замовляти ліки з будь-якого місця та в будь-який час, використовуючи лише підключення до Інтернету. Особливо це важливо для людей з обмеженими можливостями, людей похилого віку або тих, хто живе у віддалених районах, де аптеки важкодоступні.

Програми для замовлення ліків також значно підвищують зручність для користувачів. [3] Вони можуть легко переглядати ліки, порівнювати ціни, читати інструкції та відгуки інших користувачів, що зменшує потребу особисто йти в аптеку, заощаджуючи час і сили, особливо у випадках, коли ліки потрібно купувати часто.

Зручний інтерфейс і можливість швидкого пошуку необхідних ліків сприяють позитивному користуванню та підвищенню задоволеності клієнтів. Веб-додатки для замовлення ліків дозволяють пацієнтам уникати черг і контактів з іншими людьми, що допомагає знизити ризик інфікування та підвищує безпеку для пацієнтів і персоналу аптек. Можливість безконтактного замовлення та доставки ліків додає ще один рівень захисту.

Важливим аспектом є можливість збирати та аналізувати дані про замовлення лікарських препаратів. Ці дані можна використовувати для покращення послуг, персоналізації пропозицій і вдосконалення реклами. Аптеки, наприклад, можуть нагадувати пацієнтам про повторну покупку ліків або пропонувати знижки на повторні замовлення. Аналітика даних допомагає краще зрозуміти потреби клієнтів і надавати більш індивідуальні послуги.

Лікарські препарати також являються продукцією, тому серйозних

відмінностей від вимог до програмного забезпечення, згаданих у попередньому розділі, бути не може, але все ж таки деяку різницю варто взяти до уваги.

По-перше, обов'язковою є наявність інструкції до препарату на сторінці цього препарату, щоб потенційний користувач одразу міг з нею ознайомитись, та вирішити, чи підходить йому цей лікарській засіб, чи не має він алергії на якісь компоненти у складі товару, тощо.

По-друге, для ліків, які випускаються тільки за рецептом, важливо додати перевірку валідності цього рецепту за його номером, який зберігається в базі даних. Також важливо робити перевірку того, що в рецепті прописаний саме той препарат, який користувач хоче придбати.

По-третє, важливо надавати користувачу список аналогів до препарату, на сторінці якого він знаходиться. Користувачу може не підходити цей конкретний препарат з ряду причин, таких як алергія на деякі компоненти, або зависока ціна на товар. Тому він повинен мати альтернативу, яка буде задовольняти його потреби.

Цільовою аудиторією продукту є користувачі віком від 30 до 50 років[3], оскільки ці люди можуть потребувати регулярного прийому ліків через хронічні захворювання, або просто через послаблений імунітет, і при цьому часто користуються інтернетом для різних потреб, в тому числі онлайн шопінгом. Людям старшого віку вже складніше освоїти інтернет технології, а молодші мають значно менше проблем зі здоров'ям. Це, звісно, не виключає використання додатком людей з іншої вікової категорії, але орієнтуватись треба саме на цей діапазон.

Виходячи з вище наведених даних, можна зробити список додаткових вимог, які можуть висувати потенційні користувачі, а саме:

1. Список аналогів: виводити список аналогів на сторінці препарату для того, щоб користувач міг обрати найкращий для себе варіант, відштовхуючись від ціни продукції, його складу або інших критеріїв.
2. Бонусна система: розробити систему бонусів в додатку, яка нараховує користувачу бонуси при кожній його покупці. Таким чином, після кожної оплати, на рахунок користувача буде надходити певний відсоток від загальної суми чеку цього замовлення. Потім ці бонуси можна буде зняти при

наступній покупці, щоб отримати знижку.

3. Наявність перевірки рецепту: для ліків, які відпускаються по рецепту, потрібно зробити перевірку, де користувач буде вводити номер рецепту і система порівнюватиме, чи є він актуальним (в тому числі і перевіряти чи той препарат вказаний у рецепті).
4. Історія замовлень: виводити список попередніх замовлень користувача в його профілі.

Також важливо розробити функціонал для управління запасами і постачанням, щоб при замовленні ліків зі складу списувалась певна кількість препаратів, яка вказана в замовленні і щоб адміністратор, при надходженні нових товарів, міг легко додати їх в систему.

1.3 Аналіз аналогів

Покупка ліків є звичним процесом для будь-якої людини, тому рішення для їх онлайн замовлення вже присутні на ринку. Проте, як і будь-яка програма або система, вони не є досконалими.

Зазвичай, існує 2 способи замовлення :

– через інструменти бронювання, де користувач не купує препарати, а саме бронює їх у вибраній ним аптеці.

– безпосередньо через сайт конкретної аптеки;

Обидва варіанти мають свої недоліки. Говорячи про недоліки сайтів для бронювань, із головного можна виділити відсутність бонусних програм для клієнтів та неможливість одразу оплатити замовлення, що інколи може бути не зручно.

Говорячи про додатки аптек, часто ці сайти є не оптимізованими та повільними, на деяких з них немає бонусних програм, дуже довгий та складний процес замовлення та важкий для сприйняття інтерфейс.

Одним із прикладів є сайт аптеки «Подорожник» [6]. Заходячи на головну сторінку ми бачимо невеликий список категорій зліва, великий рекламний банер та

вкладку «Переглянуті товари». Також зверху знаходиться поле для пошуку товарів на сайті. Приклад головної сторінки сайту наведено на рисунку 1.3.1.

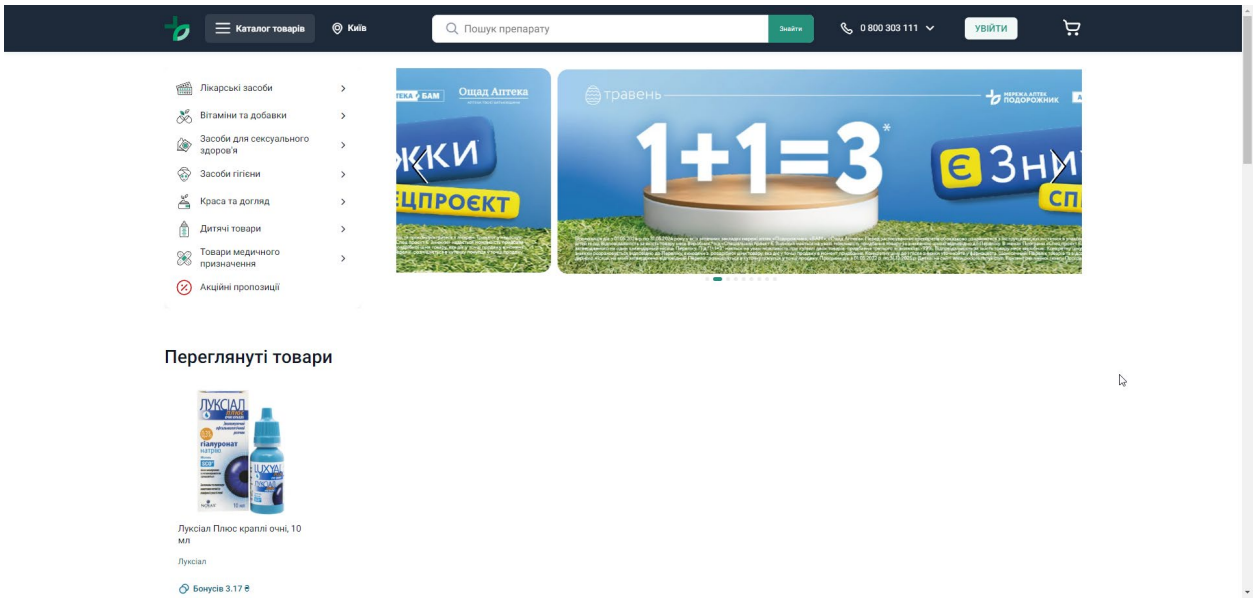


Рис. 1.3.1 Головна сторінка сайту аптеки «Подорожник»

Що на сторінці зі списком препаратів, що на сторінці самого препарату пишеться кількість бонусів, яка буде отримана у випадку покупки цього препарату.

Заходячи на сторінку будь-якого з товарів, бачимо основну картинку препарату, його ціну та умови доставки. Аналоги препаратів на сайті є, але знаходяться в окремій вкладці вгорі, що не завжди може бути зручно для нових користувачів. Приклад сторінки препарату на сайті аптеки «Подорожник» наведений на рисунку 1.3.2.

Нижче розташовується опис препарату, його інструкція, властивості, склад та відгуки користувачів. Майже в самому кінці сторінки знаходиться список «Також вас можуть зацікавити» з препаратами, а ще нижче - список переглянутих товарів.

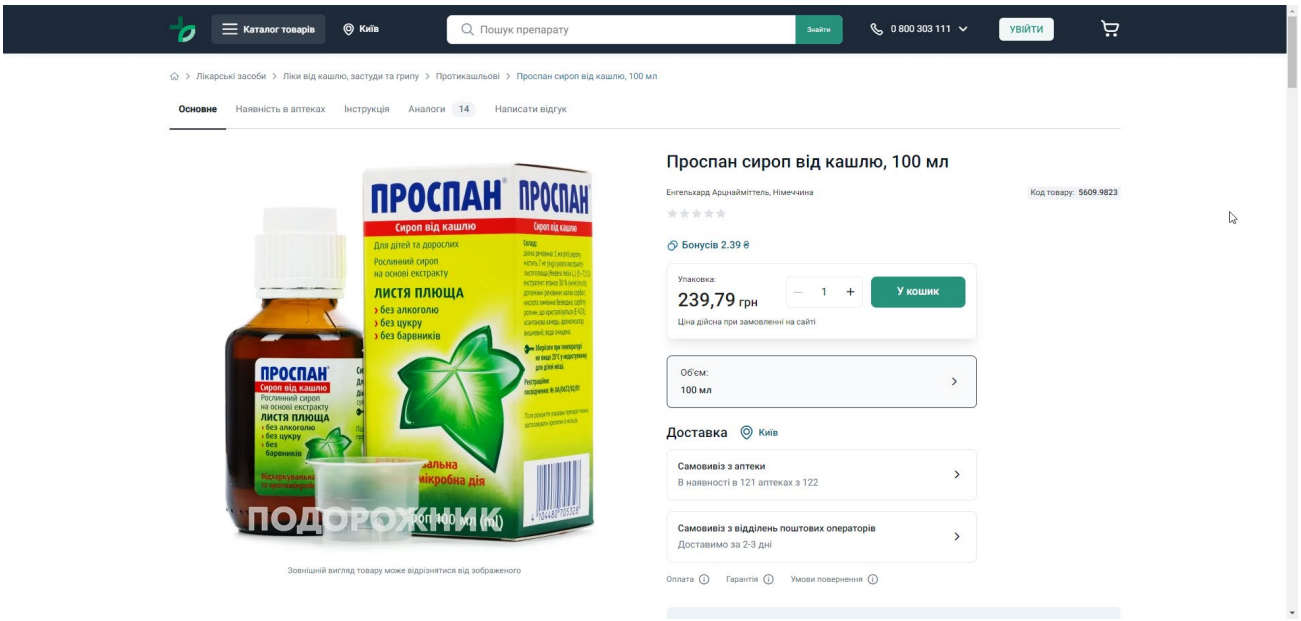


Рис. 1.3.2 Приклад сторінки препарату на сайті аптеки «Подорожник»

При додаванні товару в кошик клієнт одразу в цьому кошику може робити зміни свого замовлення, змінювати кількість вибраних ним препаратів, або видаляти з кошика.

Всі етапи оформлення замовлення розділені по вкладкам. Перша вкладка «Контактні дані», друга – «Спосіб доставки», третя – «Спосіб оплати». Сторінки для оформлення замовлення на сайті наведені на рисунках 1.3.3 – 1.3.6.

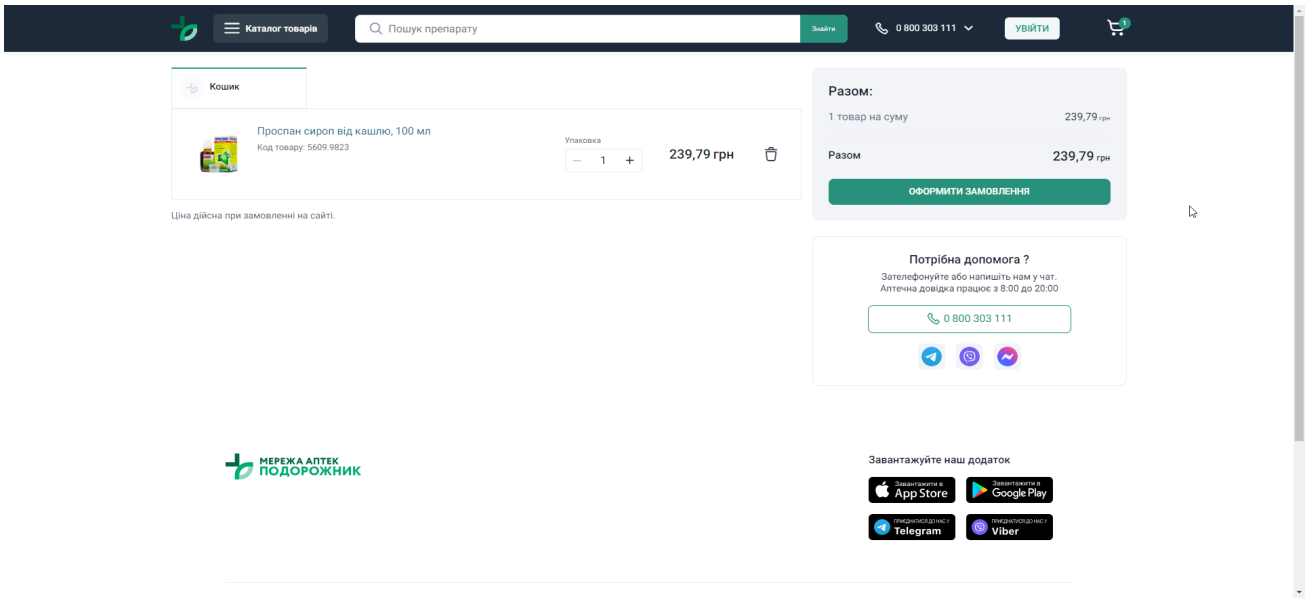


Рис. 1.3.3 Приклад сторінки кошика на сайті аптеки «Подорожник»

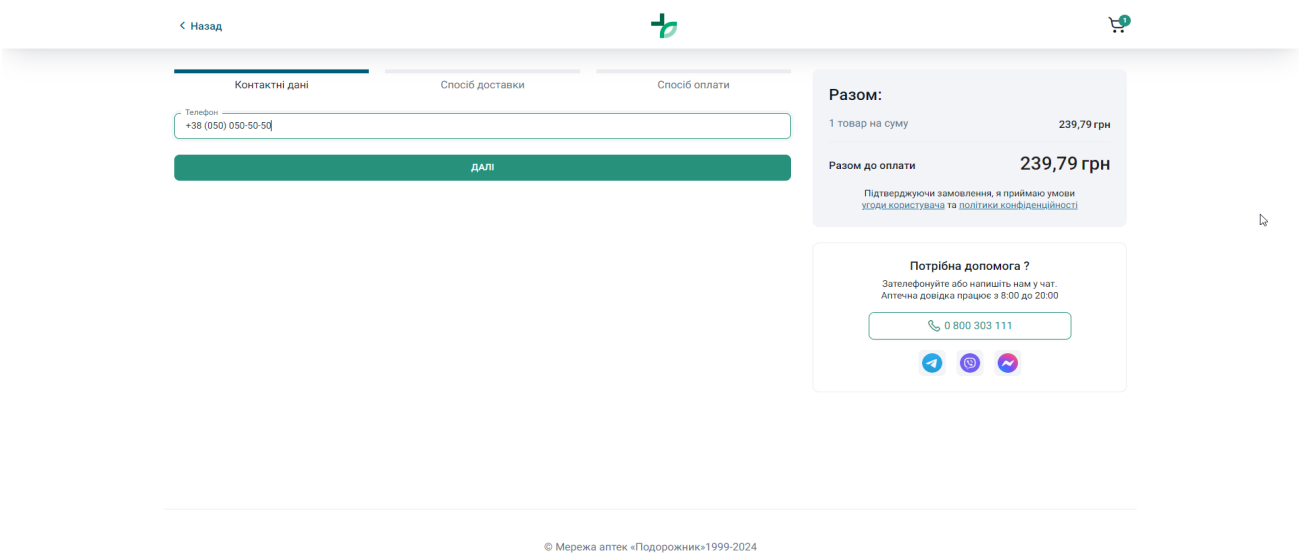


Рис. 1.3.4 Приклад сторінки оформлення контактних даних для замовлення на сайті аптеки «Подорожник»

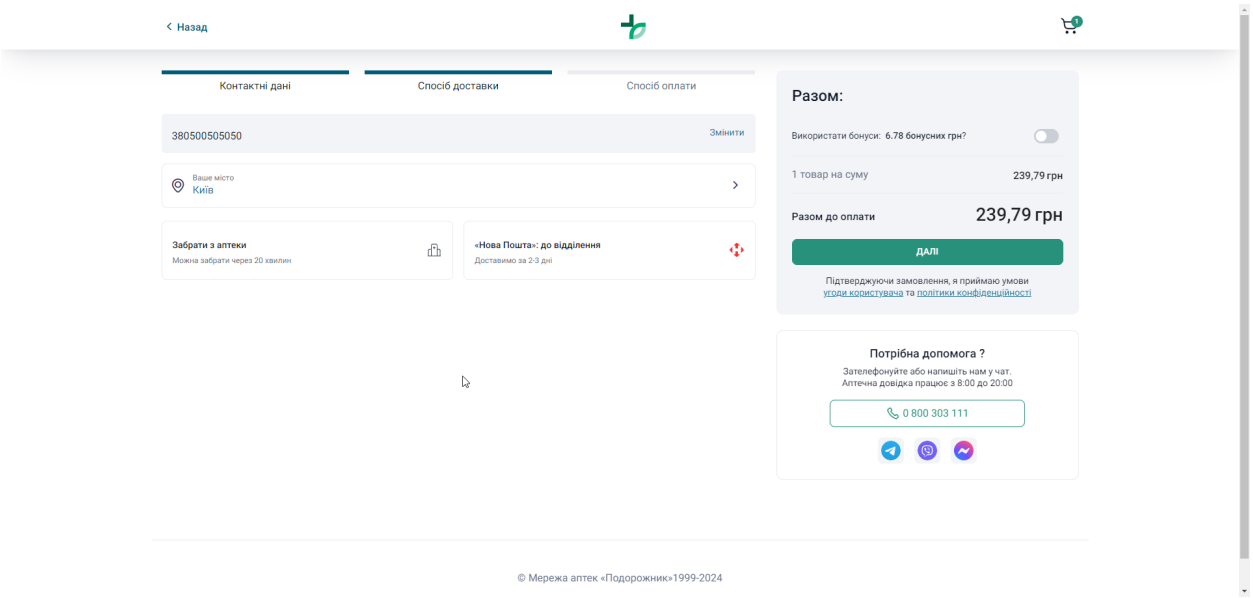


Рис. 1.3.5 Приклад сторінки оформлення даних способу доставки для замовлення на сайті аптеки «Подорожник»

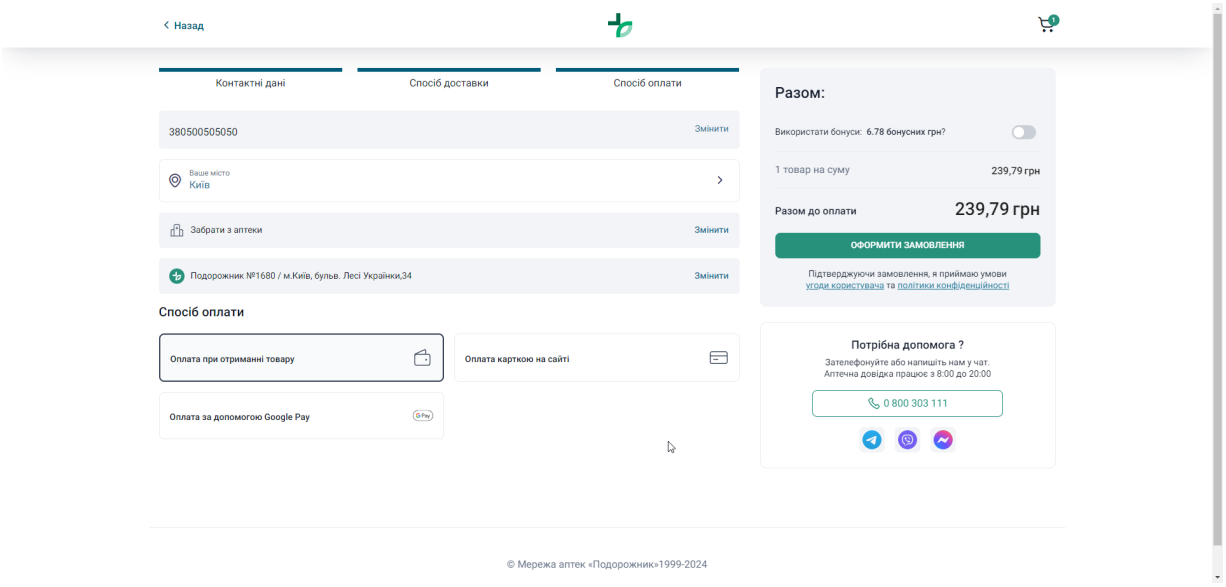


Рис. 1.3.6 Приклад сторінки оформлення даних оплати для замовлення на сайті аптеки «Подорожник»

Після натискання кнопки «Оформити замовлення» йде перенапревлення на сервіс онлайн оплати, у випадку, якщо вибраний пункт оплати на сайті або оплати картою.

До переваг сайту можна віднести:

– легкий в розумінні та сприйнятті інтерфейс;
– можливість оплати прямо на сайті;
– можливість бронювання препаратів;
– наявність бонусної системи;

До недоліків сайту можна віднести:

– не зручний пошук аналогів препарату;
– довгий процес замовлення, розбитий по вкладкам;
– відсутність перевірки валідності рецептів для препаратів, які випускаються тільки по рецептам.

Іншим прикладом є сайт аптеки «911» [7], головна сторінка якого має в собі схожу з попереднім прикладом структуру: зверху поле для пошуку товарів на сайті, зліва список категорій, великий рекламний банер по центру, під якими знаходяться кнопки для додаткового функціоналу, а під ними різні рекламні пропозиції.

Приклад головної сторінки сайту наведено на рисунку 1.3.7.

