

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Веб-система управління замовленнями для сервісу
доставки»**

Виконав студент групи З-ІПЗ-22-1
ЛЕГЕНЬКИЙ Костянтин Миколайович

Керівник роботи:
ПІОНТКІВСЬКИЙ Владислав Ігорович

Рецензент
ГОДЛЕВСЬКИЙ Юрій Олександрович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«27» лютого 2026 р.

ЗАВДАННЯ
на випускню кваліфікаційну роботу бакалавра

Здобувач вищої освіти: **ЛЕГЕНЬКИЙ Костянтин Миколайович**

Керівник роботи: **ПІОНТКІВСЬКИЙ Владислав Ігорович**

Тема роботи: **«Веб-система управління замовленнями для сервісу доставки»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №93/с**

Вихідні дані для роботи: об'єктом дослідження є процес приймання, обробки та супроводження замовлень у сервісі доставки, що охоплює ведення каталогу товарів і акцій, оформлення замовлення оператором, облік позицій, контроль статусів виконання, керування робочими змінами та призначення кур'єра; а предметом дослідження є методи та програмні засоби розробки веб-орієнтованих систем, що застосовуються для побудови операційної платформи, яка має бути надійною з технічного боку та водночас зручною, захищеною й придатною до подальшого розвитку.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Косультант	Завдання видав	Завдання прийняв
1	Піонтківський В. І.	16.03.2026	16.03.2026
2	Піонтківський В. І.	16.03.2026	16.03.2026
3	Піонтківський В. І.	16.03.2026	16.03.2026

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-системи управління замовленнями для сервісу доставки та пояснювальної записки. Загальний обсяг пояснювальної записки становить 80 сторінок, з них 64 сторінки основного тексту; робота містить 10 рисунків, 7 таблиць, 12 додатків та список використаних джерел із 36 найменувань.

Виконано розробку веб-системи управління замовленнями для сервісу доставки, яка забезпечує користувачам можливість реєстрації, авторизації, перегляду каталогу товарів, формування замовлень та відстеження їхнього стану. Основна ідея полягає у створенні зручного програмного рішення для автоматизації процесів приймання, обробки та супроводу замовлень у сервісі доставки.

У рамках проєкту було визначено ключові завдання розробки, проведено аналіз предметної області та функціональних вимог до системи, а також обґрунтовано вибір архітектурного підходу. Було реалізовано розподіл проєкту на доменний та інфраструктурний модулі відповідно до принципів гексагональної архітектури. Для реалізації застосунку обрано сучасний технологічний стек, включно з Java, Spring Boot, Spring MVC, Spring Security, PostgreSQL та Docker, що забезпечує надійність, розширюваність і зручність подальшого розвитку системи.

КЛЮЧОВІ СЛОВА: ВЕБ-СИСТЕМА, СЕРВІС ДОСТАВКИ, УПРАВЛІННЯ ЗАМОВЛЕННЯМИ, ГЕКСАГОНАЛЬНА АРХІТЕКТУРА, JAVA, SPRING BOOT, POSTGRESQL, DOCKER.

					3-ІПЗ-22-1			
Змн.	Арк.	№ докум.	Підпис	Дата	Веб-система управління замовленнями для сервісу доставки	Літ.	Арк.	Аркушів
Розроб.		Легенький К. М.					3	80
Керівник		Піонтківський В. І.						
Н. контр.		Тичина О. П.				Житомирська політехніка, група 3-ІПЗ-22-1		
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's qualification thesis consists of a web-based order management system for a delivery service and an explanatory note. The total volume of the explanatory note is 80 pages, including 64 pages of the main text; it contains 10 figures, 7 tables, 12 appendices, and a list of 36 references.

A web-based order management system for a delivery service was developed. The system allows users to register, log in, browse the product catalog, create orders, and track their status. The main idea is to create a convenient software solution for automating the processes of receiving, processing, and supporting orders in a delivery service.

As part of the project, the main development tasks were defined, the subject area and functional requirements of the system were analyzed, and the choice of architectural approach was justified. The project was divided into domain and infrastructure modules according to the principles of hexagonal architecture. The application was implemented using a modern technology stack, including Java, Spring Boot, Spring MVC, Spring Security, PostgreSQL, and Docker, which ensures reliability, scalability, and convenient further development of the system.

KEYWORDS: WEB SYSTEM, DELIVERY SERVICE, ORDER MANAGEMENT, HEXAGONAL ARCHITECTURE, JAVA, SPRING BOOT, POSTGRESQL, DOCKER.