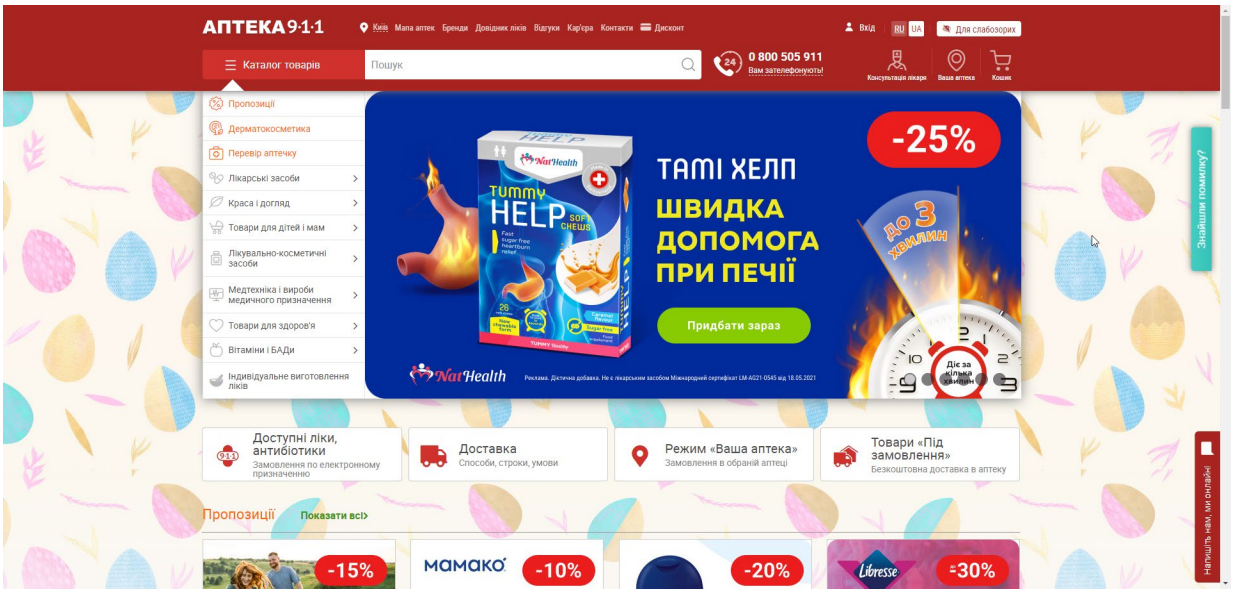


Рис. 1.3.7 Головна сторінка сайту аптеки «911»

Для замовлення препарату дії дуже схожі з минулим прикладом. Для початку треба обрати препарат для замовлення. Сторінка препарату має іншу структуру, ніж в минулому прикладі, але схожу: зліва знаходиться фотографія препарату, а по центру – ціна. Над цінником та фотографіями знаходяться вкладки з інформацією про препарат та його аналогами, що також може бути не дуже зручним для недосвідченого користувача. Правіше від цінника знаходиться інформація про доставку, а саме ціни та терміни. Нижче представлений вибір кількості товару в упаковці (в нашому випадку об’єм), ще нижче – опис препарату. Приклад сторінки препарату представлений на рисунку 1.3.8.

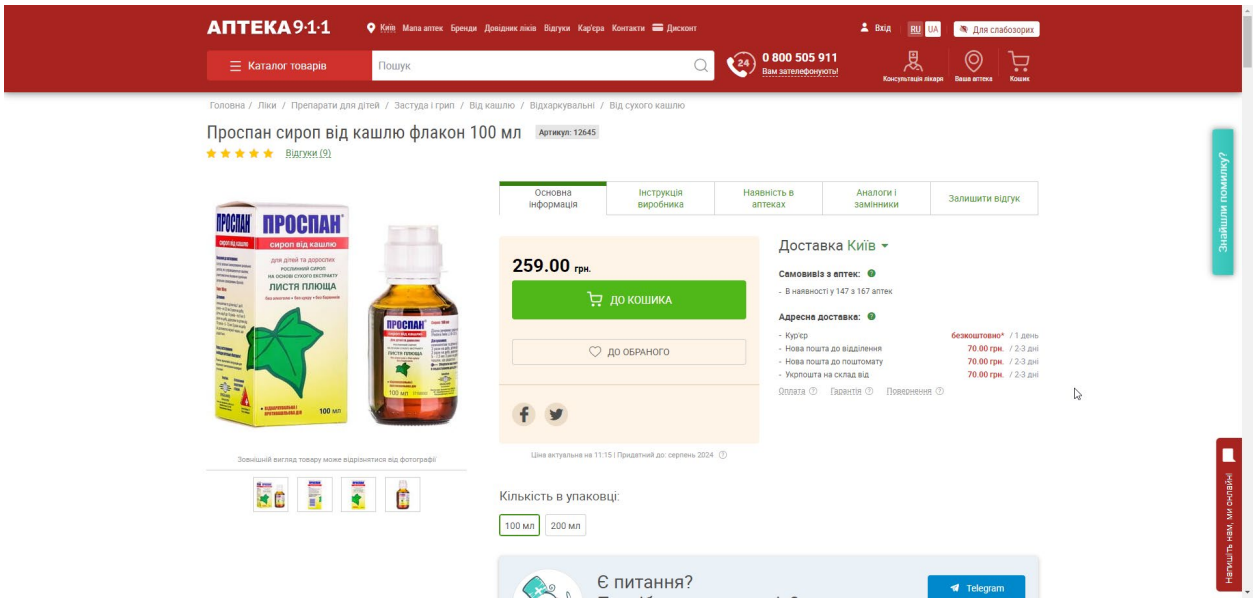


Рис. 1.3.8 Приклад сторінки препарату на сайті аптеки «911»

Після натискання кнопки «До кошика» з’являється віконце з повідомленням, що товар доданий до кошика, і пропонується або оформити замовлення, або продовжувати покупки.

Структура оформлення замовлення схожа з першим прикладом. Всі етапи розділені і перейти до наступного етапу можливо лише після заповнення попереднього.

Першим етапом є «Спосіб доставки», де клієнт обирає як доставити (або в якій з аптек забронювати) замовлення.

Другим етапом є «Адреса доставки», де клієнт обирає куди йому доставити замовлення (у випадку, якщо на першому етапі вибрано самовивіз, то в якій аптеці зробити бронювання).

Третій етап – данні клієнта. Тут користувач обирає, чи є він постійним клієнтом чи він є новим клієнтом, після чого вводить актуальні данні для замовлення, такі як номер телефону, електронна пошта, тощо. Також саме на цьому етапі відбувається вибір способу оплати. Сторінки для оформлення замовлення на сайті наведені на рисунках 1.3.9 – 1.3.11.

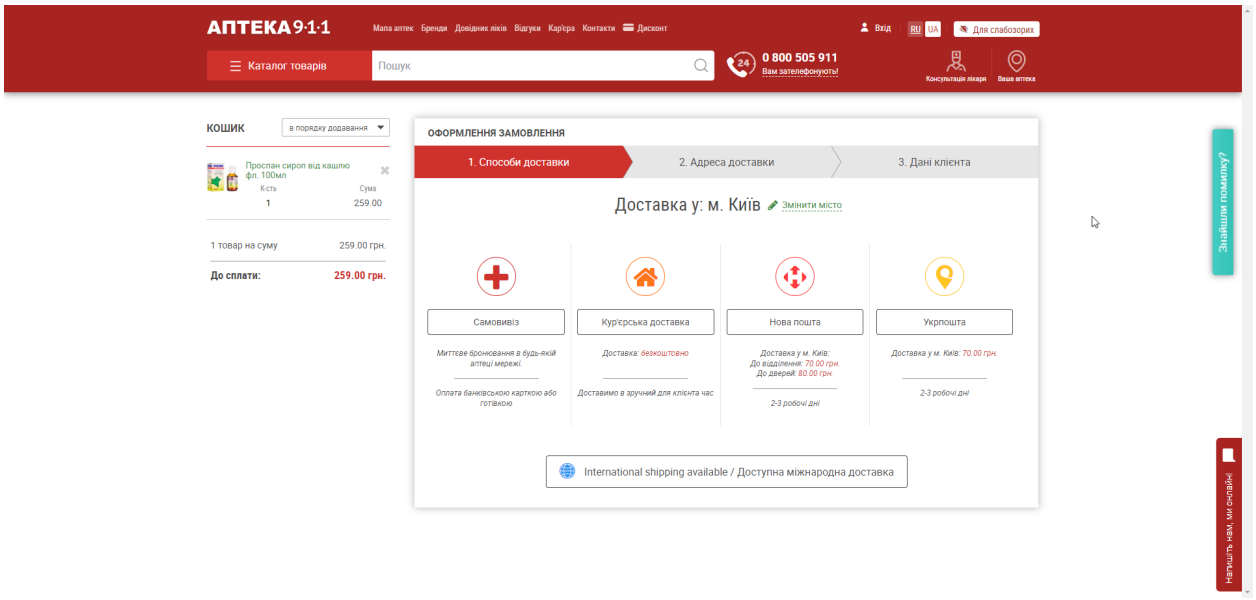


Рис. 1.3.9 Приклад сторінки вибору способу доставки для замовлення на сайті аптеки «911»

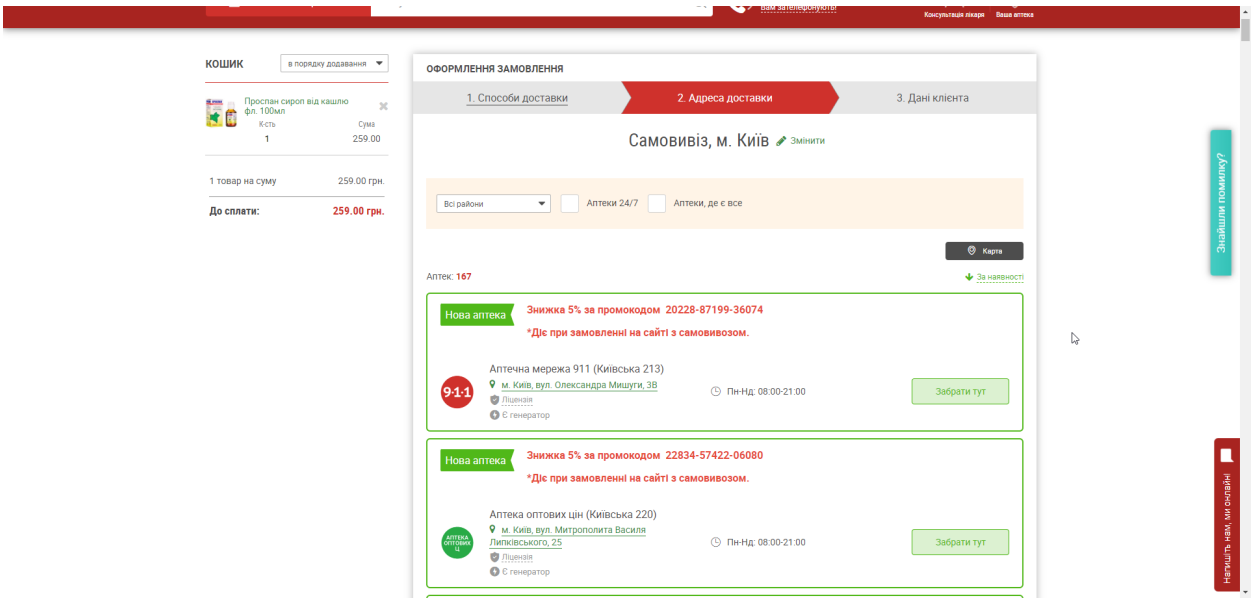


Рис. 1.3.10 Приклад сторінки вибору адреси доставки для замовлення на сайті аптеки «911»

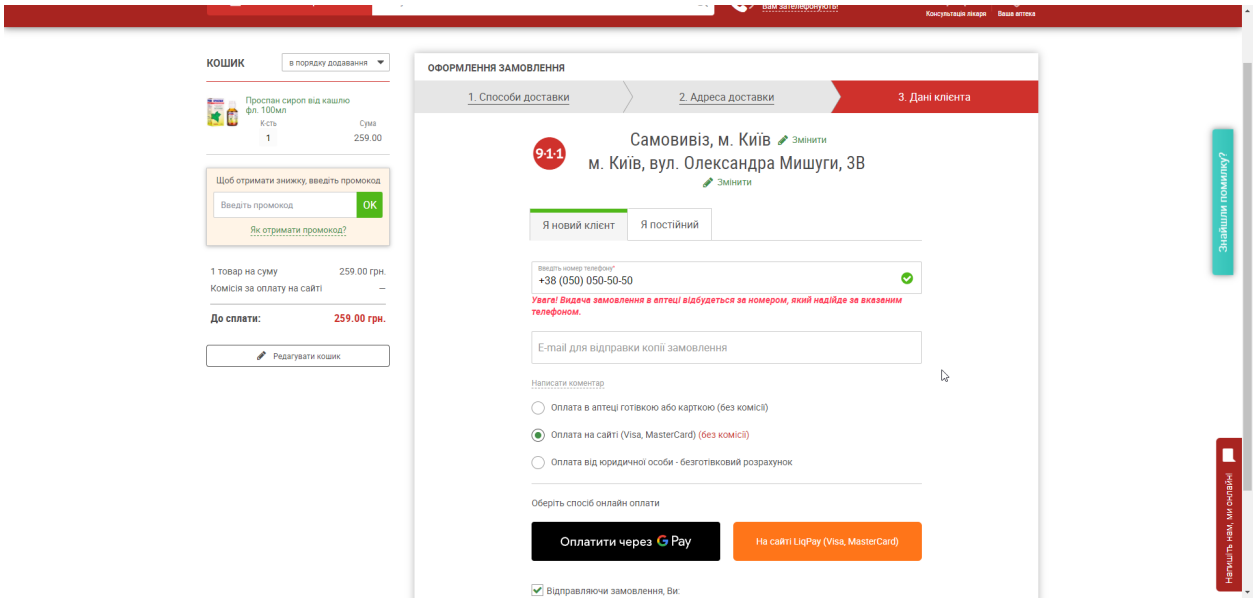


Рис. 1.3.11 Приклад сторінки вводу даних клієнта та вибору способу оплати для замовлення на сайті аптеки «911»

В залежності від вибраного способу оплати, користувачу запропонують оплатити або через сайт, використовуючи код ЄДРПОУ, або через стороні платіжні сервіси.

До переваг сайту можна віднести:

– можливість оплати прямо на сайті;
– можливість бронювання;
– опис доставки товару на сторінці товару;

Недоліки:

– важкий для сприйняття інтерфейс;
– не зручний пошук аналогів препарату;
– довгий процес замовлення, розбитий по вкладкам;
– відсутність бонусної системи;
– відсутність перевірки валідності рецептів для препаратів, які випускаються тільки по рецептам.

Ще одним прикладом великої системи для замовлення ліків є сайт аптеки «АНЦ»[8].

Головна сторінка також структурно схожа з попередніми прикладами і має

такі особливості: зліва список категорій, представлених на сайті, вгорі поле для пошуку та вкладки для переходу на профіль користувача, його замовлення, кошик та «обране». По центру знаходиться рекламний банер, під яким розташована вкладка з препаратами «Спеціальні пропозиції». А ще нижче розташована вкладка з акціями та знижками.

Як і в першому прикладі, що на сторінці зі списком препаратів, що на сторінці самого препарату пишеться кількість бонусів, яка буде отримана у випадку покупки цього препарату. Приклад головної сторінки сайту наведено на рисунку 1.3.12.

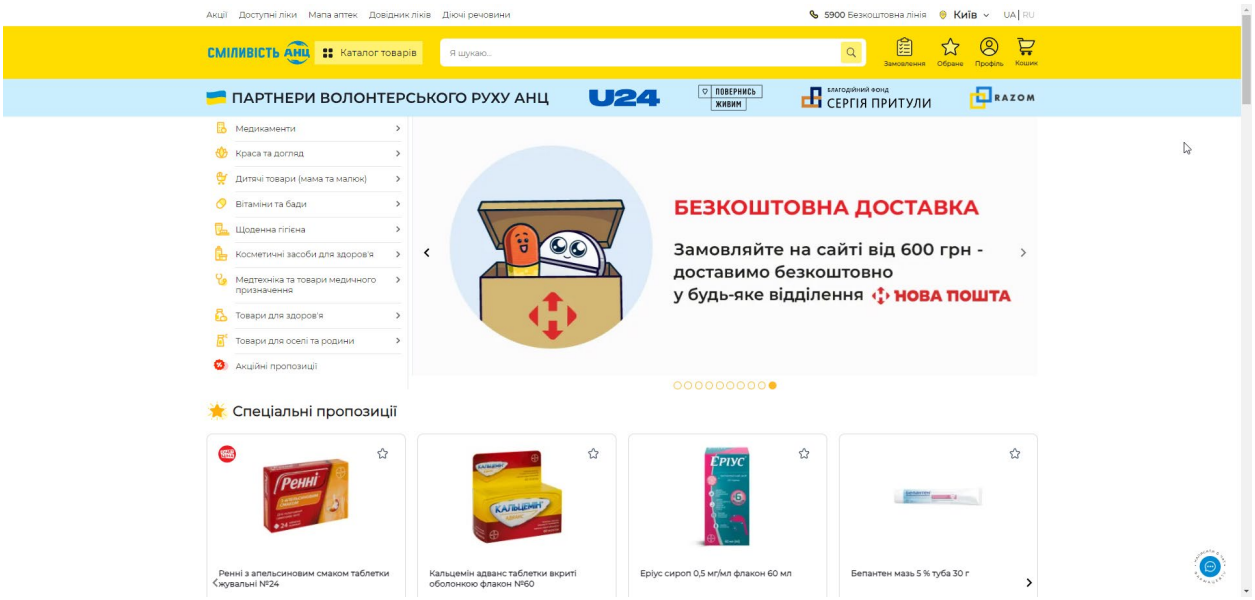


Рис. 1.3.12 Головна сторінка сайту аптеки «АНЦ»

Сторінка препарату виглядає наступним чином: зліва невелика фотографія препарату, по центру ціна та кнопка додавання до кошику. Над цими елементами список вкладок з описом товару та аналогами препарату. Нижче знаходиться способи доставки та список рекомендованих препаратів, ще нижче – опис самого препарату. Приклад сторінки препарату представлений на рисунку 3.13.

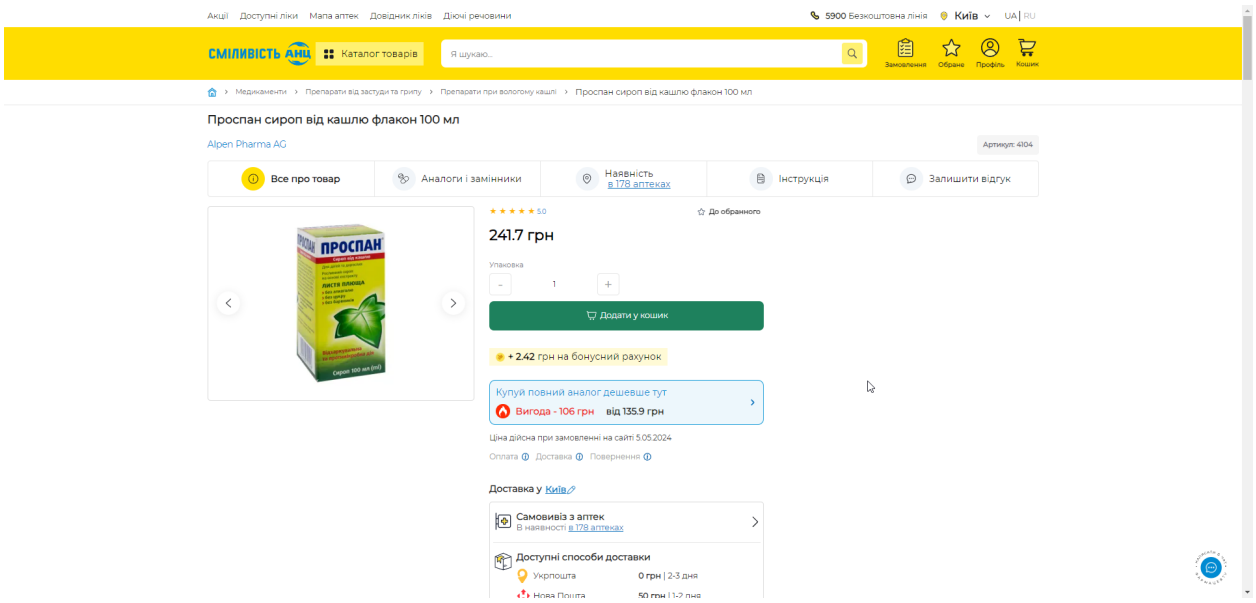


Рис. 1.3.13 Приклад сторінки препарату на сайті аптеки «АНЦ»

Для оформлення замовлення треба додати препарат в кошик за допомогою кнопки «Додати у кошик» на сторінці препарату, після чого з’являється віконце з повідомлення про успішне додання товару до кошика і пропонується перейти до оформлення замовлення, або продовжити покупки.

На сторінці кошика знаходяться всі товари, які користувач до нього додав. Після натискання кнопки «Оформлення замовлення» користувачу пропонується обрати спосіб доставки, після чого треба ввести персональну інформацію, якщо користувач не зареєстрований. Кінцевим етапом є вибір способу оплати і підтвердження замовлення.

Сторінки для оформлення замовлення на сайті наведені на рисунках 1.3.14 – 1.3.17.

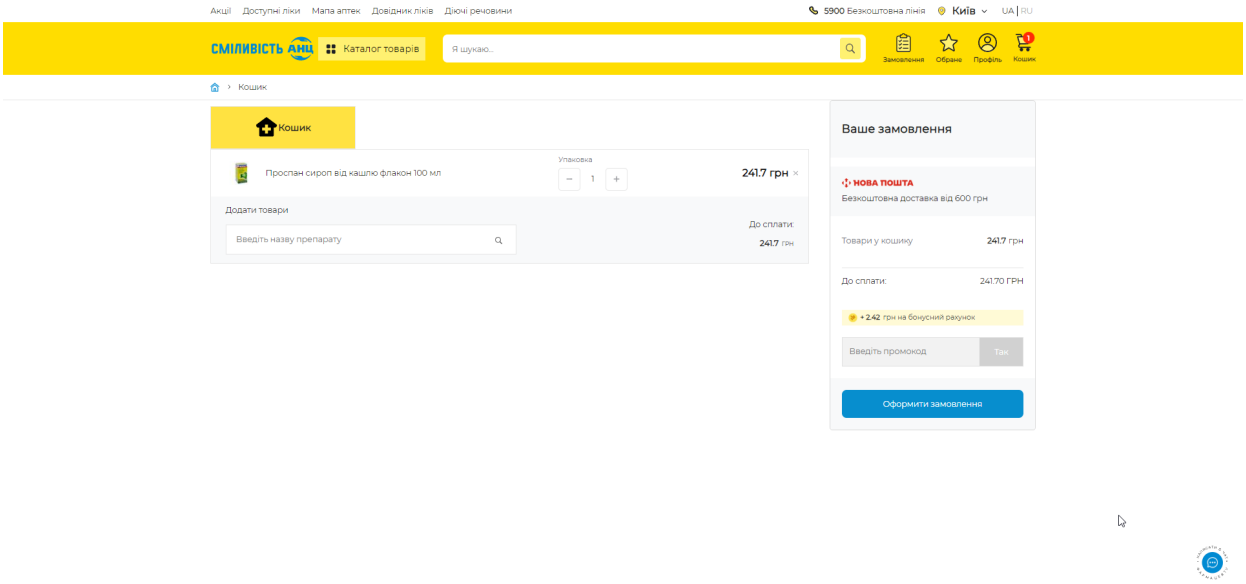


Рис. 1.3.14 Сторінка кошику на сайті аптеки «АНЦ»

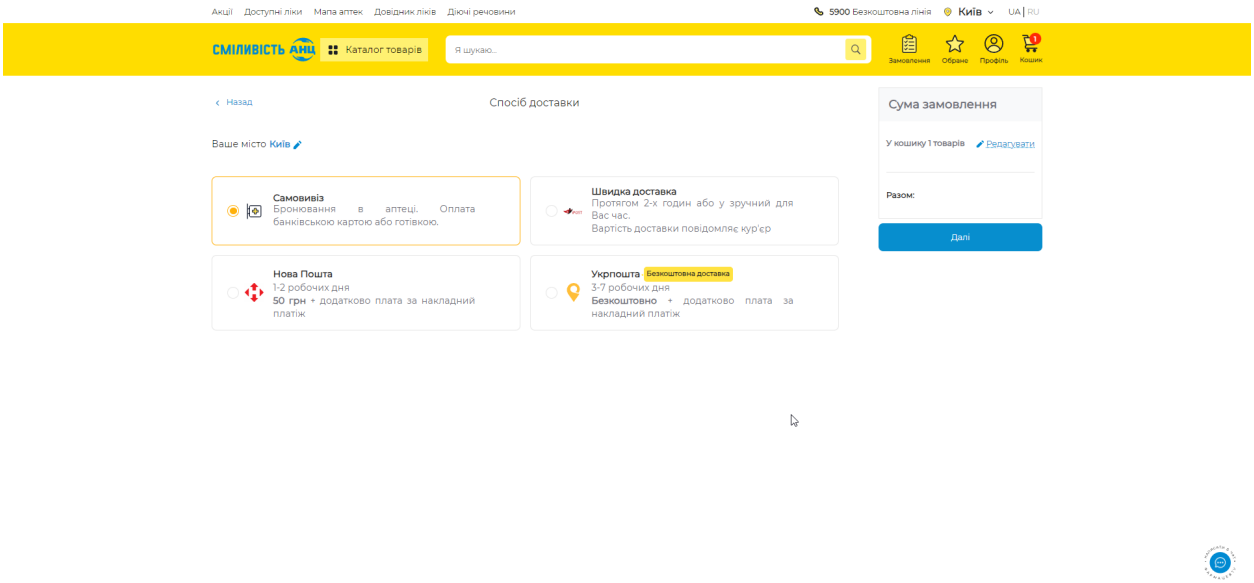


Рис. 1.3.15 Сторінка вибору способу доставки замовленя на сайті аптеки «АНЦ»

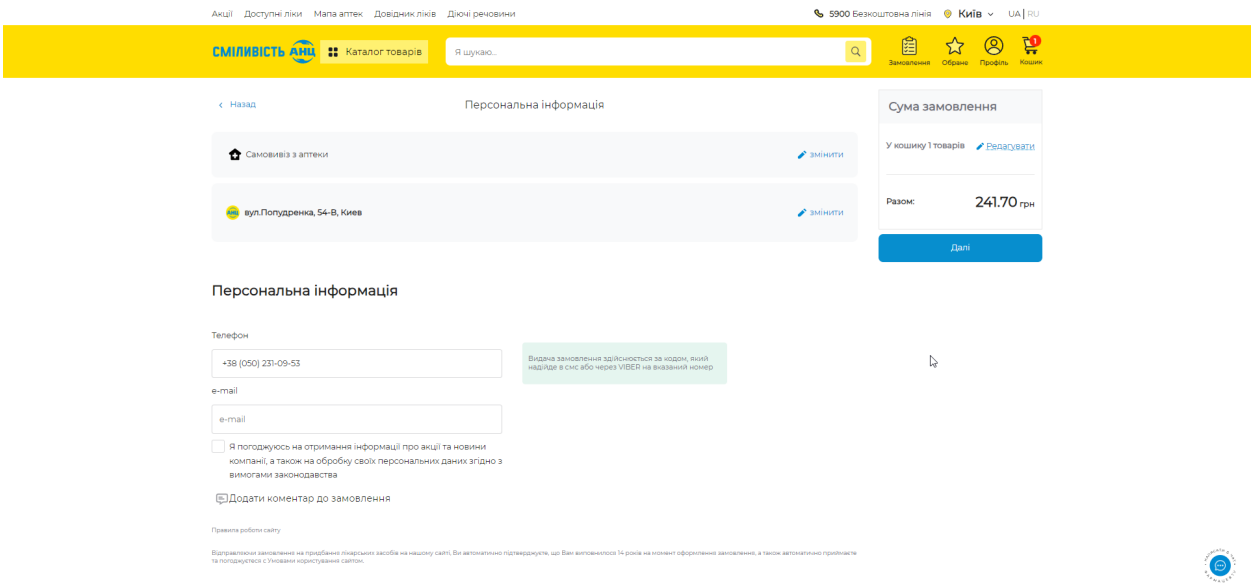


Рис. 1.3.16 Сторінка введення персональної інформації користувача на сайті аптеки «АНЦ»

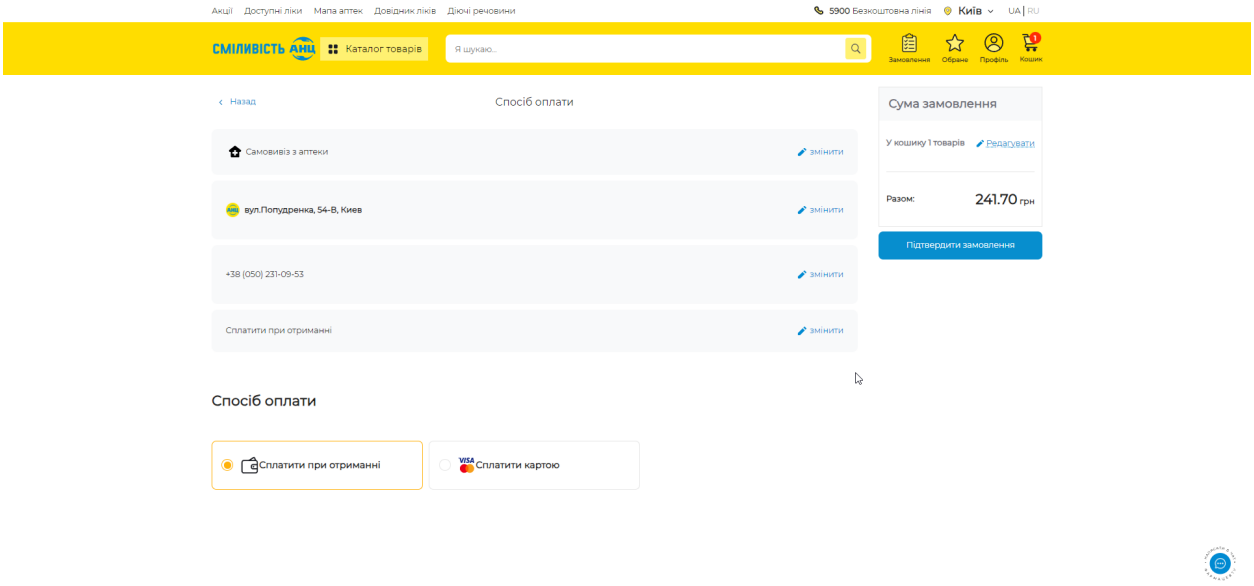


Рис. 1.3.17 Сторінка вибору способу оплати замовлення на сайті аптеки «АНЦ»

До переваг сайту можна віднести:

– легкий в розумінні та сприйнятті інтерфейс;
– можливість оплати прямо на сайті;
– можливість бронювання препаратів;
– наявність бонусної системи;
– можливість додавання товарів у «Обране»

До недоліків сайту можна віднести:

– не зручний пошук аналогів препарату;

– довгий процес замовлення, розбитий по вкладкам.

– відсутність перевірки валідності рецептів для препаратів, які випускаються тільки по рецептам.

Прикладом ресурсів для бронювання є сайт Tabletki.ua [9], і відрізняється він від сайтів аптек тим, що тут можливе лише бронювання товарів, а не покупка.

Проте є вибір між аптеками, якщо користувачу, наприклад, не підходить ціна або розташування аптеки. Звідси виходить ще один недолік, а саме відсутність бонусної програми.

Головна сторінка майже не відрізняється від аптечних сайтів, і має схожу структуру, а саме: зліва список категорій, зверху пошук товарів на сайті, та вкладки профілю користувача, його кошика та історія замовлень.

По середині знаходиться вкладка «Актуальні категорії», а під нею різні категорії на сайті . Приклад головної сторінки сайту наведено на рисунку 1.3.18.

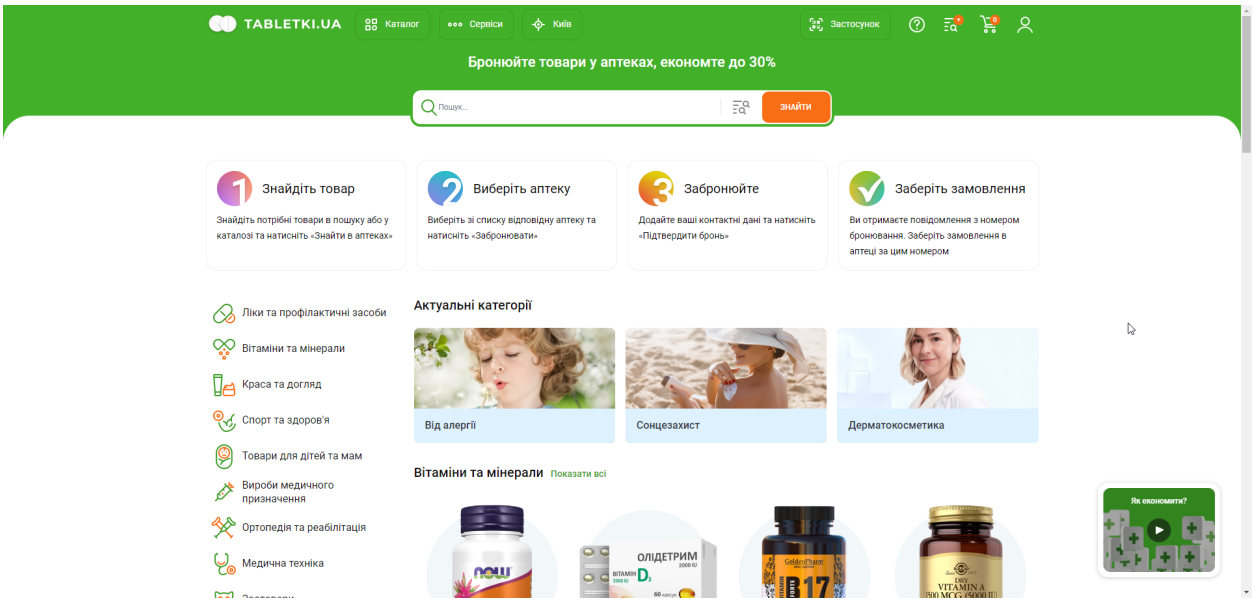


Рис. 1.3.18 Головна сторінка сайту Tabletki.ua

Сторінка товару вже структурно відрізняється від попередніх прикладів. Замість кнопки для додавання препарату в кошик, на цьому сайті знаходиться

кнопка «Знайти в аптеках». Проте інша структура дуже схожа, а саме фотографія препарату зліва сторінки, ціна по середині, список вкладок з характеристиками препарату та аналогами над ними. Нижче розташовані інструкція, склад та опис препарату. На правій частині сторінки розташовані різні рекомендації по здоров'ю. Приклад сторінки препарату на сайті наведено на рисунку 1.3.19.

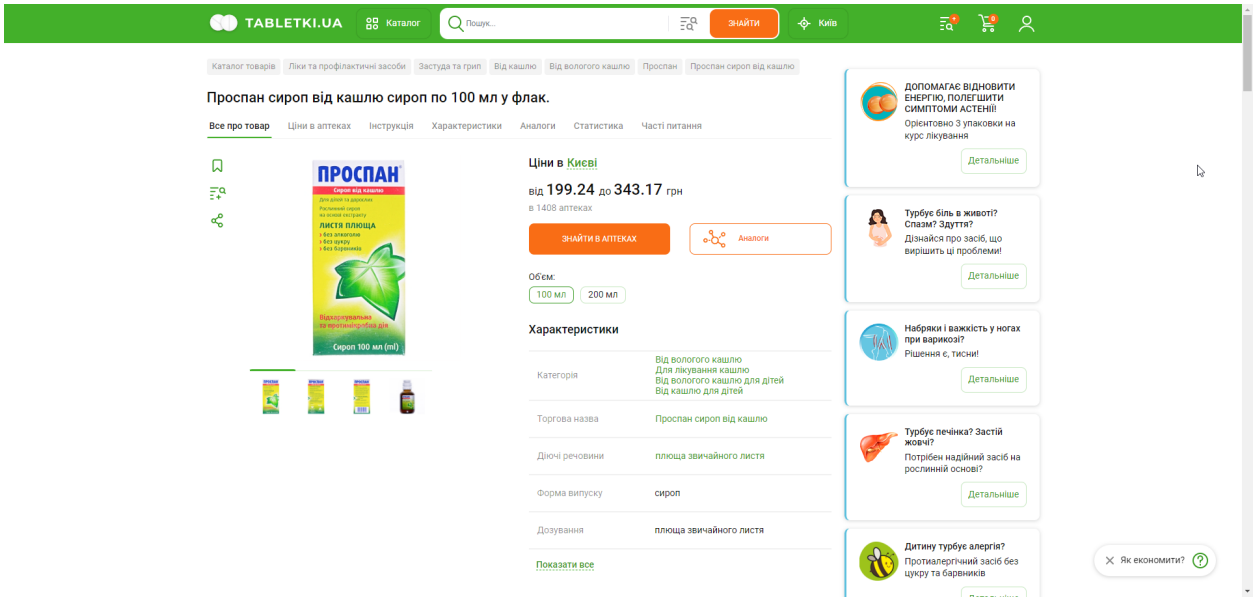


Рис 1.3.19 Сторінка препарату на сайті Tabletki,ua

При натисканні на кнопку «Знайти в аптеках» відкривається сторінка з картою та списком аптек, де є в наявності вибраний користувачем препарат.

Сторінка з картою наведено на рисунку 1.3.20.

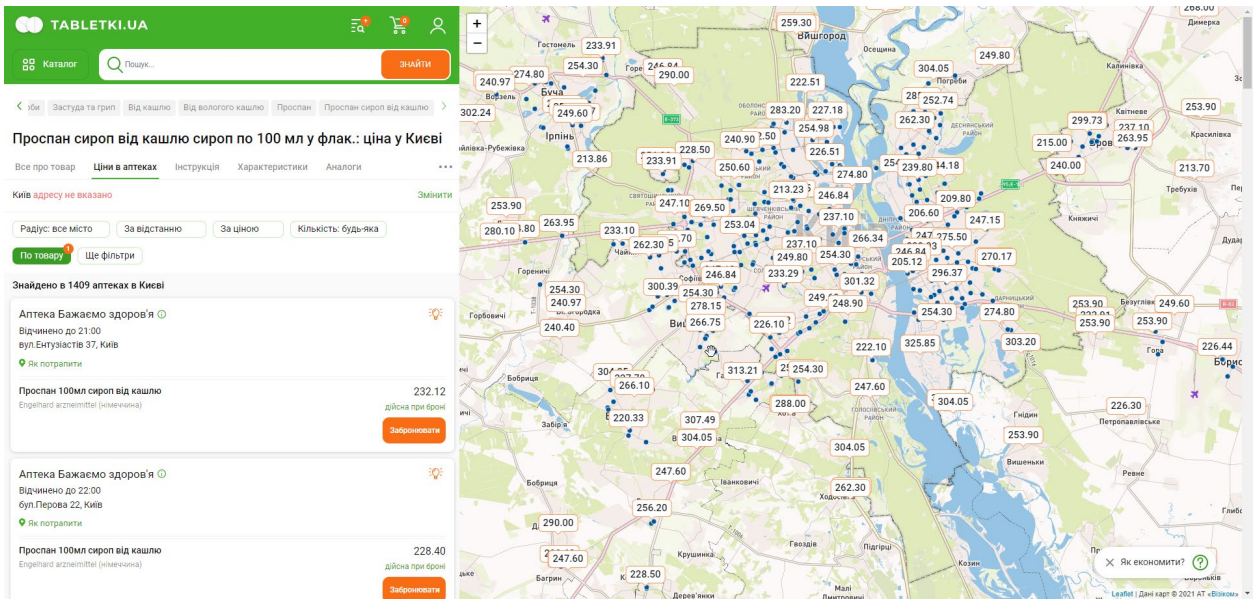


Рис. 1.3.20 Сторінка з картою та списком аптек, де є в наявності препарат

Після натискання кнопки «Забронювати» товар переходить у кошик, який виглядає вже інакше, ніж в попередніх прикладах.

В кошику просто вказаний препарат, який користувач до нього додав, є його кількість і кнопка «Оформити бронювання». Спосіб оплати та спосіб доставки вказувати не потрібно. Проте потрібно вказати номер користувача, на який прийде СМС-повідомлення з номером бронювання.

Процес замовлення препарату на сайті Tabletki.ua наведено на рисунках 1.3.21-1.3.22.

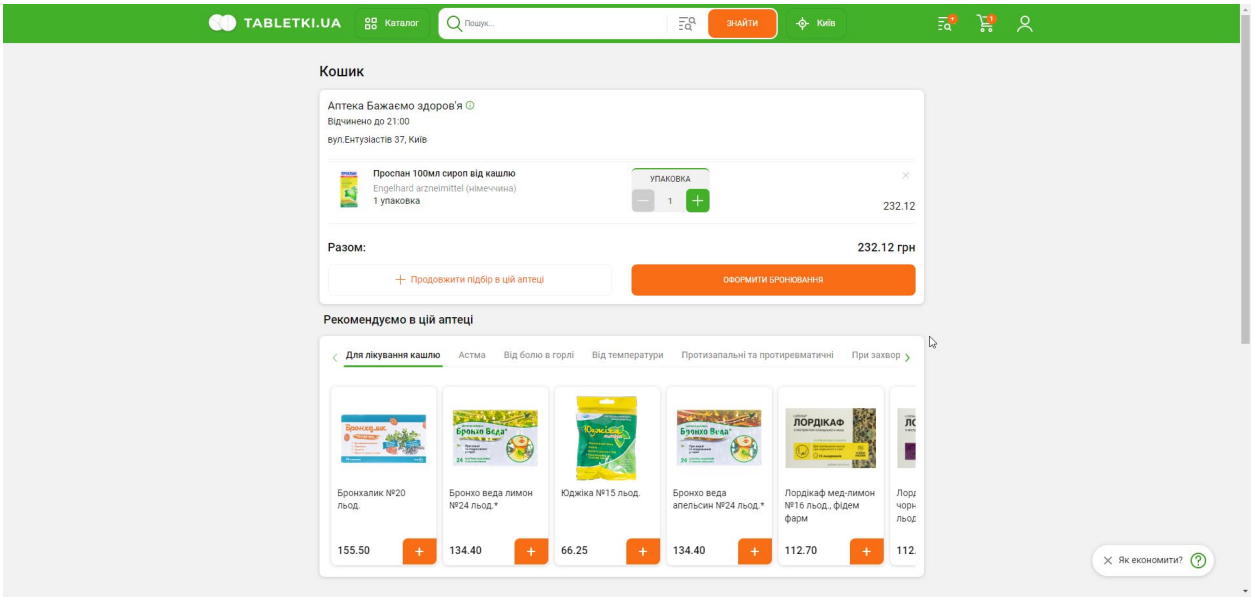


Рис 1.3.21 Сторінка кошика з товарами на сайті Tabletki.ua

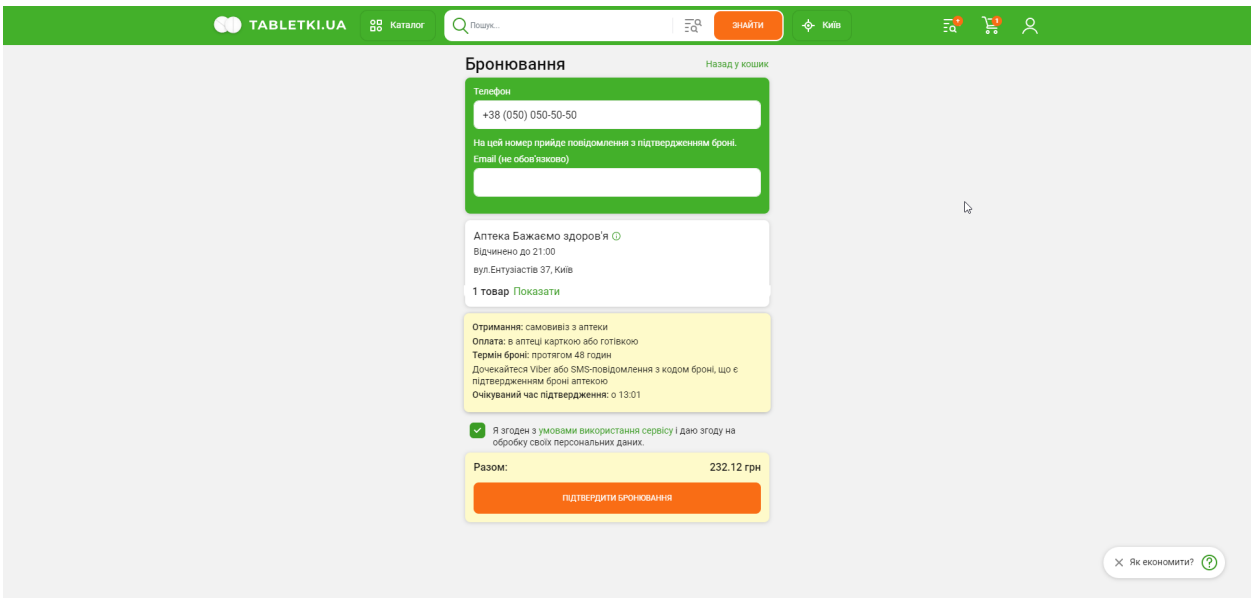


Рис 1.3.22 Сторінка бронювання товарів на сайті Tabletki.ua

До переваг сайту можна віднести:

– легкий в розумінні та сприйнятті інтерфейс;

– можливість бронювання препаратів;

– вибір між декількома аптеками;

– легкий та зрозумілий процес оформлення замовлення;

Недоліки:

– неможливість оплати замовлення одразу;

– відсутність бонусної програми;

– відсутність перевірки валідності рецептів для препаратів, які випускаються тільки по рецептам.

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.3.1.

Таблиця 1.3.1

Зведені результати аналізу характеристик додатків для замовлення ліків онлайн

Додаток / Характеристика	**Аптека “Подорожник”**	**Аптека“911”**	**Аптека “АНЦ”**	**Сервіс Tabletki.ua**
Оплата замовлення на сайті	+	+	+	-
Наявність бонусної програми	+	-	+	-
Врахування рецептурності препарату	-	-	-	-
Можливість оплати забронюваного замовлення	-	-	+	-

2 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ ВЕБ-ДОДАТКУ

2.1 Огляд REST API в якості найпоширенішого інтерфейсу для розробки веб-додатків

REST API є одним з найпоширеніших інтерфейсів для розробки веб-додатків який надає простий та ефективний спосіб взаємодії між клієнтом та сервером. [10]

Взаємодія відбувається за рахунок стандартних HTTP-методів. Найпоширенішими методами є: GET, POST, PUT та DELETE. [10]

Ось умовне представлення обміну даними через REST API для додатку “Farmakon”:

1. Запит від клієнта до сервера: коли користувач намагається отримати, наприклад, список категорій на веб-сторінці, або оновити дані про себе, веб клієнт створює HTTP-запит і відправляє його на сервер. Якщо це GET-запит, як показано на рисунку 2.1, то це простий запит на конкретний URL без будь-яких даних. Якщо ж це, наприклад, PATCH-запит, як показано на рисунку 2.2, то це запит, який надає серверу якісь певні дані, над якими (або за допомогою яких) потрібно зробити певні дії.