					3-ІІІЗ-22-1			
Змн.	Арк.	№ докум.	Підпис	Дата	Веб-система управління замовленнями для сервісу доставки	Літ.	Арк.	Аркушів
Розроб.		Легенький К. М.					4	80
Керівник		Піонтківський В. І.				Житомирська політехніка, група 3-ІІІЗ-22-1		
Н. контр.		Тичина О. П.						
Зав. каф.		Вакалюк Т. А.						

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ	9
1.1. Сервіси доставки як предметна галузь та бізнес-процеси діяльності	9
1.2. Задачі автоматизації та аналіз існуючих аналогів	13
1.3. Вимоги до системи та обґрунтування архітектурного підходу	18
Висновки до розділу 1	23
РОЗДІЛ 2. ОПИС СТЕКУ ТЕХНОЛОГІЙ ТА АРГУМЕНТАЦІЯ ВИБОРУ	24
2.1. Загальна характеристика стеку та платформа Java і Spring Boot.....	24
2.2. Архітектура застосунку та організація доступу до даних	28
2.3. Контейнеризація бази даних та підсистема безпеки.....	35
2.4. Фронтенд, порівняння альтернатив та відповідність вимогам.....	38
Висновки до розділу 2	44
РОЗДІЛ 3. ОПИС ВИКОРИСТАННЯ ПРОГРАМИ.....	46
3.1. Встановлення системи та структура інтерфейсу	46
3.2. Оформлення замовлень та керування їх виконанням	50
3.3. Робота кур'єра, сповіщення та аналітика робочих змін	56
3.4. Адміністрування системи та тестування програмного комплексу	59
Висновки до розділу 3	63
ВИСНОВКИ.....	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	65
ДОДАТКИ.....	68

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — прикладний програмний інтерфейс

CRM — система керування взаємодією з клієнтами

CRUD — базові операції над даними

DTO — об'єкт передачі даних

JPA — стандарт об'єктно-реляційного відображення

JWT — стандарт токена доступу

ORM — об'єктно-реляційне відображення

OTP — одноразовий код підтвердження

REST — архітектурний стиль вебсервісів

SPA — односторінковий застосунок

SQL — мова структурованих запитів

UML — уніфікована мова моделювання

СКБД — система керування базами даних

ПЗ — програмне забезпечення

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Сучасні сервіси доставки товарів і готових страв перетворилися на масовий спосіб обслуговування клієнтів, а конкуренція між ними дедалі частіше визначається не лише асортиментом, а й якістю внутрішніх операційних процесів. За оглядами цифрового ринку кількість онлайн-замовлень в Україні стабільно зростає, а користувачі очікують швидкого підтвердження, точного складу замовлення та прозорого відстеження його стану [14]. Для невеликого чи середнього сервісу це означає, що ручне ведення замовлень у месенджерах і паперових журналах перестає бути прийнятним: воно породжує помилки в сумах, втрату історії та неможливість контролю навантаження.

Актуальність теми зумовлена потребою у спеціалізованому програмному забезпеченні, яке формалізує операційну діяльність сервісу доставки — від приймання замовлення оператором до підтвердження виконання кур'єром. На відміну від публічного інтернет-магазину, де основним користувачем є клієнт, внутрішня диспетчерська система обслуговує насамперед працівників сервісу й має забезпечувати рольовий розподіл прав, облік робочих змін, контроль статусів та аналітику.

Метою роботи є розробка алгоритмів та реалізація програмного комплексу веб-орієнтованої системи управління замовленнями для сервісу доставки, що автоматизує приймання, обробку та супроводження замовлень, ведення каталогу й акцій, облік робочих змін і призначення кур'єрів.

Для досягнення поставленої мети визначено такі завдання: проаналізувати предметну галузь сервісів доставки та визначити задачі автоматизації; виконати огляд аналогів і сформулювати функціональні та нефункціональні вимоги; обґрунтувати вибір технологічного стеку та архітектури; реалізувати програмний комплекс із захищеним доступом за ролями; описати порядок користування програмою та виконати тестування реалізованого функціоналу.

Об'єктом дослідження є процес приймання, обробки та супроводження замовлень у сервісі доставки.

					3-ІПЗ-22-1	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є методи та програмні засоби розробки веб-орієнтованих систем, що застосовуються для побудови операційної платформи керування замовленнями.

Методи дослідження. У роботі застосовано методи системного аналізу для дослідження предметної галузі, методи об'єктно-орієнтованого проєктування та моделювання мовою UML, методи проєктування реляційних баз даних, а також методи веб-програмування й тестування програмного забезпечення.