2. Обробка запиту на сервері: сервер отримує запит, інтерпретує його вміст в потрібний для себе формат за допомогою REST API і виконує відповідну обробку цих даних, або ж просто повертає відповідь, у випадку, якщо це GET-запит.

3. Відповідь: після опрацювання запиту, сервер створює JSON відповідь і надсилає її назад на клієнтську сторону, яка вже оброблює цей JSON файл. Відповідь може містити таку інформацію, як статус операції або оновлені дані.

4. Оновлення даних на стороні клієнта: на стороні клієнта, веб-сторінка обробляє JSON-відповідь і оновлює дані, які бачить (або не бачить) користувач. Наприклад, виводиться сторінка з категоріями (рисунок 2.1.1), або ж оновлюються дані користувача (рисунок 2.1.2).

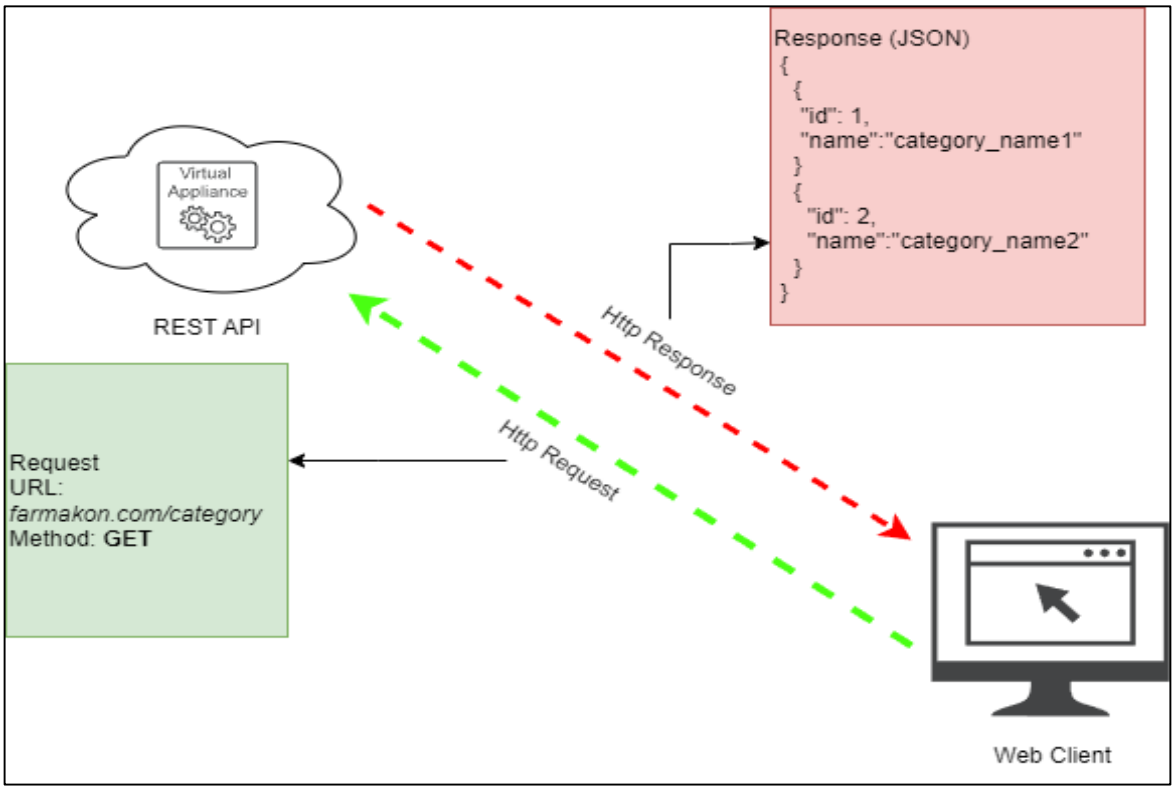


Рис. 2.1.1 Умовне представлення REST API з методом GET

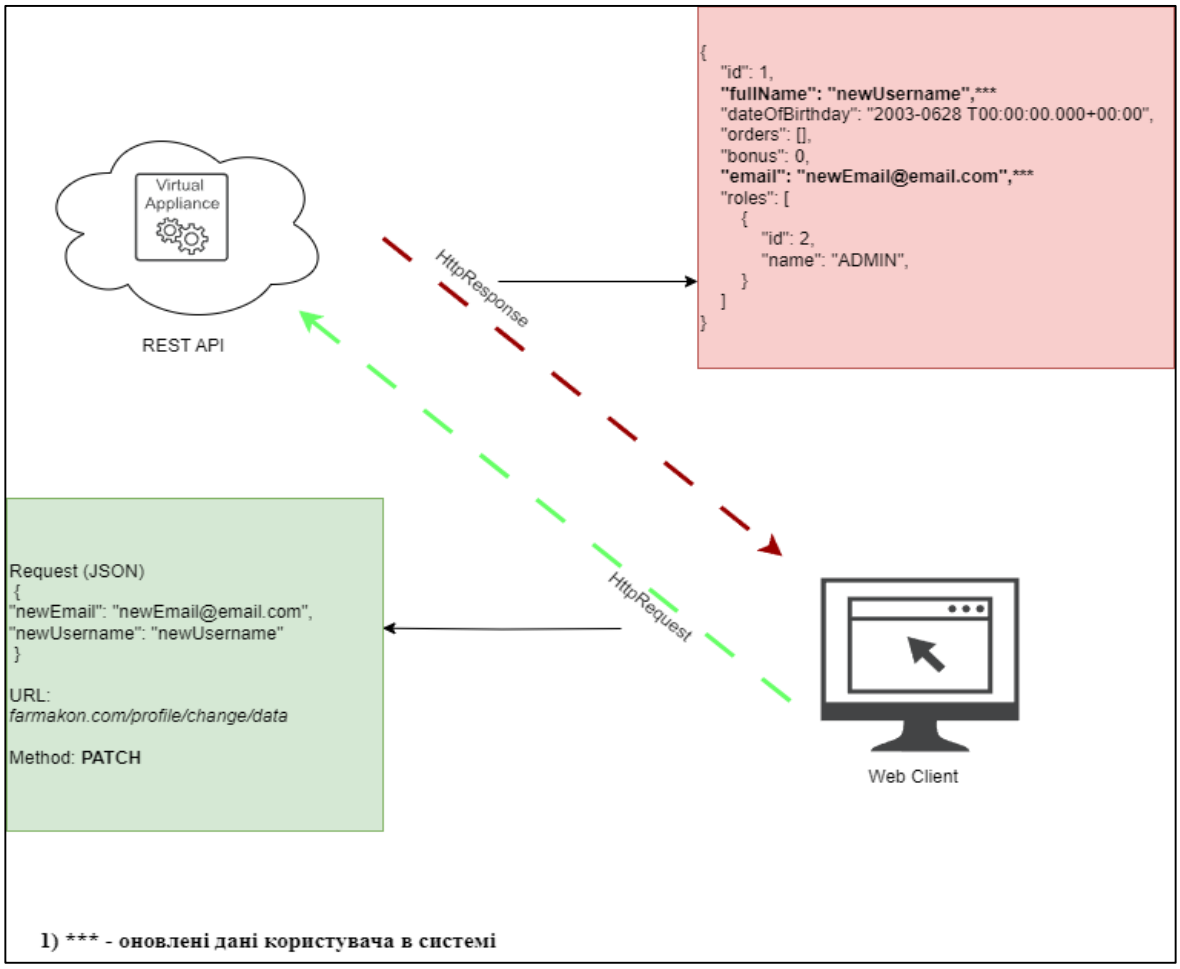


Рис. 2.1.2 Умовне представлення REST API з методом PATCH

Такий обмін даними між клієнтською частиною і серверною частиною через REST API дає змогу швидко обмінюватись даними. REST API визначає правила для створення, відправлення та обробки запитів і відповідей, стандартизує процес взаємодії та робить цей процес надійним. [10]

Цей інтерфейс має низку переваг, а саме:

- Простота і ясність: базується на простих принципах, таких як HTTP-методи і URL.
- Універсальність НТ ТР: використовується протокол HTTP, який широко підтримується на всіх сучасних платформах.
- Гнучкість форматів даних: можна використовувати різні формати, такі як JSON і XML.
- Відкритість і стандартизація: може використовуватися всіма без обмежень, сприяючи створенню стандартизованих і багаторазово використовуваних рішень.
- Підтримка кешу: підвищує продуктивність додатків і знижує навантаження на сервер.
- Простота налагодження та тестування.
- Масштабованість: дає змогу горизонтально масштабувати серверну інфраструктуру, додаючи нові сервери в міру збільшення навантаження, без зміни інтерфейсу API клієнта.
- Незалежність: клієнтська та серверна логіка є роздільними, що спрощує обслуговування та оновлення додатків.

Останній пункт є одним з основних принципів REST API, що дозволяє

розробляти та обслуговувати клієнт та сервер незалежно один від одного. Це, в свою чергу, приносить низку переваг, такі як:

- Підвищення продуктивності: код клієнта і сервера можуть бути оптимізованими незалежно один від одного.

- Підвищення безпеки: поділ клієнта і сервера дає змогу краще керувати безпекою. Сервери можуть встановлювати суворіші правила доступу до даних, а клієнти - керувати аутентифікацією та авторизацією.
- Простіше обслуговування: клієнт і сервер можуть обслуговуватися незалежно один від одного, що робить обслуговування та оновлення додатків простішим і безпечнішим.

2.2 Патерн проектування "Model-View-Controller"

MVC - архітектурний патерн проектування, який використовується під час розробки веб-додатків. Його основна ідея полягає в поділі логіки застосунку на три взаємодіючі компоненти – модель (Model), представлення (View) і контролер (Controller). [11] Кожен із цих компонентів виконує свою функцію і взаємодіє з іншими компонентами, створюючи повноцінний веб-додаток.

Модель, це компонент, що представляє дані та бізнес-логіку програми. Моделі можуть отримувати дані з баз даних та інших джерел, обробляти їх і надсилати контролерам для подальшої обробки.

Представлення використовують для відображення даних для користувача. Це може бути HTML-сторінка, JSON формат або ж будь-яке інше відображення даних. Представлення отримує дані від контролера і відображає їх у зручному для користувача вигляді.

Контролер відповідає за опрацювання запитів користувачів і взаємодію з моделями та представленнями. Контролер отримує запит від користувача, виконує необхідну дію (наприклад, отримує дані з моделі або надсилає дані в представлення) і повертає результат користувачу.

Метою цього патерну є створення гнучкої конструкції програмного забезпечення, яка спрощує подальші зміни та розширення додатку і дає змогу повторно використовувати окремі його компоненти. Використання цього патерну для великих систем полегшує їхнє розуміння та зменшує їх складність.

2.3 Трьохрівнева архітектура додатку

Трьохрівнева архітектура - підхід до розробки, при якому система поділяється на 3 рівні, а саме Controller (контролер), Service (сервіс) та Repository (репозиторій). Це забезпечує ізолювання різних аспектів програмного забезпечення один від одного, при цьому придаючи йому гнучкість та простоту в масштабуванні. Кожен рівень робить свою частину роботи не перешкоджаючи іншому рівню, але, при цьому, може використовувати його для виклику методів.

Контролер відповідає за роботу з запитами до системи, а саме за обробку HTTP-запитів та поверненню відповідей від системи, які, зазвичай, відправляються на клієнт у форматі JSON, XML або HTML. Він викликає потрібний метод в сервісі для обробки цих даних. Також, саме на цьому рівні відбувається перетворення об’єктів DTO безпосередньо в моделі, оскільки сервісам бажано працювати саме з моделями даних, а не з їх DTO для повного доступу до даних.

Сервісний рівень відповідає за бізнес логіку системи, в якій прописується весь основний функціонал програмного забезпечення. Як згадувалось вище, сервіс працює з моделями, і робиться це для того, щоб він міг працювати з базою даних через рівень репозиторіїв, оскілки в базі зберігаються саме моделі, а не їх DTO представлення.

Репозиторій на пряму працює з базою даних дістаючи та оновлюючи в ній інформацію. Саме на цьому рівні робляться CRUD операції або якісь спеціальні запити в базу даних додатку.

Трьохрівнева архітектура допомагає краще структурувати додаток для його гнучкості та чітко розділяє роботу з даними у застосунку. Схема представлення трьохрівневої архітектури зображена на рисунку 2.3.1.

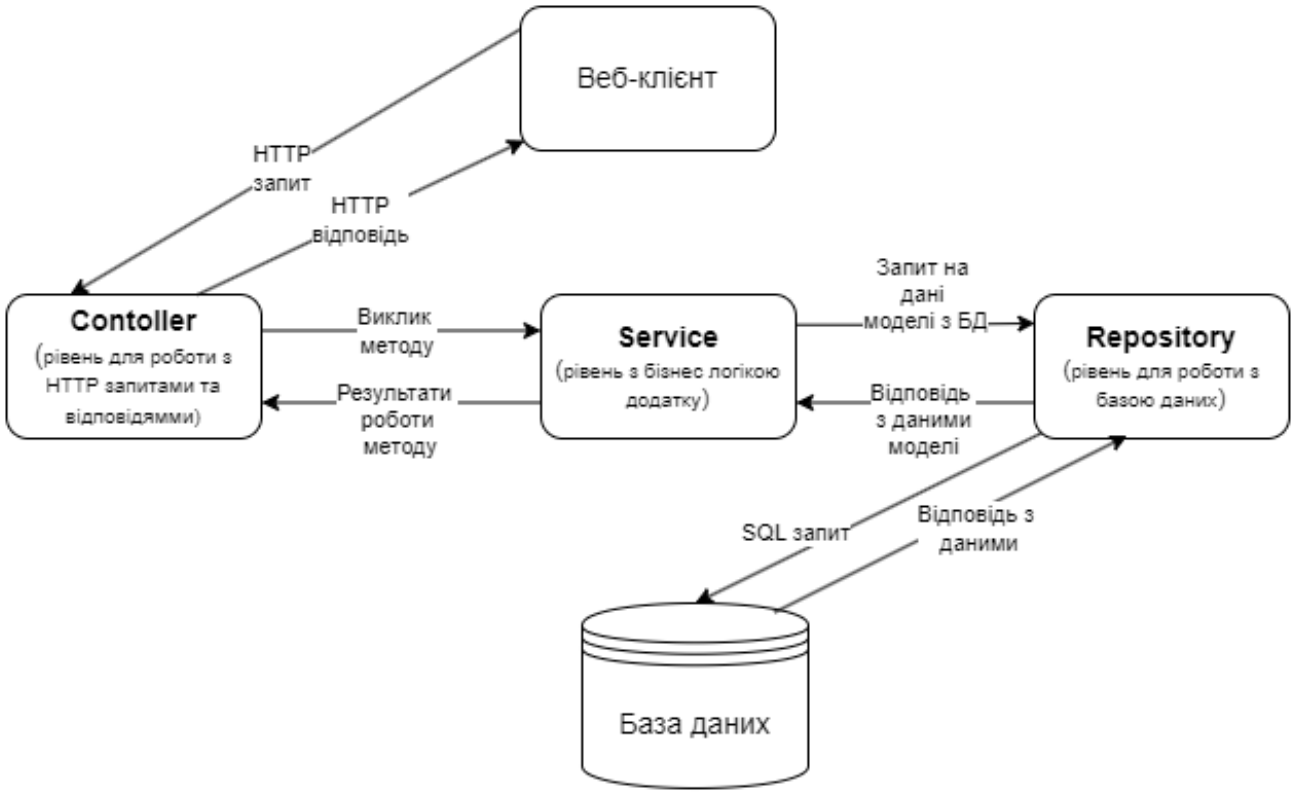


Рис. 2.3.1 Схема представлення трьохрівневої архітектури

2.4 Вибір мови програмування

"Farmakon" розробляється як веб-додаток, тобто додаток, доступний у веб-браузері. Саме тому мова програмування повинна бути заточена під веб-розробку і надавати обширний функціонал для розробки таких додатків.

Нині на ринку представлена велика кількість мов програмування, які чудово підходять для цієї задачі, проте будь-яка мова, не позбавлена недоліків, саме тому було проведено аналіз ринку.

Було проаналізовано 3 мови програмування, а саме JavaScript, C# та Java.

JavaScript - одна з найпопулярніших мов програмування для розробки серверної частини веб-додатків завдяки її широкій підтримці та екосистемі. Переваги цієї мови полягають у тому, що її використовують для розробки клієнтської частини веб-додатків, тому її легко вивчити та використовувати, а

розробникам доволі просто в ній розібратись. JS також має безліч бібліотек і фреймворків, таких як Node.js, які допомагають легко створювати додатки.

JavaScript - це високоабстрактна мова, яка дає змогу розробникам швидко розробляти програми без необхідності заглиблюватися в низькорівневі деталі. Однак у використанні JavaScript для розробки серверної частини веб-додатка є кілька недоліків.

По-перше, JavaScript - це мова з плаваючими типами даних, і якщо не правильно цим користуватись, це може призвести до непередбачуваної поведінки програми. До того ж, в цій мові немає вбудованих засобів для багатопотоковості, що може ускладнити розробку деяких типів додатків.

По-друге, у JavaScript є проблеми з безпекою, коли недоліки в коді можуть створити потенційно небезпечні вразливості.

C# - це мова програмування від компанії Microsoft, яка часто використовується для розробки серверних частин веб-додатків. Одна з головних переваг C#, це сильна типізація, яка допомагає уникнути помилок у коді.

C# також може похвалитись високим співвідношенням якості та складності коду, бо він має низку сучасних функцій, як-от властивості, делегати, лямбда-вирази та LINQ, що допомагають створювати ефективний і зрозумілий код. Це полегшує розробку, тестування та супровід веб-додатків.

Проте використання C# для розробки серверної частини веб-додатку має кілька недоліків. Навіть при тому, що .NET Core і дозволяє запускати додаток на різних платформах, він все одно не такий гнучкий, як інші мови, такі як JavaScript або Java. До того ж, у C# є складності з налаштування та розгортанням веб-додатку на сервері. Вимоги до налаштування серверів та інтеграції з іншими сервісами можуть бути вищими, ніж в інших мовах програмування.

Хоча C# має велику спільноту розробників, вона все ж таки, набагато менша, ніж спільноти інших серверних мов, таких як Java.

Говорячи про Java, варто зазначити, що мова нічим не гірше попередніх прикладів, і прекрасно підходить для робити з веб-додатками.

Вона відома своєю безпекою, надійністю та швидкістю і має вбудовані механізми для управління пам'яттю та виконання коду, що робить її дуже ефективною для веб-розробки[12].

Також Java має велику кількість потужних і простих в обслуговуванні фреймворків для розробки веб-додатків, таких як Spring Boot або Spark.

Використання цієї мови дозволяє легко масштабувати веб-додатки, що важливо для роботи з великими обсягами даних та багатьма користувачами.

Java багатоплатформена мова, що означає, що код Java може виконуватися на будь-якій платформі, яка підтримує Java Virtual Machine.[12] Це дозволяє веб-додаткам, які написані на Java, працювати на різних операційних системах без жодних змін коду. При цьому, вона відома своєю швидкістю виконання завдяки оптимізації JVM, що дозволяє створювати високопродуктивні та ефективні додатки.

Порівнюючи Java з вище наведеними аналогами, варто відзначити, що завдяки більшому набору фреймворків та бібліотек для веб-розробки, більшій кросплатформеності, підтримці відкритих стандартів, кращої безпеки та надійності, Java є кращим вибором для розробки цього додатку.

2.5 Огляд Java фреймворків для розробки веб-додатків

Java є однією з найпоширеніших мов програмування для створення веб-додатків і тому має велику спільноту розробників, які активно розробляють різні інструменти. Фреймворки та бібліотеки для веб-розробки різноманітні і можуть виконувати різні завдання, що дозволяє обирати інструменти, які найкраще відповідають заданим потребам.

При проведенні дослідження розглядалось 2 найпоширеніші та найбільш підходящі для веб-розробки фреймворки, а саме Spring Boot та Spark. Обидва фреймворки мають ряд переваг та недоліків, відрізняються в кількості доступних інструментів, складності, та швидкості роботи.

Spring Boot - це потужний фреймворк, побудований на основі фреймворку Spring для розробки веб-додатків на мові Java. Однією з головних особливостей Spring Boot є принцип "Конвенція замість конфігурації", тобто фреймворк надає стандартний набір налаштувань та конфігурацій, щоб не треба було витрачати багато часу на налаштування своїх додатків. [13] Це економить значну кількість часу та ресурсів. При створенні пустого проекту Spring Boot, одразу можна запустити сервер без великої кількості конфігурацій, що сильно підвищує простоту використання та швидкість розробки додатку.

Spring Boot має велику екосистему з різними модулями та бібліотеками для роботи з базами даних, безпекою, веб-сервісами тощо. Він надає вбудовані сервери додатків, такі як Apache Tomcat і Jetty, що дозволяють запускати додатки безпосередньо з JAR і WAR файлів. [13]

Перевагою Spring Boot є те, що велика спільнота розробників активно підтримує фреймворк і надає багато корисних матеріалів, документації та плагінів. Spring Boot підримує системи управління залежностями Maven і Gradle, які дозволяють легко додавати нові бібліотеки та модулі до проекту.

Недоліком цього фреймворку є те, що він досить важкий та ресурсозатратний. Таким чином навіть невеличкі проекти можуть досить довго запускатись. До того ж, великий об’єм різних модулів можуть зробити код програми менш зрозумілим і складним в обслуговуванні, проте цей недолік можна нівелювати якісним та чистим кодом. Однак, ці мінуси часто компенсуються швидкістю та гнучкістю розробки, які пропонує Spring Boot, за рахунок великої та добре написаної документації, великої кількістю гайдів, а також, що найголовніше, обширним функціоналом та чималою чисельністю додаткових модулів.

Spark – легкий фреймворк для розробки веб-додатків на мовах програмування Java та Kotlin. Він відомий своєю простотою та швидкістю, і є гарним варіантом для розробки невеликих веб-додатків. Великою перевагою є його підхід до розробки, який дозволяє швидко створювати веб-додатки без надмірної складності.

Spark робить процес обробки HTTP-запитів дуже простим, надаючи легкий API для обробки HTTP-запитів і відповідей. Він має вбудовану систему шаблонів

для відображення веб-сторінок, що полегшує створення динамічного контенту на стороні сервера.

Ще однією перевагою є простота встановлення та використання. Фреймворк не потребує складної конфігурації і дуже легко запускається.

Spark має чудову документацію та активну спільноту розробників, що також можна вважати його перевагою.

Проте, недоліком Spark є його не велика функціональність у порівнянні з іншими фреймворками, такими як Spring Boot. Через свій мінімалістичний підхід, Spark може не бути дуже гарним вибором для складних веб-додатків з великою кількістю функцій. При цьому, він немає великого обсягу інструментів для вирішення різних завдань.

Саме тому вибір пав саме на Spring Boot, за рахунок більшого набору функціоналу і простоті роботи з цим фунціоналом. Так, швидкість у Spark дійсно є кращою, проте більший набір інструментів та можливостей нівелює цей недолік.

2.6 Spring Boot

Spring Boot - це фреймворк для розробки Java-додатків, який робить створення веб-додатків і мікросервісів легким та простим. [13] Однією з головних переваг Spring Boot є його функція автоконфігурації, яка дозволяє швидко створювати додатки без великої кількості налаштувань.

Фреймворк базується на Spring Framework, який забезпечує більш високорівневий підхід до розробки та пропонує багато готових компонентів і рішень для управління залежностями, створенням конфігурацій та обробки виключень. [13] Spring Boot включає в себе ряд функцій, які дозволяють швидко створювати прототипи та запускати додатки без додаткових зусиль.

Ще однією важливою особливістю Spring Boot є вбудований контейнер для запуску додатків. [13] Це дозволяє створювати і тестувати додатки локально, без необхідності розгортати їх на сервері. Він також надає можливість легко змінювати

налаштування програми без необхідності перекомпіляції або перезапуску програми.

Одним з основних принципів Spring Boot є "конвенція перед конфігурацією", яка мінімізує час, який буде витрачено на налаштування та конфігурації додатку. Він надає великий вибір технологій та бібліотек для веб-розробки, підтримуючи різні шаблони проектування та архітектурні підходи, такі як MVC або REST. [13]

Однією з найбільших переваг Spring Boot є його широка екосистема. Існує безліч бібліотек, фреймворків та розширень, які можуть розширити функціональність програмного забезпечення в залежності від завдання. Існує велика спільнота розробників, які активно підтримують фреймворки, дають поради та допомагають вирішувати проблеми.

Однією з найважливіших особливостей Spring Boot є те, що його можна легко і швидко інтегрувати з іншими технологіями та сервісами. Він підтримує широкий спектр механізмів, таких як REST-сервіси, мікросервісна архітектура, бази даних, аутентифікація та авторизація. Це дозволяє легко будувати складні системи та додатки.

Загалом, Spring Boot - це потужний інструмент для розробки веб-додатків, який дозволяє швидко створювати стабільні, високопродуктивні додатки з мінімальними зусиллями. Його гнучкість, велика екосистема та простота використання роблять його одним з найкращих варіантів для сучасної веб-розробки.

2.7 Вибір СУБД

СУБД - це набір програмних засобів, що використовуються для створення, зберігання, організації, управління та доступу до баз даних. Вони забезпечують зручний та ефективний доступ до даних, гарантують їх цілісність та безпеку.

Перші СУБД з'явилися в 1960-х , а саме реляційні СУБД з'явилися в 1970-х роках і зберігали дані в табличних структурах і стали популярними завдяки своїй простоті та ефективності.

У 1980-х роках почали розроблятися об'єктно-орієнтовані СУБД, які могли зберігати об'єкти разом з їхніми зв'язками. Це було дуже корисно для складних даних, які можна представити у вигляді об'єктів.

З появою інтернету з'явилися хмарні СУБД, а доступ до баз даних став можливим через інтернет. Тому стало можливим зберігати і обробляти без використання апаратного забезпечення великі обсяги даних.

Одним з найважливіших кроків у розвитку СУБД був винахід мови структурованих запитів SQL, яка стала стандартом для роботи з реляційними базами даних.

Таким чином вибір був дуже складним, оскільки на ринку представлена величезна кількість готових і якісних рішень, кожне з яких має свої переваги та недоліки. Тому було проаналізовано декілька СУБД, а саме MySQL, PostrgeSQL та MongoDB.

MySQL - одна з найпопулярніших реляційних систем управління базами даних з відкритим кодом. Основною перевагою є швидкість та ефективність, особливо при оптимізації запитів. MySQL добре підходить для великих обсягів даних та високонавантажених систем і має широку підтримку.

Однак MySQL має деякі недоліки. Наприклад, порівняно з іншими реляційними базами даних, вона може бути менш надійною при виконанні деяких операцій, таких як зміна схеми бази даних. До того ж, безпекова компонента MySQL є не з найкращих.

PostgreSQL - ще одна реляційна база даних з відкритим вихідним кодом, відома своєю SQL стандартизацією та розширеними можливостями. PostgreSQL має багато функцій, яких немає в інших реляційних базах даних. Наприклад, таблиці без блокувань, робота з JSON та багато інших. [14] Вона має потужну систему безпеки та підтримку транзакцій, що робить її дуже потужним інструментом. [14] Однак PostgreSQL є більш ресурсомісткою, ніж MySQL, особливо при роботі з великими обсягами даних. Цю СУБД складніше конфігурувати і налаштовувати, ніж інші системи, проте ці недоліки можна компенсувати правильною експлуатацією та правильними налаштування БД.

MongoDB - це документно-орієнтована база даних, яка відрізняється від реляційних баз даних тим, що зберігає дані у вигляді документів у JSON форматі. MongoDB належить до класу NoSQL систем управління базами даних і працює з документами, а не з таблицями і є крос-платформеною системою. Її особливості дозволяють використовувати СУБД для різних завдань, надаючи максимальну продуктивність і надійність.

За рахунок використання документів, а не таблиць, MongoDB є дуже швидкою системою. Вона також легко масштабується, що робить її придатною для великих обсягів даних і високонавантажених систем. Однак, вона має і недоліки. Наприклад, ця система може бути менш ефективною, ніж реляційні бази даних, для складних операцій обміну даними, таких як злиття. І так як вона зберігає дані у вигляді JSON-документів, це може вимагати більше місця для зберігання.

Таким чином, проаналізувавши ринок, було вибрано PostgreSQL, як надійний та швидкий інструмент, який, при правильному налаштуванні та використанні, буде надавати величезний спектр функціоналу, швидко працювати та безпечно зберігати всі данні додатку, не потребуючи, при цьому, складних та великих обсягів конфігурації. Саме через об’ємний функціонал, гарну оптимізацію та безпеку PostgreSQL є чудовим вибором.

2.8 Огляд Hibernate як основної бібліотеки для роботи з базою даних

У Java для взаємодії з базами даних використовується API JDBC, що дозволяє Java-програмам працювати з різними базами даних через єдиний набір методів. Перевага використання JDBC від інших підходів для інших мов програмування, таких як PHP, полягає в тому, що в Java використовується єдиний інтерфейс для роботи з різними реляційними базами даних. При використанні JDBC програма на Java не взаємодіє безпосередньо з таблицями бази даних, а використовує спеціальні драйвери, що і робить цей API дуже універсальним. Ці драйвери підключаються і працюють з базою даних, і завдяки такому підходу будь-яка програма на Java може

працювати з різними БД, використовуючи ті самі методи без потреби переписувати код.

Перевага JDBC полягає у уніфікації методів для роботи з різними базами даних. Використання єдиного інтерфейсу дозволяє спростити розробку і зробити код більш зрозумілим. JDBC-драйвери підвантажуються динамічно під час виконання програми, що дозволяє автоматично ініціалізувати та викликати їх, коли програма взаємодіє з базою даних через певну URL-адресу.

Хоча JDBC і є зручним для взаємодії з різними базами даних, проте він є досить складним і може вимагати багато коду для певних операцій. Наприклад, робота з ORM вимагає ручного відображення об'єктів Java в таблиці бази даних і навпаки. Саме тут на допомогу приходить Hibernate.

Hibernate спрощує роботу з базами даних і надає високорівневий API для виконання операцій з ними. Він автоматично вирішує багато завдань, які зазвичай вимагають великих обсягів коду при використанні JDBC. Наприклад, Hibernate робить роботу з ORM простішою та інтуїтивно зрозумілішою, використовуючи анотації для зіставлення об'єктів з таблицями бази даних. [15] Це значно економить ресурси та час.

Hibernate дозволяє виконувати запити до бази даних за допомогою мови запитів HQL, яка схожа на SQL, але використовує об'єктну модель даних, тому є можливість виконувати складні операції над об'єктами без прив'язки до конкретної бази даних. [15] Завдяки своїм можливостям і простоті використання, Hibernate став популярним вибором для роботи з базами даних в Java-проектах, за допомогою якого можна швидко реалізувати функції для маніпулювання даними без написання складного JDBC-коду. Це і є головною перевагою над JDBC, а саме відсутність великого об’єму коду для конфігурації, налаштувань та додаткових, одноманітних для всіх проектів, речей.

Таким чином, при виборі рішення, яке допоможе ефективно працювати з всіма даними додатку без зайвих затрат часу та ресурсів, бібліотека Hibernate, найпопулярніша бібліотека для роботи з БД у Java, є прекрасним вибором.

3 ПРОЕКТУВАННЯ, РОЗРОБКА ТА ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ДОДАТКУ

3.1 Проектування додатку за допомогою UML засобів

Під час проектування програми використання інструментів UML, таких як діаграми класів, діаграми взаємодії, діаграми станів та інших, відіграє важливу роль. UML надає чітке та структуроване представлення зв’язків між різними частинами системи. Це дозволяє краще розуміти потреби та вимоги системи, що розробляється, і створює основу для подальшого розвитку програмного забезпечення.

Правильне використання UML дозволяє аналізувати та проектувати програмні системи на ранніх стадіях розробки, що дозволяє виявити й усунути потенційні проблеми ще до початку розробки. Це допомагає зменшити час, необхідний для вирішення проблем, пізніше в процесі розробки.

UML засоби допомагають створювати більш якісне та більш стабільне програмне забезпечення. Завдяки чіткому представленню взаємозв'язків і структур системи можна одразу легко виявляти та усувати дефекти в архітектурі та дизайні програми, що дає більшу надійність і ефективність застосунку. До того ж, UML допомагає аналізувати ризики та прогнозувати проблеми, які можуть виникнути під час розробки програмного забезпечення.

Підсумовуючи, використання інструментів UML під час розробки програми є важливим і необхідним, оскільки це сприяє розробці високоякісного програмного забезпечення та покращує комунікацію між учасниками проекту.

3.1.1 Діаграма використання

Діаграма варіантів використання — одна з основних діаграм в UML, що показує функціональність системи та взаємодію між акторами (користувачами) і

системою. Ця діаграма допомагає візуалізувати основні функції системи з точки зору користувача. Це дозволяє аналізувати завдання, які можуть виконувати користувачі системи, і визначати, як ці завдання впливають на саму систему.

Ця діаграма допомагає визначити основні функції системи з точки зору користувача. Це дозволяє краще зрозуміти вимоги до функціональності система та як вона використовуватиметься. Вона допомагає уникнути непорозумінь між розробниками та клієнтами проекту щодо функціональних вимог системи. Графічне представлення взаємодії між користувачем і системою дозволяє краще зрозуміти, як працює система і які можливості система надасть користувачеві. До того ж, саме ця діаграма може служити основою для подальшого аналізу та проектування системи.

Говорячи про додаток «Farmakon», діаграма використання представлена на рисунку 3.1.1. Вона має два актори, а саме «Користувач» та «Адміністратор».

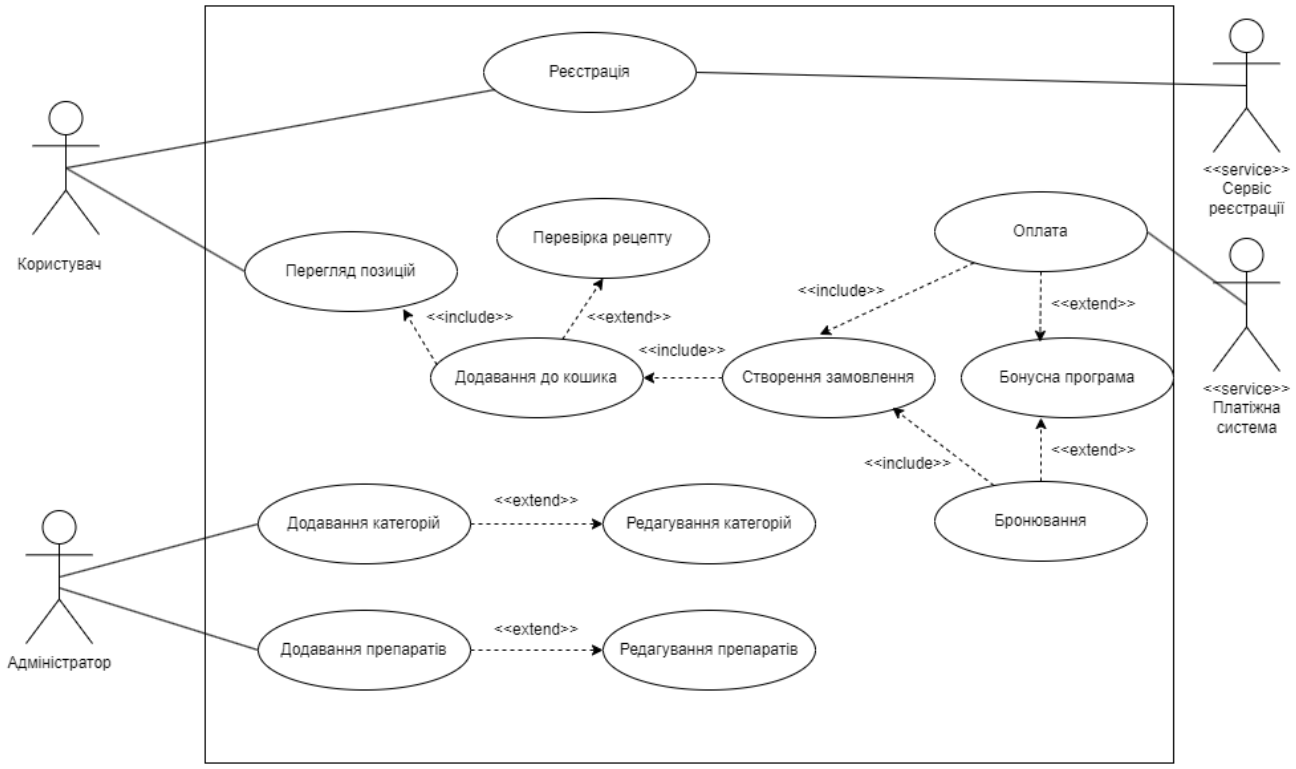


Рис 3.1.1 Діаграма використання додатку «Farmakon»

Таким чином, можна вже виявити деякі вимоги до додатку, а саме:

Для користувача:

1. Користувач повинен мати можливість зареєструватись;
2. Користувач повинен мати можливість переглядати всі наявні позиції;
3. Користувач повинен мати можливість додавати ці позиції до кошика;
4. Користувач повинен мати можливість створити замовлення на основі доданих до кошика позицій;
5. Користувач повинен мати можливість ввести номер рецепту при додаванні лікарського засобу, який випускається за рецептом;
6. Користувач повинен мати можливість оплатити замовлення;
7. Користувач повинен мати можливість зробити бронювання замовлення;
8. Користувач повинен мати доступ до бонусної програми;

Для Адміністратора:

1. Адміністратор повинен мати можливість додавати категорії;
2. Адміністратор повинен мати можливість редагувати категорії;
3. Адміністратор повинен мати можливість додавати препарати;
4. Адміністратор повинен мати можливість редагувати препарати;

Таким чином, за допомогою діаграми використання було виведено певні вимоги для серверної частини додатку «Farmakon» для користувачів та адміністраторів сервісу.

3.1.2 Діаграма класів

Діаграма класів є однією з основних діаграм в UML, що показує структуру системи, класи, властивості, методи та зв'язки між ними. Ця діаграма допомагає візуалізувати структуру програмного забезпечення та зв’язки між його компонентами, що робить її корисною під час проектування програмних систем.

Однією з переваг використання діаграм класів є можливість аналізувати та проектувати структуру програми на ранніх стадіях розробки. Діаграми класів

дозволяють визначити основні класи, властивості та методи програми, допомагаючи краще зрозуміти, які класи слід створити та як вони взаємодіють один з одним. Графічне відображення класів та їхніх зв'язків полегшує обмін інформацією про структуру програми та дозволяє уникнути непорозумінь між розробниками. Крім того, діаграми класів можна використовувати як основу для подальшого проектування програмного забезпечення.

Говорячи про додаток «Farmakon», діаграму класі представлено на рисунку 3.1.2.

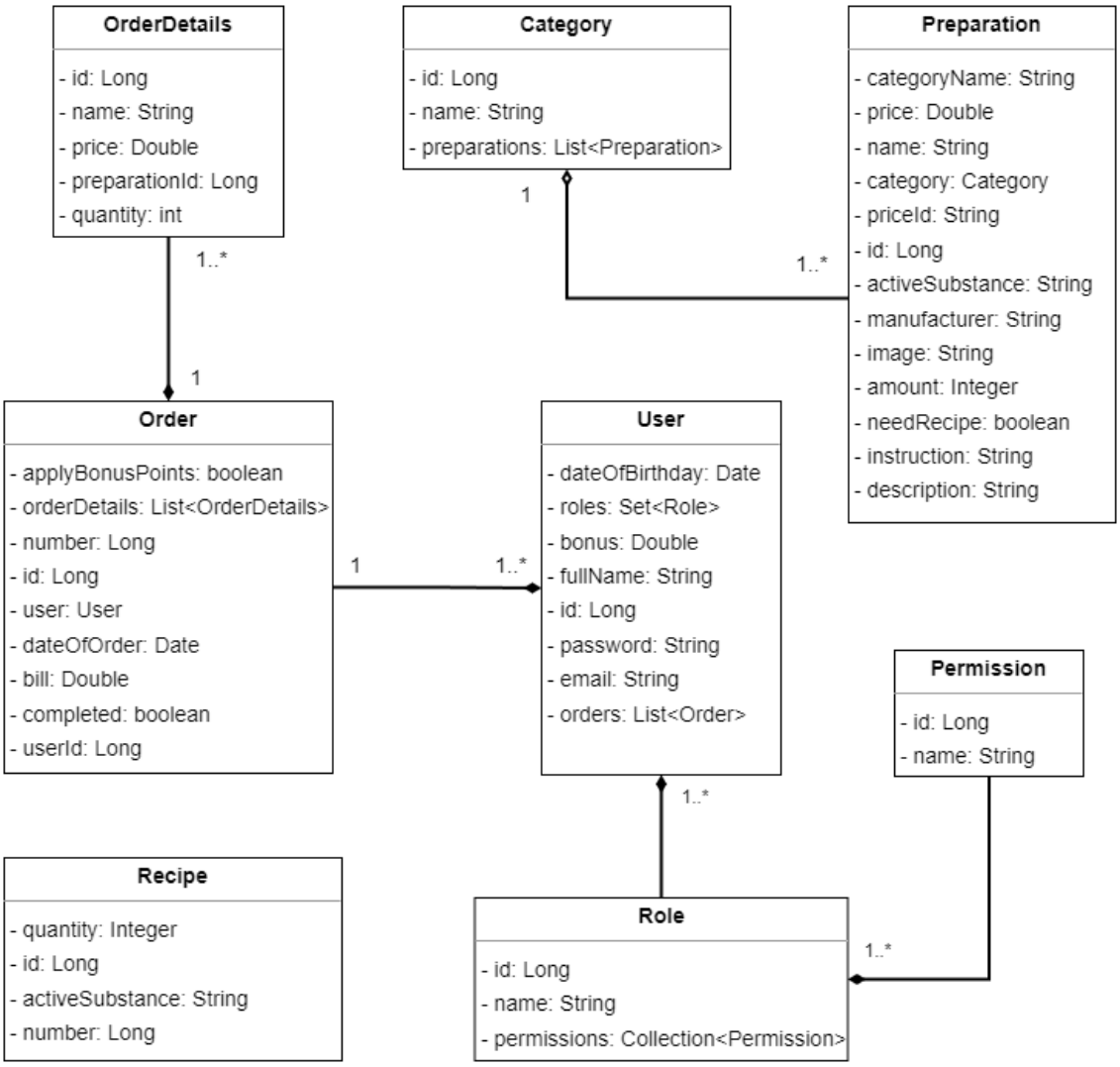


Рис 3.1.2 Діаграма класів для додатку «Farmakon»

З цієї діаграми можна побачити основні класи для, так званих, моделей додатку та їх відношення між собою. Наприклад, модель Order зв’язується з моделлю OrderDetails зв’язком «Один до багатьох», оскільки одне замовлення може мати в собі багато деталей замовлення.

3.1.3 Діаграма бази даних

Діаграма бази даних є важливим інструментом у проектуванні бази даних. Вона відображає структуру бази даних, включаючи таблиці, поля, зв’язки між таблицями та інші елементи. Ця діаграма допомагає краще зрозуміти структуру бази даних і зв’язки між її компонентами. Однією з головних переваг використання діаграми бази даних є можливість моделювати та аналізувати структуру бази даних на ранніх етапах процесу розробки.

Діаграми дозволяють визначити основні таблиці, поля і зв'язки між ними, сприяють кращому розумінню структури даних і дозволяють уникнути проблем при подальшій роботі з базою даних, надаючи основні дані про базу даних та її структуру.

Діаграма класів для додатку «Farmakon» представлена на рисунку 3.1.3.

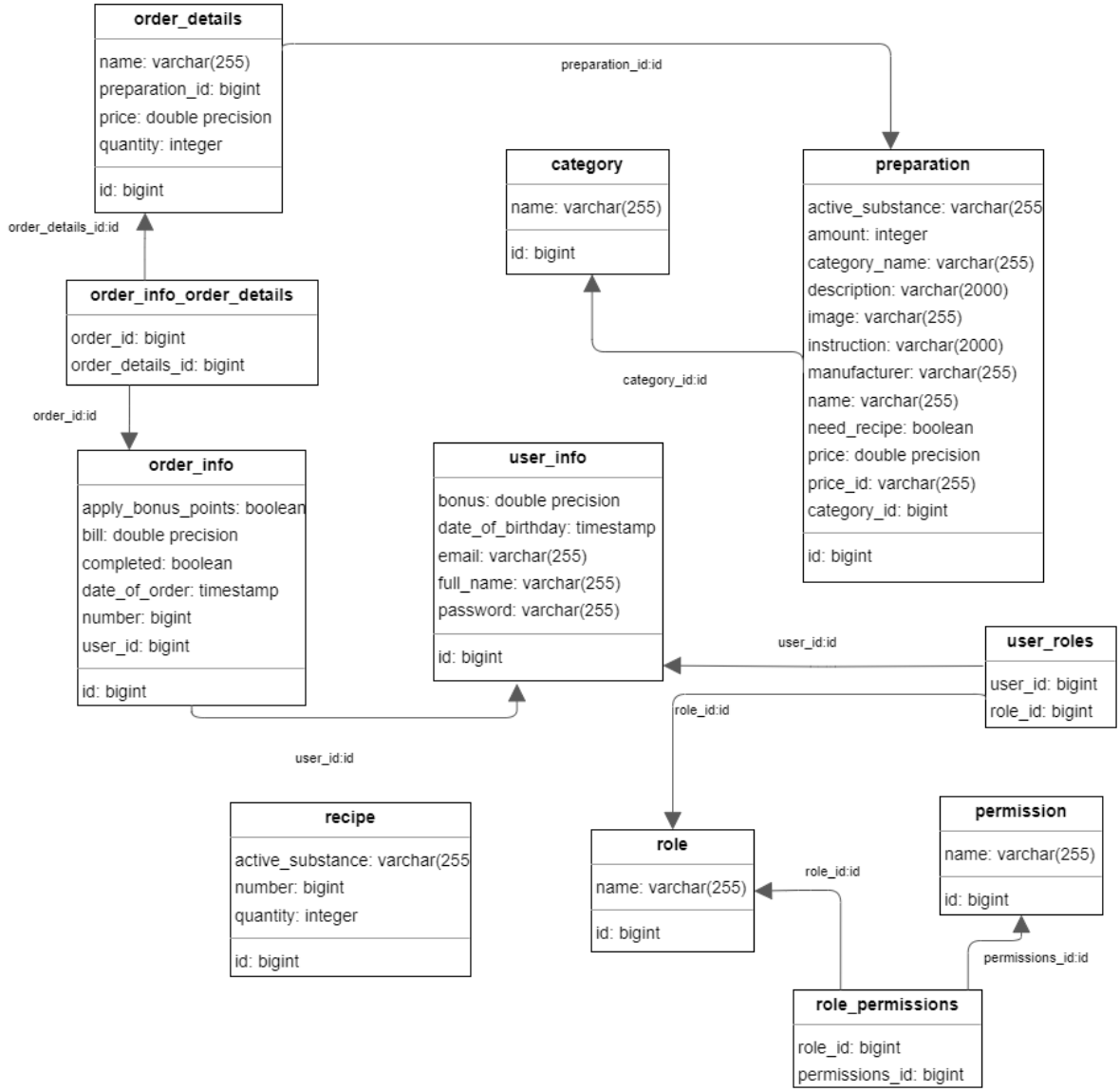


Рис. 3.1.3 Діаграма бази даних для додатку «Farmakon»

3.2 Розробка основного функціоналу додатку

Додаток розробляється на основі трьохрівневої архітектури, яка в свою чергу ділить структуру на 3 рівні: контролери, сервіси, та репозиторії.