Практичне значення одержаних результатів полягає в тому, що розроблений програмний комплекс може використовуватися як основа операційної системи невеликого сервісу доставки: він забезпечує оформлення замовлень оператором, контроль статусів виконання, облік робочих змін з аналітикою та призначення кур'єрів, а завдяки модульній архітектурі легко розширюється новими функціями без переписування ядра.

Структура роботи. Пояснювальна записка складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі подано огляд предметної галузі сервісів доставки, аналіз аналогів і вимоги до системи. У другому розділі описано обраний стек технологій та наведено аргументацію його вибору. У третьому розділі описано порядок користування програмою та результати тестування.

Робота спирається на чинні методичні рекомендації кафедри інженерії програмного забезпечення [1].

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Сервіси доставки як предметна галузь та бізнес-процеси діяльності

Предметною галуззю кваліфікаційної роботи є операційна діяльність сервісів доставки — підприємств та закладів, що приймають замовлення на товари або готові страви й організовують їх передавання клієнту через власних або залучених кур'єрів. До цієї галузі належать служби доставки їжі, невеликі магазини з адресною доставкою, аптеки, квіткові й продуктові сервіси, а також локальні заклади, що приймають замовлення телефоном чи в месенджерах. Спільним для них є потреба швидко зафіксувати замовлення, правильно порахувати його вартість, передати кур'єру та проконтролювати виконання.

Протягом останніх років споживча поведінка суттєво змістилася в бік цифрових каналів. Клієнт очікує, що замовлення буде прийняте за лічені хвилини, склад і сума будуть однозначними, а статус виконання — прозорим. Така зміна очікувань підвищує вимоги не лише до вітрини сервісу, а й до його внутрішньої організації: навіть невеликий заклад повинен мати інструмент, що дозволяє оператору швидко оформити замовлення, а керівнику — бачити фактичні показники роботи [14].

Особливістю предметної галузі є те, що ключову роль у ній відіграє не клієнт, а працівник сервісу. Саме оператор приймає звернення, уточнює позиції, формує замовлення й передає його в роботу, а кур'єр виконує доставку. Тому система, що автоматизує цю діяльність, є внутрішньою операційною (диспетчерською) платформою, а не публічним магазином. Це принципово впливає на склад функцій: у центрі системи перебувають інструменти оператора й кур'єра, рольовий доступ і контроль виконання, а не клієнтський кошик і самостійне оформлення замовлення.

Без спеціалізованого програмного забезпечення операційна діяльність сервісу доставки залишається трудомісткою та схильною до помилок. Оператор змушений вручну приймати дані від клієнта, уточнювати склад замовлення, рахувати суму, перевіряти наявність позицій, передавати замовлення кур'єру й окремо повідомляти клієнта про статус. За такого підходу легко припуститися

					3-ІПЗ-22-1	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

помилки в сумі, втратити замовлення, продублювати дані або втратити історію, що ускладнює подальший аналіз ефективності.

Автоматизація цих операцій дає змогу формалізувати бізнес-процес: кожне замовлення набуває чіткої структури, кожна позиція прив'язана до товару, сума обчислюється автоматично, а статус відображає реальний етап виконання. Це зменшує ризик помилок, прискорює роботу й створює основу для аналітики. Крім того, веб-орієнтований характер системи дозволяє працювати через браузер без встановлення окремого клієнтського застосунку та розгортати рішення як локально, так і в хмарному середовищі.

Отже, автоматизація операційної діяльності сервісу доставки є актуальною прикладною задачею. Розроблювана система спрямована на те, щоб замінити розрізнені ручні дії єдиним інструментом, який підтримує повний цикл керування замовленням, забезпечує контрольований доступ за ролями та надає керівнику фактичні показники роботи для ухвалення рішень.

Доцільно також окреслити типову організаційну структуру невеликого сервісу доставки, оскільки саме вона визначає коло користувачів системи. Як правило, такий сервіс налічує одного-двох операторів, що приймають замовлення, кількох кур'єрів і власника або керівника, який водночас виконує функції адміністратора. За такого складу немає сенсу впроваджувати важку корпоративну систему: потрібен компактний інструмент, що покриває щоденні операції без надмірних витрат на впровадження й обслуговування.