Контролери – класи, які використовуються для обробки HTTP-запитів до додатку. Вони можуть приймати тіло запиту, в якому містяться певні дані, над якими, або з допомогою яких, потрібно зробити певні дії. Або ж можуть і не мати тіла запиту. Основна задача таких класів – обробляти запити, перетворювати тіла запитів з DTO, які приходять в різних форматах (в додатку «Farmakon»

використовується JSON), в класи-моделі, викликати потрібні методи в сервісах і передавати в них ці моделі. Важливо зазначити, що в контролерах не може бути будь-якої бізнес-логіки, вони просто слугують буфером між веб-стороною та сервісами.

Сервіси – класи, в яких відбувається вся бізнес логіка додатку. Ці класи працюють безпосередньо з моделями, змінюють дані в базі даних та виконують всі бізнес задачі проекту. Для доступу до бази даних сервіси використовують наступний, третій рівень додатку – репозиторії.

Репозиторії - це інтерфейси або класи, що відповідають за взаємодію з базою даних. Вони забезпечують абстракцію доступу до даних, яка відокремлює бізнес-логіку від операційних деталей бази даних. Репозиторії зазвичай описують методи вибору, вставки, оновлення та видалення даних із бази даних. Це дозволяє зосередитися на бізнес-логіці в сервісах, не вникаючи в подробиці використання бази даних.

Наприклад, у випадку додатку «Farmakon», репозиторій може містити методи отримання списку всіх ліків, здійснення пошуку ліків за певними критеріями, додавання нових ліків до бази даних, оновлення інформації про ліки або її видалення. Використання репозиторію в трьохрівневої архітектурі дозволяє розділяти компоненти, спрощує роботу з базою даних і робить систему більш масштабованою та зручною для обслуговування.

3.2.1 Розробка сервісу для реєстрації та автентифікації користувача

Одним з основних сервісів додатку є сервіс для реєстрації та автентифікації користувача. Він надає бізнес логіку для додавання нового користувача в базу даних при його реєстрації, роблячи при цьому перевірку, чи не існує користувача з такою електронною поштою в базі даних. Якщо ж ні, то сервіс створює новий об’єкт моделі User, та додає його в базу даних створюючи, при цьому, новий JWT токен для його подальшої авторизації в системі та кодує пароль. Приклад коду для реєстрації нового користувача наведений на рисунку 3.2.1.

```
public JwtResponse register(RegisterRequest registerRequest){  4 usages   d4ng3r228 +2
    if (userService.existUserByEmail(registerRequest.getEmail())){
        throw new RuntimeException("User is existed");
    }

    Role role = roleRepository.findByName(registerRequest.getRole())
            .orElseThrow(()-> new RuntimeException("Role not found"));

    User user = User.builder()
            .email(registerRequest.getEmail())
            .password(passwordEncoder.encode(registerRequest.getPassword()))
            .fullName(registerRequest.getFullName())
            .dateOfBirthday(registerRequest.getDateOfBirthday())
            .roles(Collections.singleton(role))
            .bonus(0.0)
            .build();

    userRepository.save(user);
    String jwtToken = jwtUtils.generateToken(user);

    return JwtResponse.builder()
            .token(jwtToken)
            .user(userMapper.toDto(user))
            .build();
}
```

Рис. 3.2.1 Приклад коду для реєстрації нового користувача

Автентифікація користувача відбувається за рахунок перевірки, чи є користувач з такою поштовою скринькою в базі, і, якщо є, перевіряє пароль, роблячи безпосередній запит в базу даних, що покращує безпеку додатку, адже пароль в такому випадку знаходиться лише в базі даних в закодованому вигляді. Приклад коду для автентифікації користувача наведений на рисунку 3.2.2.

```
public JwtResponse login(LoginRequest loginRequest) {  2 usages   d4ng3r228 +1 *
    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    loginRequest.getEmail(), loginRequest.getPassword()));
    UserDetails user = userDetailsService.loadUserByUsername(loginRequest.getEmail());

    SecurityContextHolder.getContext().setAuthentication(authentication);

    String jwtToken = jwtUtils.generateToken(user);
    return JwtResponse.builder()
            .token(jwtToken)
            .user(userMapper
                    .toDto(userService
                            .findUserByEmail(loginRequest.getEmail())))
            .build();
}
```

Рис. 3.2.2 Приклад коду для автентифікації користувача

3.2.2 Розробка сервісу для роботи з препаратами

Робота з препаратами є однією з найважливіших функцій розробленого додатку. Саме тому, через постійну робот з цим сервісом, його важливо оптимізувати. Основними методами в сервісі можна назвати метод для отримання всіх наявних препаратів для виведення на сторінку, метод для отримання всіх препаратів за категорією, а також методи для створення, редагування та видалення препарату. Приклад коду з цими методами зображений на рисунку 3.2.3.

```java
@Transactional(readOnly = true)  1 usage  d4ng3r228
public List<Preparation> getAllPreparations(PageRequest pageRequest){
    Page<Preparation> page = preparationRepository.findAll(pageRequest);
    return page.getContent();
}
@Transactional(readOnly = true)  1 usage  ivanpoltavskiy
public List<Preparation> getAllPreparationsByCategory(Long categoryId, PageRequest pageRequest){
    if (categoryService.existsById(categoryId)){
    return preparationRepository.findAllByCategoryId(categoryId, pageRequest);
    }
    else {
        throw new RuntimeException("Category not found");
    }
}
@Transactional  1 usage  new *
public Preparation createPreparation(Preparation preparation){
    Category category = categoryService.findByName(preparation.getCategoryName());
    preparation.setCategory(category);
    preparation.setCategoryName(category.getName());
    return preparationRepository.save(preparation);
}

@Transactional  1 usage  new *
public Preparation updatePreparation(Preparation preparation){
    if (!preparationRepository.existsPreparationById(preparation.getId())){
        throw new PreparationNotFoundException("Preparation not found");
    }
    Category category = categoryService.findByName(preparation.getCategoryName());
    preparation.setCategory(category);
    preparation.setCategoryName(category.getName());
    return preparationRepository.save(preparation);
}

@Transactional  1 usage  new *
public void deletePreparation(Long id){
    if (preparationRepository.existsPreparationById(id)){
        preparationRepository.deleteById(id);
    } else{
        throw new PreparationNotFoundException("Preparation not found");
    }
```

Рис. 3.2.3 Приклад коду основних методів для роботи з препаратами

Але також є й додаткові методи, які виконують певну бізнес логіку додатку. Наприклад метод «reductionQuantity», який зображений на рисунку 3.2.4, при змінює кількість препаратів на умовному складі на ту кількість, яка приходить в параметрах. Викликається він в іншому сервісі, а саме в сервісі для замовлення, і при створенні замовлення за допомогою цього методу змінюється кількість товарів в базі даних.

```
@Transactional  1 usage  ivanpoltavskiy
public void reductionQuantity(Long preparationId, int amount){
    preparationRepository.findById(preparationId).ifPresent(preparation -> {
        int newAmount = preparation.getAmount() - amount;
        preparation.setAmount(newAmount);
        preparationRepository.save(preparation);
    });
}
```

Рис. 3.2.4 Приклад коду для зміни кількості препаратів в базі даних

Можна також виділити метод отримання препарату за його ідентифікатором (Id), в якому додатково прописана логіка для визначення всіх аналогів препарату за його активною речовиною. Код методу зображений на рисунку 3.2.5.

```
@Transactional(readOnly = true)  1 usage  ivanpoltavskiy
public PreparationResponse getPreparation(Long id){
    Preparation preparation = preparationRepository.findById(id)
            .orElseThrow(()->new PreparationNotFoundException("Preparation not found"));

    List<Preparation> analogs = preparationRepository.findAllByActiveSubstance(preparation.getActiveSubstance()) List<Preparation>
            .stream() Stream<Preparation>
            .filter(p -> !p.getId().equals(id)) // Відфільтрувати сам препарат, на якому викликається метод
            .limit( maxSize: 5) // Обмежити кількість аналогів до 5
            .collect(Collectors.toList());

    return preparationMapper.toDto(preparation, analogs);
}
```

Рис. 3.2.5 Приклад коду для отримання препарату за його ідентифікатором

3.2.3 Розробка сервісу для роботи з замовленням

Функціонал замовлення також є невід’ємною частиною додатку, оскільки він є одним із основних вимог до додатку. Замовлення забезпечують користувачам можливість купувати і бронювати медикаменти онлайн та отримувати інформацію про свої попередні покупки. Для реалізації цього функціоналу було розроблено сервіс, який відповідає за обробку замовлень та взаємодію з іншими компонентами системи, такими як сервіси підготовки даних та репозиторії.

Сервіс містить в собі 3 основні методи для створення замовлення, переведення статусу замовлення зі статусу «Забронювано» в статус «Завершено», та метод для отримання списку всіх замовлень користувача.

При створенні замовлення, об’єкт Order зберігається в базу даних. При цьому, саме на цьому моменті у користувача буде списано бонуси з його бонусного рахунку, якщо він захоче це зробити. Реалізація методу зображена на рисунку 3.2.6.

```
public Order takeNewOrder(Order order, User user) {  1 usage   ivanpoltavskiy +2 *
    order.setUser(user);
    order.setUserId(user.getId());
    List<OrderDetails> orderDetails = order.getOrderDetails();
    double bill = 0.0;
    for (OrderDetails orderDetail : orderDetails) {
        Preparation preparation = preparationService.getPreparationById(orderDetail.getPreparationId());
        bill += preparation.getPrice() * orderDetail.getQuantity();
        orderDetail.setName(preparation.getName());
        orderDetail.setPrice(preparation.getPrice());
        preparationService.reductionQuantity(orderDetail.getPreparationId(), orderDetail.getQuantity());
    }
    if (order.isApplyBonusPoints() && user.getBonus() > 0.0){
        bill = bill - user.getBonus();
        user.setBonus(0.0);
    }
    order.setBill(bill);
    return orderRepository.save(order);
}
```

Рис. 3.2.6 Приклад коду для створення нового замовлення

Будь-яке нове замовлення при його створенні автоматично вважається бронюванням. А вже після оплати товару, статус замовлення переходить в статус «Завершено», і, при цьому, користувачу на бонусний рахунок додається 5% бонусів від загальної суми замовлення. Реалізація методу представлена на рисунку 3.2.7

```
public void endingOrder(Long orderId) {  1 usage   d4ng3r228 +1
    Order currentOrder = orderRepository.getById(orderId);
    if (!currentOrder.isCompleted()) {
        currentOrder.setCompleted(true);
        User currentOrderUser = currentOrder.getUser();
        Double bonus = currentOrderUser.getBonus();
        currentOrderUser.setBonus(bonus + (currentOrder.getBill() / 20));
    }
    orderRepository.save(currentOrder);
}
```

Рис. 3.2.7 Приклад коду для завершення замовлення

Також є метод для отримання всіх замовлення користувача, реалізація якого показана на рисунку 3.2.8.

```
    @Transactional(readOnly = true)  1 usage  ivanpoltavskiy
    public List<Order> getAllOrdersByUser(User user) {
        return orderRepository.findAllByUser(user);
    }
}
```

Рис.3.2.8 Приклад коду для отримання всіх замовлень користувача

3.2.4 Розробка сервісу для перевірки валідності рецептів для рецептурних препаратів

Однією з функцій, яка кардинально відрізняє «Farmakon» від інших додатків, є процес замовлення рецептурних препаратів. Зазвичай, для таких препаратів в інших системах не потребується рецепту. В додатку «Famakon», щоб мати можливість додати рецептурний препарат до замовлення, користувачу потрібно ввести номер рецепту, після чого система перевіряє на валідність цей рецепт. Якщо рецепт з вказаним номером є в базі даних, метод «checkRecipe» перевіряє на відповідність діючу речовину, яка вказана в рецепті, з діючою речовиною препарату, який користувач хоче додати до кошика. Також перевіряється відповідність кількості препаратів. Реалізація методу зображена на рисунку 3.2.9.

```
@Transactional  1 usage  ivanpoltavskiy
public boolean checkRecipe(RecipeCheckRequest recipeRequest) {

    Preparation preparation = preparationService.getPreparationById(recipeRequest.getPreparationId());
    Optional<Recipe> recipe = recipeRepository.findByNumber(recipeRequest.getNumber());
    if (recipe.filter(value -> preparation.getActiveSubstance().equals(value.getActiveSubstance()) &&
            value.getQuantity().equals(recipeRequest.getQuantity())).isPresent()) {
        recipeRepository.deleteByNumber(recipeRequest.getNumber());
        return true;
    } else return false;
}
```

Рис. 3.2.9 Приклад коду методу для перевірки валідності рецепту

3.3 Розробка безпекової складової додатку

Безпекова складова додатку є дуже важливою частиною системи, оскільки вона забезпечує надійність роботи додатку та аутентифікацію користувачів з подальшою перевіркою, що ці користувачі можуть робити в системі.

При розробці додатку для забезпечення безпеки використовувався модуль Spring Security. Він надає інструменти для створення безпечних веб-додатків, контролю доступу та управління автентифікацією і авторизацією користувачів. Цей модуль забезпечує високий рівень гнучкості завдяки широкому спектру можливостей конфігурації. Таким чином можна легко налаштовувати безпеку додатку відповідно до конкретних потреб за допомогою конфігураційних файлів XML, Java-коду або анотацій.

При цьому, Spring Security надає вбудовану підтримку різних методів автентифікації, наприклад: базова автентифікація, автентифікація за допомогою форми входу, OAuth2, тощо. Також є вбудована підтримка авторизації у вигляді рольового доступу, методу рівня виразів тощо. Це дозволяє швидко впровадити надійний механізм контролю доступу. Цей модуль також легко інтегрується з іншими компонентами Spring.

Spring Security забезпечує захист від багатьох поширених вразливостей, таких як Cross-Site Scripting , CSRF, Clickjacking, Session Fixation та інші. Це робить додатки безпечнішими без необхідності додаткових налаштувань.

Реалізація безпекової компоненти в додатку «Farmakon» відбувається у декількох класах. Основний клас з налаштуваннями – «BaseSecurityConfig». Код класу зображений на рисунках 3.3.1 та 3.3.2. В цьому класі прописана більша частина безпекової компоненти додатку, а саме:

1. HTTP безпека: Вимикає захист від CSRF, що є доречним, оскільки в додатку використовується JWT токен, дозволяє налаштування CORS і обробляє виключення при неавторизованих запитах.

2. Аутентифікація: Додає фільтр для обробки JWT токенів, налаштовує аутентифікаційний провайдер з використанням кастомного сервісу користувачів і шифрування паролів.

3. Авторизація: Визначає правила доступу до різних ресурсів, дозволяючи доступ до деяких публічних ресурсів без авторизації та обмежуючи доступ до адміністративних функцій тільки для користувачів з роллю ADMIN.

Цей клас встановлює основні механізми для забезпечення безпеки додатку, включаючи аутентифікацію користувачів, обмеження доступу на основі ролей, і захист HTTP запитів.

```
@EnableWebSecurity  d4ng3r228 +2
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(securedEnabled = true)

public class BaseSecurityConfig implements WebMvcConfigurer {

    private final JwtAuthEntryPoint authEntryPoint;
    private final UserDetailsServiceCustom userDetailsService;
    private final JwtAuthFilter jwtAuthFilter;

    @Bean  d4ng3r228 +2
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
        http
                .csrf() CsrfConfigurer<HttpSecurity>
                .disable() HttpSecurity
                .cors(Customizer.withDefaults())
                .sessionManagement() SessionManagementConfigurer<HttpSecurity>
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and() HttpSecurity
                .exceptionHandling() ExceptionHandlingConfigurer<HttpSecurity>
                .authenticationEntryPoint(authEntryPoint)
                .and() HttpSecurity
                .authenticationProvider(authenticationProvider())
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
                .authorizeRequests() ExpressionInterceptUrlRegistry
                .antMatchers("/category/findAll",
                        "/preparation/getAllAnalogs/*",
                        "/preparation/getAll",
                        "/preparation/getById/*",
                        "/preparation/byCategory/**",
                        "/auth/**") AuthorizedUrl
                .permitAll() ExpressionInterceptUrlRegistry
                .antMatchers("/admin/getUsers").hasRole("ADMIN")
                .antMatchers(HttpMethod.POST, "/preparation", "/category", "/recipe/create").hasRole("ADMIN")
                .antMatchers(HttpMethod.PUT, "/preparation", "/category").hasRole("ADMIN")
                .antMatchers(HttpMethod.DELETE, "/preparation/*", "/category/*").hasRole("ADMIN")
                .anyRequest().authenticated();

        return http.build();
    }
```

Рис. 3.3.1 Приклад коду для безпекової компоненти додатку

```
    @Bean  d4ng3r228
    public AuthenticationProvider authenticationProvider(){
        final DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService);
        authenticationProvider.setPasswordEncoder(encoder());
        return authenticationProvider;
    }
    @Bean  d4ng3r228
    public PasswordEncoder encoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean  d4ng3r228
    public AuthenticationManager authenticationManagerBean(AuthenticationConfiguration authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

Рис. 3.3.2 Приклад коду для допоміжних методів для налаштування безпеки

Як раніше зазначалось, в додатку використовується JWT токен для авторизації користувача. Налаштування JWT токену зображено на рисунку 3.3.3.

```
@Component  d4ng3r228
@RequiredArgsConstructor
@Slf4j
public class JwtAuthFilter extends OncePerRequestFilter {
    private static final String TOKEN_PREFIX ="Bearer "; 2 usages
    private final JwtUtils jwtUtils;
    private final UserDetailsService userDetailsService;

    @Override  d4ng3r228
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        final String authHeader = request.getHeader(AUTHORIZATION);
        final String userEmail;
        final String jwtToken;

        if (authHeader == null || !authHeader.startsWith(TOKEN_PREFIX)){
            filterChain.doFilter(request, response);
            return;
        }

        jwtToken = authHeader.substring(TOKEN_PREFIX.length());
        userEmail = jwtUtils.extractUsername(jwtToken);

        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userDetailsService.loadUserByUsername(userEmail);

            if (jwtUtils.isTokenValid(jwtToken, userDetails)){
                UsernamePasswordAuthenticationToken authToken =
                        new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities());
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
            else {
                log.error("Access token has expired or revoked");
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Рис. 3.3.3 Приклад коду для налаштування роботи JWT токену

3.4 Тестування серверної частини

Тестування відіграє важливу роль у розробці серверної частини програми, забезпечуючи якість, стабільність і безпеку програмного забезпечення. Це дозволяє виявляти та виправляти помилки на ранній стадії розробки, зменшуючи ризик винику помилок у майбутньому. Без належного тестування програма може мати несправності, які можуть привести до збоїв роботи додатку. Тестування гарантує, що всі компоненти серверної частини функціонують згідно з вимогами. Це включає перевірку логіки бізнес-процесів, коректності обробки даних, роботи з базою даних і взаємодії з іншими сервісами. Завдяки тестам, розробники можуть бути впевнені, що кожна частина системи працює як очікується, навіть після внесення змін або додавання нових функцій.

Тестування також допомагає забезпечити безпеку серверної частини програми. Виявлення вразливостей на ранній стадії розробки допомагає запобігти потенційним загрозам для системи. Перевірка обробки даних і запобігання SQL-ін’єкціям та іншим типам атак є важливою частиною тестування безпеки. Тести також дозволяють покращити код. При написані тестів, розробник змушений думати про структуру та логіку коду, що часто призводить до чистішого та зручнішого коду. Це також сприяє кращій проектній документації, оскільки тести можуть служити живими документами, що показують, як працюватимуть різні частини системи.

При розробці додатку «Farmakon» тестування відбувалось двома шляхами, а саме за допомогою інтеграційних та юніт тестів безпосередньо в проекті, а також за допомогою API-тестування в програмі Postman.

3.4.1 Інтеграційне та юніт тестування за допомогою Junit 5 та Mockito

JUnit 5 та Mockito - це інструменти для тестування програмного забезпечення в Java. JUnit 5 - це фреймворк для написання та виконання юніт-тестів, які перевіряють окремі компоненти програми, що дозволяє прямо під час розробки

бути впевненим, що написані структури працюють правильно. JUnit 5 дозволяє легко писати тести, використовуючи анотації для позначення тестових методів та проведення певних дій перед або після тестом. Він має зручний інтерфейс для перевірки очікуваних результатів тестів та надає широкий спектр можливостей при тестуванні програмного забезпечення.

Mockito - це бібліотека для створення підроблених об'єктів (моків), що дозволяють замінити реальні об'єкти в тестах. Mockito допомагає створювати моки для заміни залежностей в тестах, що дозволяє ізолювати код від зовнішніх факторів і забезпечує більш точне тестування. Таким чином йде перевірка конкретно визначеного функціоналу, а додаткові функції, які потрібні для написання цього тесту, будуть замінені дублерами. Ці інструменти ідеально поєднуються для написання ефективних юніт-тестів. JUnit 5 дозволяє писати самі тести, а Mockito - забезпечує можливість створювати моки для заміни залежностей. Це дозволяє легко тестувати компоненти програми окремо від інших частин системи, що сприяє покращенню якості коду та зменшенню кількості помилок.

При розробці додатку «Farmakon» важливою була перевірка основного функціоналу програми та його бізнес логіки, тому проводилось тестування саме сервісів програмного забезпечення.

Для початку проводилось тестування аутентифікації та авторизації користувача для впевненості, що безпекова складова проекту працює правильно.

Для уникнення дублювання коду і для конфігурації тестів був написаний метод «setUp», який показаний на рисунку 3.4.1. В ньому відбувається мінімально потрібна конфігурація для коректної роботи тесту. Також, важливо уточнити, що цей метод виконується перед кожним тестом без виключення за допомогою анотації @BeforeEach з фреймворку JUnit.

```
@BeforeEach  ivanpoltavskiy *
void setUp() {
    registerRequest = new RegisterRequest();
    registerRequest.setEmail("test@example.com");
    registerRequest.setPassword("password");
    registerRequest.setFullName("Test User");
    registerRequest.setDateOfBirthday(null);
    registerRequest.setRole("ROLE_USER");

    loginRequest = new LoginRequest();
    loginRequest.setEmail("test@example.com");
    loginRequest.setPassword("password");

    role = new Role();
    role.setName("ROLE_USER");

    user = User.builder()
            .email("test@example.com")
            .password("encodedPassword")
            .fullName("Test User")
            .dateOfBirthday(null)
            .roles(Collections.singleton(role))
            .bonus(0.0)
            .build();

    userDetails = org.springframework.security.core.userdetails.User.withUsername("test@example.com")
            .password("encodedPassword")
            .authorities("ROLE_USER")
            .build();

    authentication = mock(Authentication.class);
}
```

Рис. 3.4.1 Реалізація методу «setUp» для конфігурації тестів

Далі на рисунках 3.4.2 та 3.4.3 зображені основні тести сервісу для перевірки коректної роботи та правильної реєстрації, авторизації та автентифікації користувача.

```java
@Test  ivanpoltavskiy
void register_ShouldRegisterUser_WhenValidRequest() {
    when(userService.existUserByEmail(registerRequest.getEmail())).thenReturn( value: false);
    when(roleRepository.findByName(registerRequest.getRole())).thenReturn(Optional.of(role));
    when(passwordEncoder.encode(registerRequest.getPassword())).thenReturn( value: "encodedPassword");
    when(userRepository.save(any(User.class))).thenReturn(user);
    when(jwtUtils.generateToken(any(User.class))).thenReturn( value: "jwtToken");
    when(userMapper.toDto(any(User.class))).thenReturn( value: null); // Return a proper DTO if needed

    JwtResponse jwtResponse = authService.register(registerRequest);

    assertNotNull(jwtResponse);
    assertEquals( expected: "jwtToken", jwtResponse.getToken());
    verify(userRepository, times( wantedNumberOfInvocations: 1)).save(any(User.class));
}

@Test  ivanpoltavskiy
void register_ShouldThrowException_WhenUserAlreadyExists() {
    when(userService.existUserByEmail(registerRequest.getEmail())).thenReturn( value: true);

    Exception exception = assertThrows(RuntimeException.class, () -> {
        authService.register(registerRequest);
    });

    assertEquals( expected: "User is existed", exception.getMessage());
}

@Test  ivanpoltavskiy
void register_ShouldThrowException_WhenRoleNotFound() {
    when(userService.existUserByEmail(registerRequest.getEmail())).thenReturn( value: false);
    when(roleRepository.findByName(registerRequest.getRole())).thenReturn( value: Optional.empty());

    Exception exception = assertThrows(RuntimeException.class, () -> {
        authService.register(registerRequest);
    });

    assertEquals( expected: "Role not found", exception.getMessage());
}
```

Рис. 3.4.2 Тестові методи для сервісу для реєстрації, авторизації та автентифікації користувача

```
@Test  ivanpoltavskiy
void login_ShouldAuthenticateAndReturnJwtResponse_WhenValidCredentials() {
    when(authenticationManager.authenticate(any(UsernamePasswordAuthenticationToken.class))).thenReturn(authentication);
    when(userDetailsService.loadUserByUsername(loginRequest.getEmail())).thenReturn(userDetails);
    when(jwtUtils.generateToken(any(UserDetails.class))).thenReturn( value: "jwtToken");
    when(userService.findUserByEmail(loginRequest.getEmail())).thenReturn(user);
    when(userMapper.toDto(any(User.class))).thenReturn( value: null); // Return a proper DTO if needed

    JwtResponse jwtResponse = authService.login(loginRequest);

    assertNotNull(jwtResponse);
    assertEquals( expected: "jwtToken", jwtResponse.getToken());
    verify(authenticationManager, times( wantedNumberOfInvocations: 1)).authenticate(any(UsernamePasswordAuthenticationToken.class));
}

@Test  ivanpoltavskiy
void changePassword_ShouldChangePassword_WhenValidCurrentPassword() {
    when(passwordEncoder.matches( rawPassword: "currentPassword", user.getPassword())).thenReturn( value: true);
    when(passwordEncoder.matches( rawPassword: "newPassword", user.getPassword())).thenReturn( value: false);
    when(passwordEncoder.encode( rawPassword: "newPassword")).thenReturn( value: "encodedNewPassword");
    when(userRepository.save(any(User.class))).thenReturn(user);

    User updatedUser = authService.changePassword( currentPassword: "currentPassword",  newPassword: "newPassword", user);

    assertNotNull(updatedUser);
    assertEquals( expected: "encodedNewPassword", updatedUser.getPassword());
    verify(userRepository, times( wantedNumberOfInvocations: 1)).save(any(User.class));
}

@Test  ivanpoltavskiy
void changePassword_ShouldThrowException_WhenCurrentPasswordDoesNotMatch() {
    when(passwordEncoder.matches( rawPassword: "currentPassword", user.getPassword())).thenReturn( value: false);

    Exception exception = assertThrows(RuntimeException.class, () -> {
        authService.changePassword( currentPassword: "currentPassword",  newPassword: "newPassword", user);
    });

    assertEquals( expected: "Passwords do not match", exception.getMessage());
}
```

Рис. 3.4.3 Тестові методи для сервісу для реєстрації, авторизації та автентифікації користувача

Наступним сервісом, який був покритий тестами, був сервіс для оформлення замовлень. Так як цей сервіс являє собою дуже важливу частину проекту, його тестування також проводилось за допомогою інтеграційних та юніт тестів, оскільки важливо зменшити ризик появ помилок при роботі з таким функціоналом. Тестові методи для цього сервісу зображені на рисунках 3.4.4 та 3.4.5.

```java
@Test  ivanpoltavskiy *
public void testTakeNewOrder() {
    User user = new User();
    user.setId(1L);
    user.setBonus(100.0);

    Preparation preparation = new Preparation();
    preparation.setId(1L);
    preparation.setName("Test Preparation");
    preparation.setPrice(10.0);

    OrderDetails orderDetails = new OrderDetails();
    orderDetails.setPreparationId(1L);
    orderDetails.setQuantity(2);

    List<OrderDetails> orderDetailsList = new ArrayList<>();
    orderDetailsList.add(orderDetails);

    Order order = new Order();
    order.setId(1L);
    order.setOrderDetails(orderDetailsList);
    order.setApplyBonusPoints(false);
    order.setDateOfOrder(new Date());
    order.setNumber(1L);

    when(preparationService.getPreparationById(1L)).thenReturn(preparation);
    when(orderRepository.save(any(Order.class))).thenAnswer(invocation -> {
        Order savedOrder = invocation.getArgument( index: 0);
        savedOrder.setId(1L);
        return savedOrder;
    });

    Order savedOrder = orderService.takeNewOrder(order, user);

    assertEquals( expected: 1, savedOrder.getOrderDetails().size());
    assertEquals( expected: 20.0, savedOrder.getBill());
    assertEquals( expected: 100.0, user.getBonus());
    verify(orderRepository, times( wantedNumberOfInvocations: 1)).save(order);
}
```

Рис. 3.4.4 Тестовий метод для перевірки успішного створення замовлення

```java
@Test  ivanpoltavskiy
public void testEndingOrder() {
    Order order = new Order();
    order.setId(1L);
    order.setCompleted(false);
    order.setBill(100.0);

    User user = new User();
    user.setBonus(0.0);
    order.setUser(user);

    when(orderRepository.getById(1L)).thenReturn(order);

    orderService.endingOrder( orderId: 1L);

    assertTrue(order.isCompleted());
    assertEquals( expected: 5.0, user.getBonus());
    verify(orderRepository, times( wantedNumberOfInvocations: 1)).save(order);
}

@Test  ivanpoltavskiy
public void testGetAllOrdersByUser() {
    User user = new User();
    user.setId(1L);

    List<Order> orders = new ArrayList<>();
    Order order1 = new Order();
    order1.setUser(user);
    Order order2 = new Order();
    order2.setUser(user);
    orders.add(order1);
    orders.add(order2);

    when(orderRepository.findAllByUser(user)).thenReturn(orders);

    List<Order> result = orderService.getAllOrdersByUser(user);

    assertEquals( expected: 2, result.size());
    assertEquals(user, result.get(0).getUser());
    assertEquals(user, result.get(1).getUser());
}
```

Рис. 3.4.5 Тестові методи для перевірки завершення замовлення та виводу всіх замовлень користувача

Також було проведено тестування сервісу для перевірки рецептів на валідність. Цей сервіс є важливою частиною додатку, оскільки треба знизити ризики виникнення помилок при покупці користувачем рецептурних препаратів. Тестові методи для цього сервісу зображені на рисунку 3.4.6.

```
    @Test  new *
    public void testGetRecipe() {
        Recipe recipe = new Recipe();
        recipe.setId(1L);
        when(recipeRepository.findById(1L)).thenReturn(java.util.Optional.of(recipe));

        Recipe retrievedRecipe = recipeService.getRecipe( id: 1L);

        assertNotNull(retrievedRecipe);
        assertEquals( expected: 1L, retrievedRecipe.getId());
    }

    @Test  new *
    public void testCheckRecipe() {
        RecipeCheckRequest recipeRequest = new RecipeCheckRequest();
        recipeRequest.setPreparationId(1L);
        recipeRequest.setNumber(12345L);
        recipeRequest.setQuantity(10);

        Preparation preparation = new Preparation();
        preparation.setActiveSubstance("Substance");
        when(preparationService.getPreparationById(1L)).thenReturn(preparation);

        Recipe recipe = new Recipe();
        recipe.setActiveSubstance("Substance");
        recipe.setQuantity(10);
        when(recipeRepository.findByNumber(12345L)).thenReturn(java.util.Optional.of(recipe));

        boolean result = recipeService.checkRecipe(recipeRequest);

        assertTrue(result);
        verify(recipeRepository, times( wantedNumberOfInvocations: 1)).deleteByNumber(12345L);
    }
}
```

Рис. 3.4.6 Тестові методи для перевірки сервісу для валідації рецептів

3.4.2 Тестування за допомогою Postman

Ще одним видом тестування, яке було використано при розробці, є API-тестування. API-тестування - це процес перевірки програмного інтерфейсу додатка (API), який визначає способи взаємодії між різними компонентами програмного забезпечення. Це тестування спрямоване на перевірку правильності взаємодії між програмними модулями, їх функціональність та відповідність специфікаціям API.

Основні аспекти API-тестування включають функціональність, надійність, безпеку та відмовостійкість. Функціональне тестування перевіряє, як API реагує на різні вхідні дані та умови, чи повертає очікувані результати та статуси відповіді. Надійність тестується шляхом перевірки стабільності та надійності API під навантаженням, відсутністю витоків пам'яті або інших проблем. Безпека тестування гарантує, що API захищений від вразливостей, таких як атаки злому або переповнення буфера. Відмовостійкість перевіряє, як API поводиться при негативних умовах, таких як втрата з'єднання або відмова сервера. При розробці було проведено лише функціональне тестування, оскільки розмір додатку зводить до мінімуму можливі проблеми з надійністтю та відмовостійкісттю.

Для виконання API-тестування використовуються спеціалізовані інструменти, такі як Postman, SoapUI, REST Assured та інші. Ці інструменти дозволяють легко створювати, виконувати та аналізувати результати тестів для різних типів API, таких як REST, SOAP, GraphQL тощо. У цій дипломній роботі було використана саме програма Postman для тестування REST API.

Важливість API-тестування полягає в гарантуванні якості програмного забезпечення, оскільки воно дозволяє виявляти проблеми в API на ранніх етапах розробки, забезпечуючи високу якість та стабільність програми.

Як і у випадку з інтеграційними та юніт тестами, спочатку перевірявся функціонал реєстрації, автентифікації та авторизації користувача. На рисунку 3.4.7 зображено тестування реєстрації користувача. Ми вводимо потрібні данні і правильний HTTP-запит і сервер нам повертає об’єкт користувача разом з JWT токеном для його подальшої авторизації в системі.

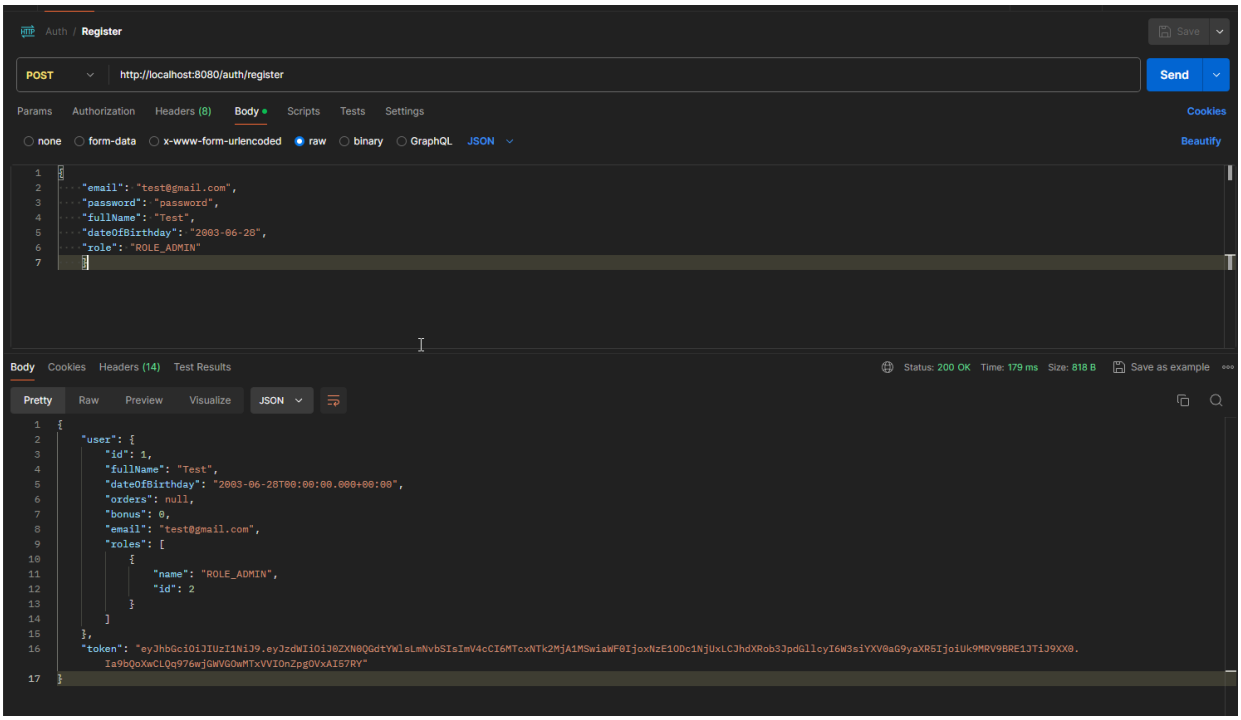


Рис. 3.4.7 Результат тестування реєстрації користувача в Postman

На рисунку 3.4.8 зображений результат тестування HTTP-методу для автентифікації користувача. Тут система перевіряє вхідні данні, а саме електронну пошту та пароль, і, якщо все вірно, також віддає відповідь з об’єктом користувача та його JWT токен.

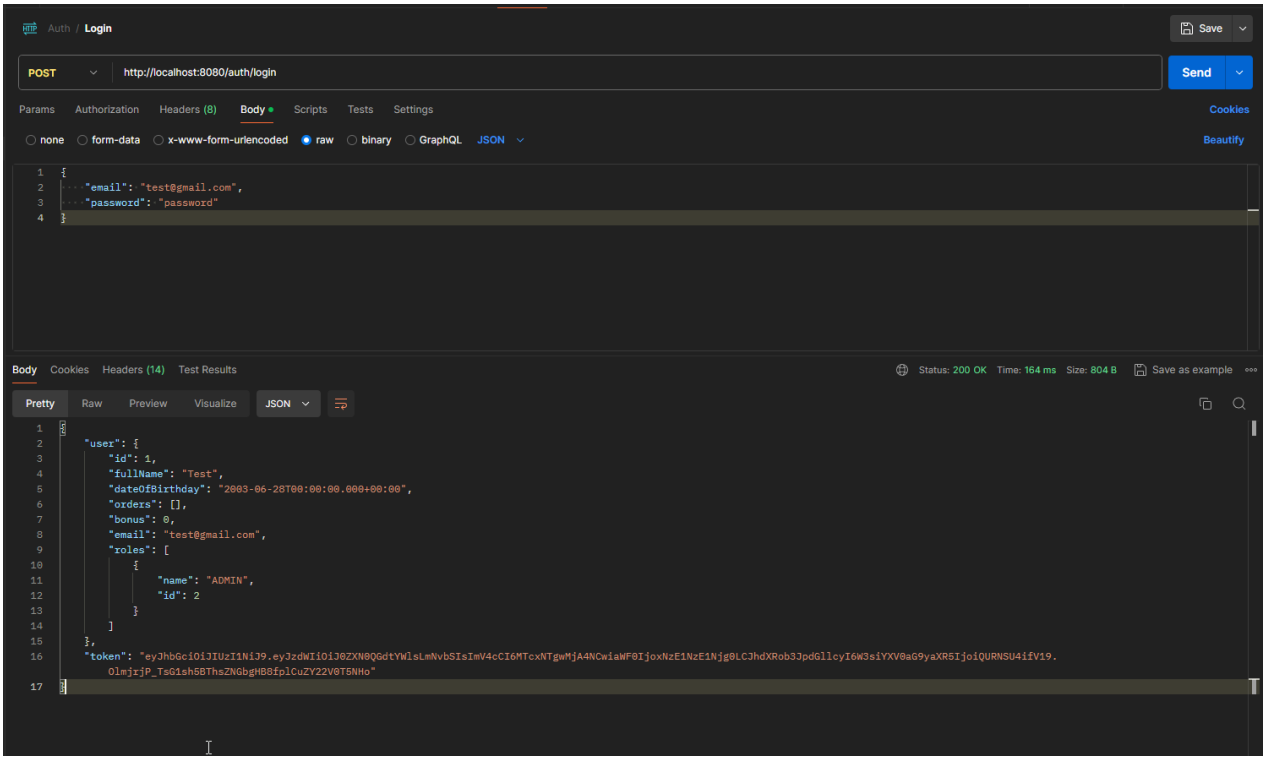


Рис. 3.4.8 Результат тестування автентифікації користувача в Postman

Для створення замовлення серверна частина потребує надати статус цього замовлення, данні про те, чи хоче користувач зняти наявні в нього бонуси, номер замовлення, і список препартів та їх кількість. Якщо все вдало, то система повертає об’єкт цього замовлення. Результат тестування методу зображений на рисунку 3.4.9.

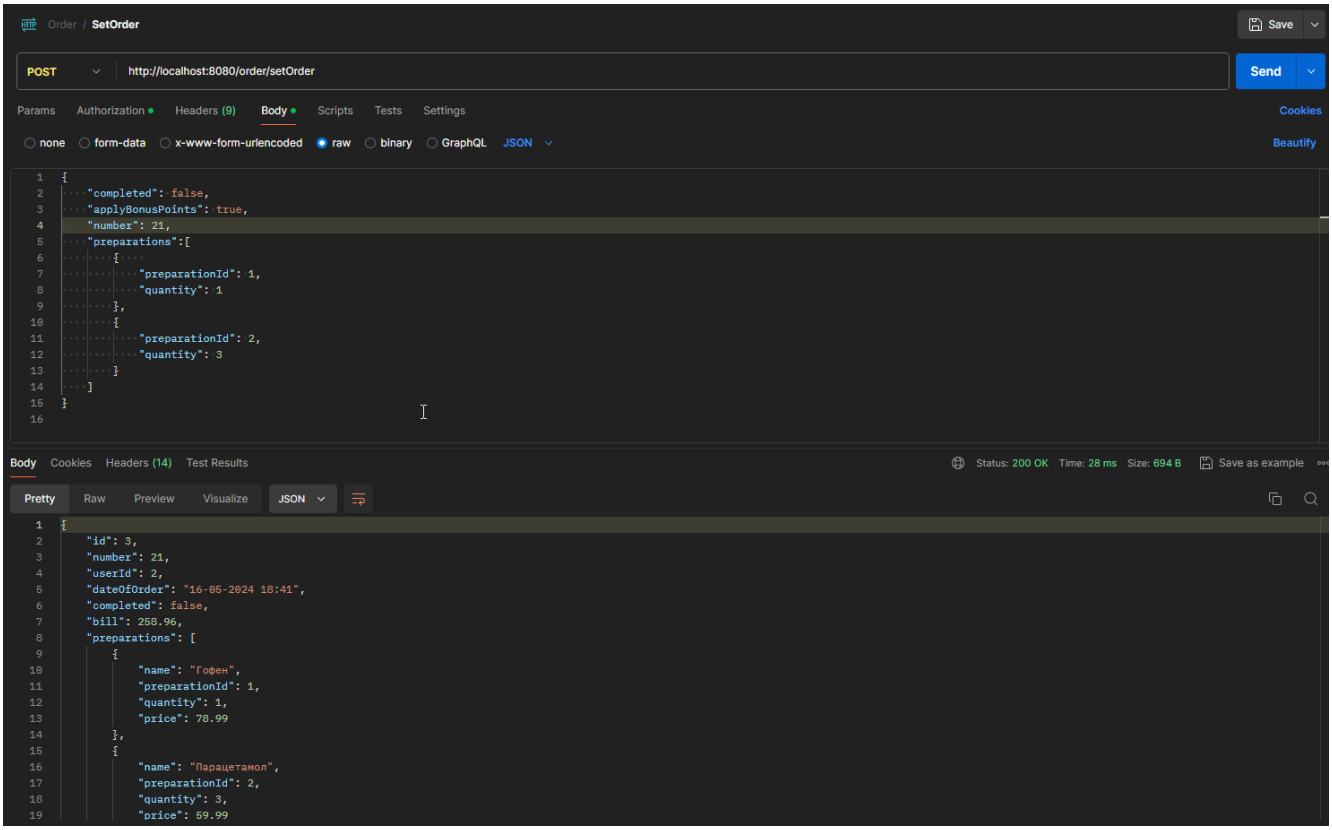


Рис. 3.4.9 Результат тестування створення нового замовлення в Postman

Перевірка рецепту потребує набір даних, який містить в собі номер цього рецепту, ідентифікатор препарату, який користувач хоче додати в кошик та кількість. Натомість, система повертає true, якщо всі введені данні збігаються з даними з бази, та false, якщо бодай хоча б один з цих параметрів не відповідає даним з бази даних. Результат тестування перевірки рецепту зображений на рисунку 3.4.10.