Ще однією характерною рисою галузі є висока ціна помилки в умовах обмеженого часу. На відміну від сфер, де дані вводяться неспішно, оператор сервісу доставки працює під час розмови з клієнтом, тому інтерфейс має бути максимально передбачуваним, а система — брати на себе рутинні перевірки й обчислення. Автоматичний розрахунок суми, контроль наявності відкритої зміни та перевірка коректності даних знімають з оператора частину розумового навантаження й зменшують імовірність помилки в стресових умовах пікового попиту.

Варто також зважати на сезонність і нерівномірність навантаження, характерні для сервісів доставки. У години пік кількість замовлень різко зростає,

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

тому швидкість оформлення стає критичною: оператор повинен внести замовлення за лічені секунди, а будь-яка затримка або помилка безпосередньо впливає на якість обслуговування. Програмна система має підтримувати такий темп роботи, мінімізуючи кількість ручних дій і автоматизуючи розрахунки, тоді як паперовий або месенджерний облік у пікові періоди стає вузьким місцем.

Ще одним важливим аспектом галузі є потреба у накопиченні та аналізі історичних даних. Власник сервісу зацікавлений не лише в оперативному прийманні замовлень, а й у розумінні того, які товари користуються попитом, у які години надходить найбільше замовлень, яка частка скасувань і яка середня вартість замовлення. Без структурованого зберігання даних така аналітика неможлива, тому система від початку має проєктуватися з урахуванням подальшого формування звітності.

Щоб повніше розкрити предметну галузь, від загальної характеристики сервісів доставки доцільно перейти до розгляду їхніх бізнес-процесів у природній послідовності. Саме формалізація цих процесів, кожен з яких має власних учасників, вхідні та вихідні дані й правила переходу до наступного етапу, є основою для подальшого визначення функцій системи.

Щоб коректно визначити функції системи, доцільно розглянути бізнес-процеси сервісу доставки в їх природній послідовності. Кожен із цих процесів має власних учасників, вхідні та вихідні дані й правила переходу до наступного етапу, тому їх формалізація є основою для подальшого проєктування.

Перший процес — підготовка та підтримання каталогу. Адміністратор додає товари, формує їх опис, визначає ціну, прив'язує до категорій, завантажує зображення та налаштовує акційні пропозиції. Якщо каталог неактуальний, оператор не може коректно сформулювати замовлення, а клієнт отримує застарілу інформацію про асортимент і ціни. Тому підтримання каталогу є базовою умовою роботи всієї системи, а не другорядною адміністративною дією.

Другий процес — відкриття робочої зміни. Перед прийманням замовлень оператор відкриває зміну, яка стає одиницею обліку та аналітики.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

Поки зміна не відкрита, створення замовлень заблоковано, що запобігає появі замовлень поза обліковим періодом. Один сервіс одночасно працює лише з однією відкритою зміною, що відповідає реальній організації роботи невеликого закладу та спрощує формування зведених показників.

Третій і центральний процес — оформлення замовлення. Оператор приймає звернення клієнта, обирає товари з каталогу, зазначає кількість, вводить контактні дані й адресу доставки, за потреби застосовує акцію, після чого система розраховує суму та створює замовлення з позиціями у початковому стані. Якщо хоча б один товар відсутній або дані некоректні, система повертає зрозумілу помилку й не зберігає неповні дані, що захищає цілісність обліку.

Четвертий процес — обробка та доставка. Замовлення послідовно проходить контрольовані статуси: підтвердження, підготовку, передавання в доставку та виконання. Відповідальний працівник призначає кур'єра, який виконує доставку, підтверджує її або позначає проблему. Контрольований перелік статусів дозволяє у будь-який момент бачити стан кожного замовлення й фільтрувати їх за етапами.

П'ятий процес — закриття зміни та формування аналітики. Після завершення роботи оператор закриває зміну, а незавершені замовлення автоматично позначаються як невиконані. За зміною формується статистика: кількість замовлень і клієнтів, виручка, середній чек, найпопулярніші товари та погодинний розподіл навантаження. Ці дані допомагають оцінювати завантаженість і ухвалювати управлінські рішення.

У наведених процесах беруть участь три типи працівників. Адміністратор відповідає за довідники й персонал, оператор — за приймання та ведення замовлень і робочих змін, кур'єр — за виконання доставки. Такий розподіл ролей прямо відображає організацію роботи сервісу й покладений в основу рольової моделі доступу, яка розглядається далі під час формування вимог.

Окремо слід виділити процес взаємодії з клієнтом, який, хоч і не передбачає його входу в систему, істотно впливає на вимоги до даних. Оператор фіксує ім'я, номер телефону та адресу клієнта, а система зберігає клієнта як контакт доставки. Завдяки цьому повторні замовлення того самого клієнта пов'язуються з єдиним

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

записом, що дозволяє згодом аналізувати кількість унікальних клієнтів за зміну та формувати базу контактів без створення повноцінних облікових записів.