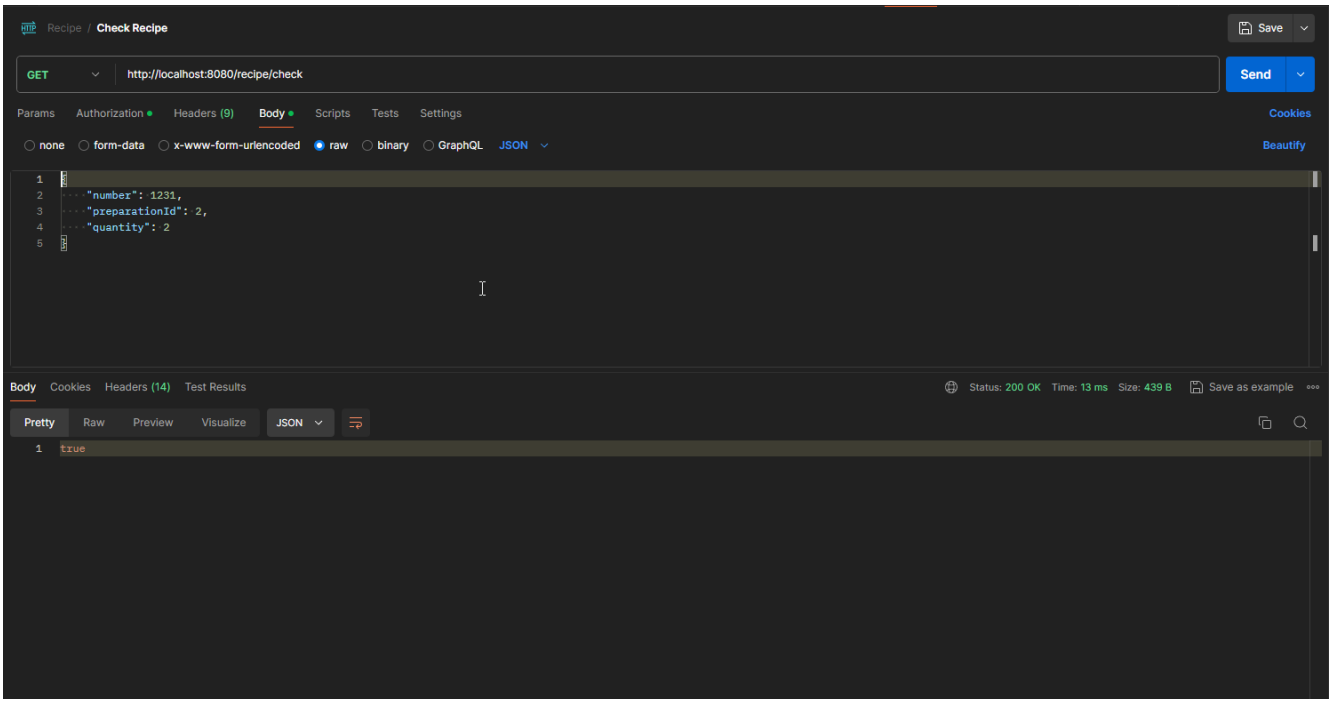


Рис. 3.4.10 Результат тестування перевірки рецепту на валідність в Postman

ВИСНОВКИ

В процесі написання бакалаврської роботи було проведено аналіз існуючих засобів для замовлення та бронювання ліків онлайн, визначено їх переваги та недоліки і, на основі результатів дослідження існуючих рішень, визначено вимоги до програмного забезпечення.

Було проведено огляд рішень для побудови веб-застосунку, включаючи архітектурні підходи, використання різних API, взаємодію з базами даних, засоби забезпечення безпеки та інші аспекти розробки.

Проведено огляд IT-засобів, які можуть бути використані при розробці додатку. Обрані технології забезпечують просте подальше розширення проекту, шляхом підключення сторонніх API або розробки додаткового функціоналу.

Була спроектована архітектура додатку та розроблена серверна частина веб застосунку на основі визначених вимог та з використанням вибраних технологій.

Проведено інтеграційні та юніт тести серверної частини додатку для забезпечення його працездатності.

Розроблена серверна частина додатку покриває більшість потреб потенційних користувачів на ринку, пропонуючи великий набір функціоналу, який, в деяких випадках, є більшим, а ніж в аналогах. Обрані технології забезпечують просте подальше розширення проекту, шляхом підключення сторонніх API або розробки додатково функціоналу. Таким чином, можна зробити висновок, що вибраний стек технологій вибраний правильно та є оптимальним для поставленої задачі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Боковець, В. В.; Л. П. Давидюк. "Електронна торгівля і її значення для розвитку бізнесу." ВІСНИК (2021): 1210.
2. Майборода, Ольга Євгенівна; Анастасія Юріївна Теребінська. "РИНОК ЕЛЕКТРОННОЇ КОМЕРЦІЇ В УКРАЇНІ ТА ЇЇ РОЗВИТОК." Multidisciplinary academic research, innovation and results 13 (2022): 218.
3. Дорикевич, Катерина, Курило Катерина; Дзвенислава Грушковська. "ДОСЛІДЖЕННЯ САЙТІВ АПТЕК В УКРАЇНІ ЯК СКЛАДОВОЇ ТЕЛЕФАРМАЦІЇ."
4. Селіверстова, Г. С. "АПТЕКИ УКРАЇНИ XXI СТОЛІТТЯ. ОНЛАЙН АПТЕКИ." INTERNATIONAL HUMANITARIAN UNIVERSITY: 55
5. Пестун, І.; Бабічева, Г. АНАЛІЗ ЦІЛЬОВОЇ АУДИТОРІЇ АПТЕЧНИХ ЗАКЛАДІВ У ПРОЦЕСІ СТВОРЕННЯ КЛІЄНТСЬКОЇ БАЗИ ДАНИХ. Фарм. час. 2014.
6. Аптека "Подорожник" [Електронний ресурс] – Режим доступу до ресурсу: https://podorozhnyk.ua/.
7. Аптека "911" [Електронний ресурс] – Режим доступу до ресурсу: https://apteka911.ua/ua.
8. Аптека "АНЦ" [Електронний ресурс] – Режим доступу до ресурсу: https://anc.ua/.
9. Tabletki.ua [Електронний ресурс] – Режим доступу до ресурсу: https://tabletki.ua/.
10. What is a RESTful API? [Електронний ресурс] // AWS. Amazon Web Services, Inc. – Режим доступу до ресурсу: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls.
11. MVC Framework Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/mvc-framework-introduction/.
12. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/en/
13. Spting Boot [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-boot.
14. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.
15. Hibernate [Електронний ресурс] – Режим доступу до ресурсу: https://hibernate.org/orm/.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку для аптеки мовою Javaз використанням фреймворкуAngular

Виконав студент 4 курсу

групи ПД-43
Полтавський Іван Андрійович
Керівник роботи

к.т.н. доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підтримка процесу вибору, бронювання та покупки лікарських препаратів онлайн.

Об’єкт дослідження - процесу вибору, бронювання та покупки лікарських препаратів онлайн.

Предмет дослідження – розробка серверної частини додатку “Farmakon” для покупки та бронювання лікарських препаратів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз існуючих засобів для вибору, бронювання та покупки лікарських препаратів онлайн в аптеці , визначити їх переваги та недоліки.
2. Визначити вимоги до Web-застосунку на основі результату дослідження існуючих рішень.
3. Провести огляд технічних рішень для побудови Web-застосунку для вибору, бронювання та покупки лікарських препаратів онлайн в аптеці, включаючи архітектурні підходи, використання різних API, взаємодію з базами даних, засоби забезпечення безпеки та інші аспекти розробки.
4. Розробити архітектуру Web-застосунку на основі проведеного огляду архітектурних підходів.
5. На основі розробленої архітектури розробити моделі та сервіси для реалізації бізнес-логіки.
6. Налаштувати базу даних та розробити DAO рівень для роботи з нею.
7. Провести тестування Web-застосунку для вибору, бронювання та покупки лікарських препаратів онлайн в аптеці.

3

АНАЛІЗ АНАЛОГІВ

Додаток / Характеристика	Аптека “Подорожник”	Аптека “911”	Аптека “АНЦ”	Сервіс Tabletki.ua	Додаток «Farmakon»
Оплата замовлення на сайті	+	+	+	-	+
Наявність бонусної програми	+	-	+	-	+
Врахування рецептурності препарату	-	-	-	-	+
Можливість оплати забронюваного замовлення	-	-	+	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні :

1. Можливість реєстрації та автентифікації користувача
2. Можливість користувачу редагувати свої дані.
3. Можливість користувача переглядати замовлення.
4. Можливість бронювання та замовлення препаратів.
5. Забезпечення функціональності управління системою для адміністраторів.
6. Перевірка валідності рецептів для рецептурних препаратів: для таких ліків користувач повинен ввести номер рецепту, після чого система перевіряє відповідність активної речовини, зазначеної в рецепті, активній речовині препарату, який користувач бажає придбати.
7. Наявність бонусної системи.

Нефункціональні :

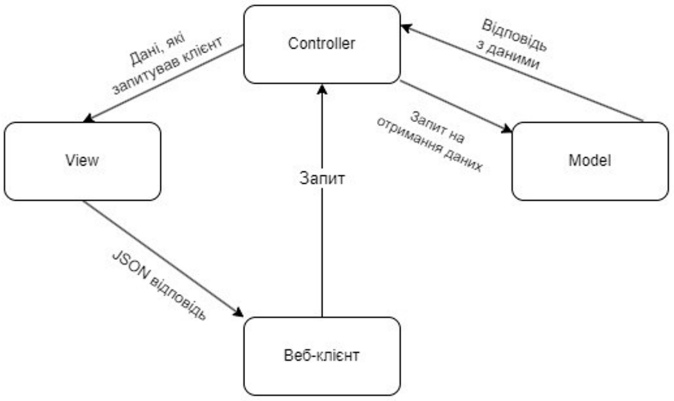
1. Легка масштабованість серверної частини додатку для його подальшого розвитку.
2. Система повинна бути простою в обслуговуванні завдяки правильно обраній архітектурі.
3. Авторизація користувача за допомогою JWT токену.
4. Система має бути мультиплатформеною, з можливістю запуску на операційних системах Windows, macOS та Linux.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

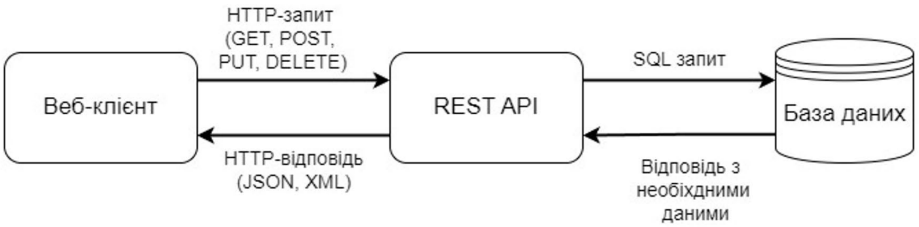
6

СХЕМА ПРЕДСТАВЛЕННЯ MVC АРХІТЕКТУРИ



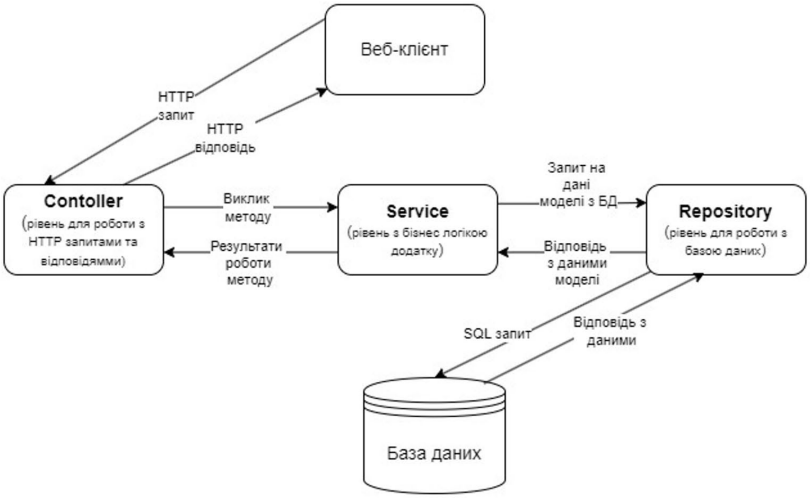
7

СХЕМА ПРЕДСТАВЛЕННЯ REST API



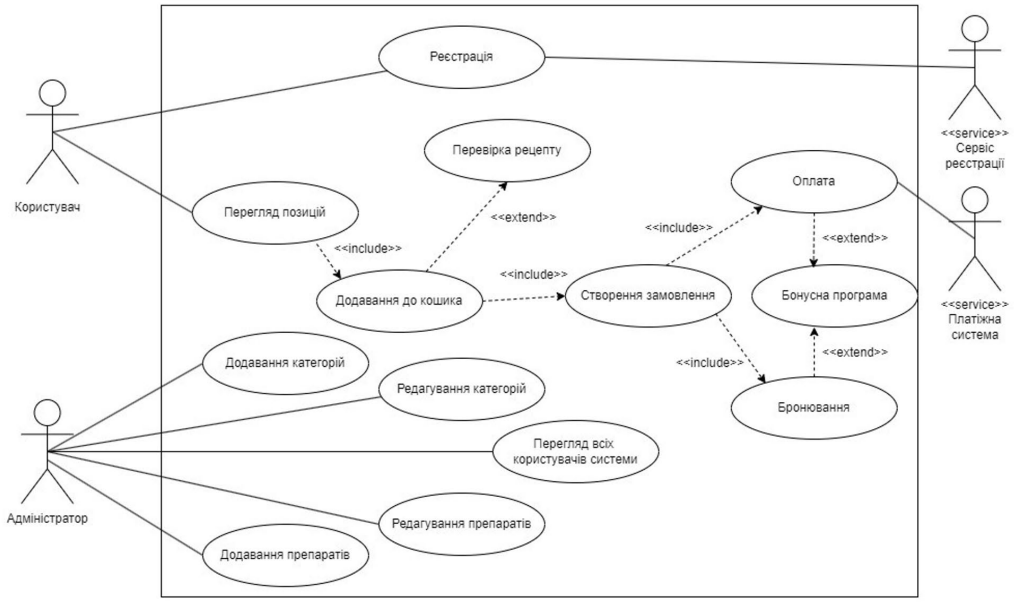
8

СХЕМА ПРЕДСТАВЛЕННЯ ТРЬОХРІВНЕВОЇ АРХІТЕКТУРИ



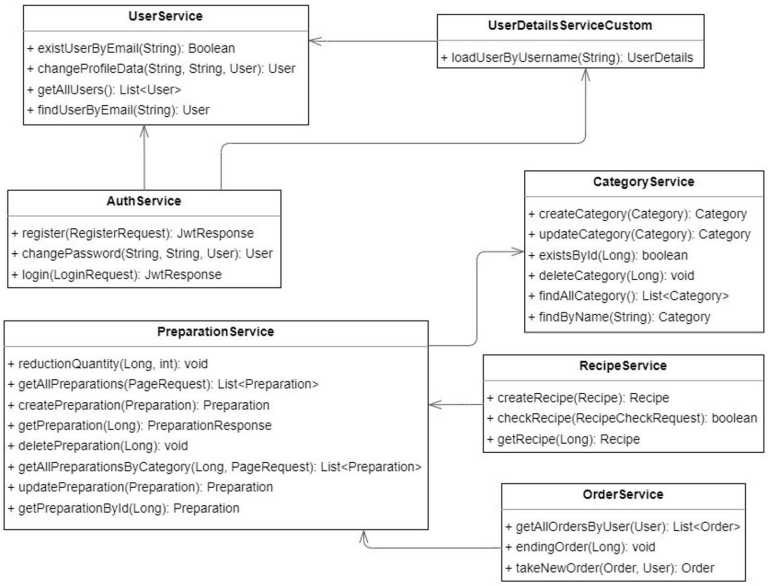
9

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



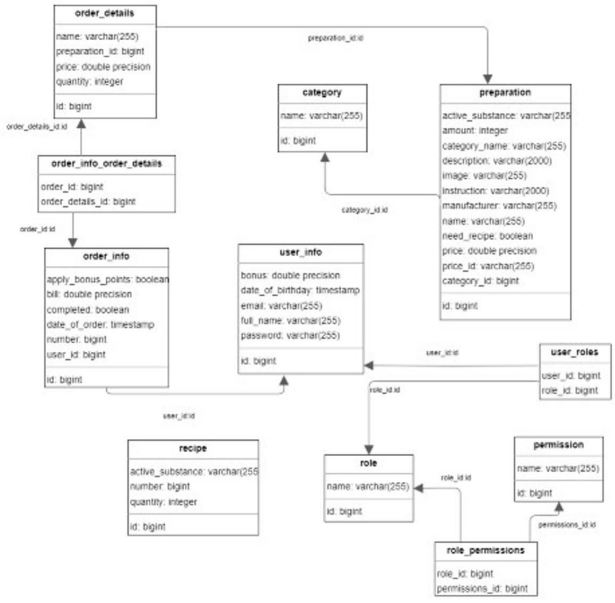
10

ДІАГРАМА КЛАСІВ



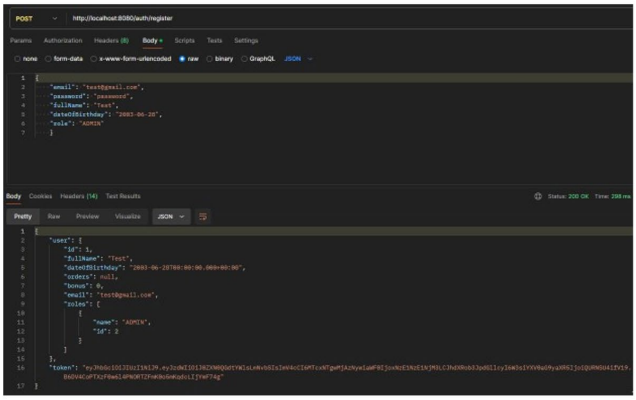
11

ДІАГРАМА БАЗИ ДАНИХ

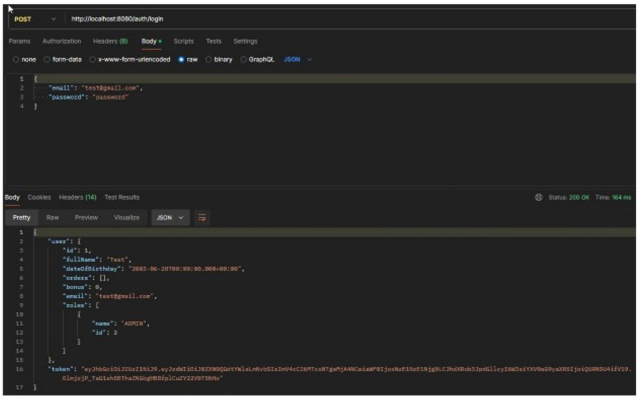


12

ДЕМОНСТРАЦІЯ РОБОТИ API

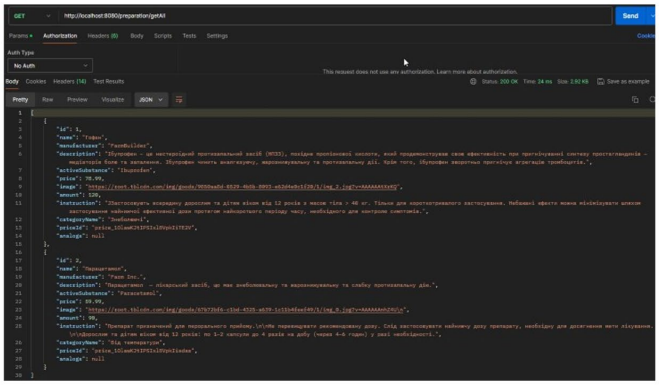


Реєстрація користувача

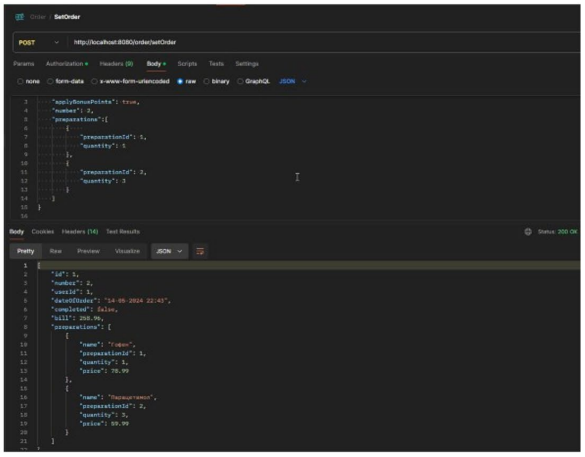


Автентифікація користувача

ДЕМОНСТРАЦІЯ РОБОТИ API

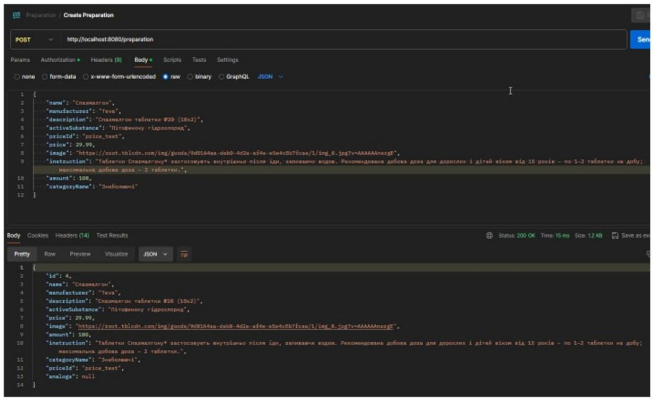


Вивід всіх наявних препаратів в системі

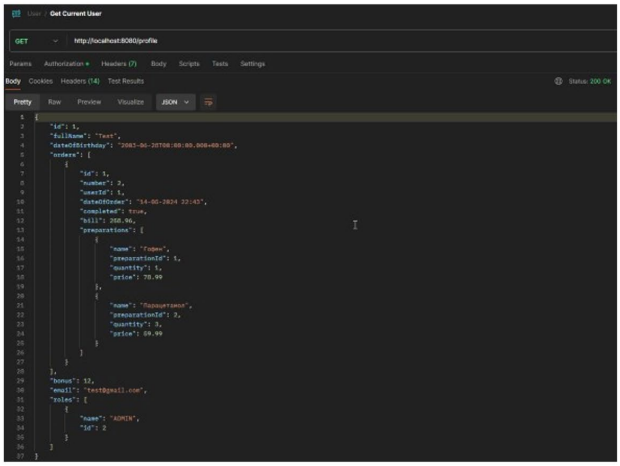


Створення нового замовлення

ДЕМОНСТРАЦІЯ РОБОТИ API



Додавання нового препарату адміністратором



Профіль користувача

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Полтавський І. А. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ ДЛЯ ЗАМОВЛЕННЯ ЛІКІВ ОНЛАЙН/ Яскевич В. О., Полтавський І. А. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційнө комунікаційних технологіях». Збірник тез. 24 квітня 2024 року, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С.108.

2. Полтавський І. А. ВИБІР НАБОРУ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ ВЕБ-ДОДАТКУ «FARMAKON»/ Яскевич В. О., Полтавський І. А. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційнө комунікаційних технологіях». Збірник тез. 24 квітня 2024 року, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 411.

ВИСНОВКИ

1. Проведено аналіз існуючих засобів для замовлення та бронювання ліків онлайн, визначено їх переваги та недоліки і, на основі результатівдослідження існуючих рішень, визначено вимоги до програмного забезпечення.
2. Проведено огляд рішень для побудови веб-застосунку, включаючи архітектурні підходи, використання різних API, взаємодію з базами даних, засоби забезпечення безпеки та інші аспекти розробки.
3. Спроектовано архітектуру додатку з використаням REST API, MVC та трьохрівневої архітектури.
3. Проведено огляд IT-засобів, які можуть бути використані при розробці додатку.Обрані технології забезпечують просте подальше розширення проекту, шляхом підключення сторонніх API або розробки додаткового функціоналу.
4. Розроблено серверну частину веб застосунку на основі визначених вимог та з використанням вибраних технологій.
5. Проведено інтеграційні та юніт тести серверної частини додатку для забезпечення його працездатності.