Усі описані бізнес-процеси пов'язані між собою та утворюють замкнений цикл: підготовлений каталог уможливлює оформлення замовлення, відкрита зміна — його приймання, оброблене замовлення — доставку, а закрита зміна — аналітику, результати якої впливають на подальше наповнення каталогу й планування роботи. Саме цілісність цього циклу, а не окремі операції, визначає цінність системи й вимагає, щоб програмне забезпечення підтримувало переходи між етапами, а не лише зберігало розрізнені записи.

Між бізнес-процесами існують також винятки та нестандартні ситуації, які система повинна враховувати. Клієнт може скасувати замовлення вже після його оформлення, кур'єр може зіткнутися з недоступністю клієнта чи відмовою від замовлення, а товар може виявитися тимчасово відсутнім. Без програмної підтримки такі ситуації обробляються неформально й часто залишаються незадокументованими, тоді як система фіксує їх через відповідні статуси (скасовано, проблема), зберігаючи причину й роблячи ці події частиною обліку, а не випадковою усною інформацією.

Координація між учасниками процесу є ще одним аспектом, що потребує автоматизації. У ручному режимі передавання замовлення кур'єру та зворотне інформування про виконання відбуваються через телефонні дзвінки або повідомлення, що сповільнює роботу й створює ризик втрати інформації. Програмна система замінює цю взаємодію спільним станом даних: оператор бачить, що кур'єр прийняв і виконав доставку, а кур'єр — що йому призначено нове замовлення, без потреби в окремому каналі узгодження.

1.2. Задачі автоматизації та аналіз існуючих аналогів

На підставі аналізу бізнес-процесів сформовано перелік задач, що підлягають автоматизації. Ці задачі охоплюють як безпосередню роботу із замовленнями, так і допоміжні операції, без яких основний цикл не може функціонувати коректно.

До задач, пов'язаних із доступом і персоналом, належать автентифікація працівника за номером телефону й одноразовим кодом, розмежування прав за

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

ролями, ведення обліку працівників та зміна їхніх ролей адміністратором. Ці задачі забезпечують, що адміністративні дії недоступні звичайному оператору, а операції кур'єра виконуються лише кур'єром через Telegram-бот.

До задач, пов'язаних із каталогом, належать ведення товарів і категорій, керування акційними пропозиціями з відсотковою або фіксованою знижкою та мінімальною сумою замовлення, а також завантаження зображень товарів і акцій. Каталог є джерелом даних для оформлення замовлення, тому його коректність безпосередньо впливає на правильність розрахунків.

До задач, пов'язаних із замовленнями, належать відкриття та закриття робочої зміни, оформлення замовлення з кількох позицій із автоматичним розрахунком суми та знижки, контроль і зміна статусів, призначення кур'єра, а також робота кур'єра з власними доставками. Окремою задачею є формування аналітики робочих змін, що перетворює накопичені дані на показники для прийняття рішень.

Сукупність цих задач визначає мінімально достатній функціонал системи. Їх реалізація дозволяє охопити повний цикл керування замовленням — від підготовки каталогу й відкриття зміни до виконання доставки та підбиття підсумків — і саме вони покладені в основу вимог, сформульованих далі.

Реалізація перелічених задач має підпорядковуватися спільним принципам. По-перше, дані вводяться один раз і повторно використовуються на всіх етапах: введений під час оформлення склад замовлення автоматично потрапляє в обробку, доставку та аналітику без повторного введення. По-друге, кожна операція супроводжується перевіркою коректності, щоб у систему не потрапляли суперечливі дані. По-третє, доступ до операцій обмежується відповідно до ролі працівника, що убезпечує систему від помилкових або несанкціонованих дій.

Перелічені задачі різняться за частотою виконання та критичністю. Оформлення замовлення й зміна статусів виконуються десятки разів за зміну, тому до них висуваються найвищі вимоги щодо швидкості та зручності. Ведення каталогу та керування працівниками виконуються рідше, проте помилки в них впливають на всю подальшу роботу. Така класифікація задач враховується під час

					3-ІПЗ-22-1	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

проектування інтерфейсу: найчастіші дії робляться максимально швидкими, а рідші — захищеними додатковими перевітками.

Перелічені задачі можна згрупувати за рівнями системи. На рівні даних — це коректне збереження сутностей та підтримання зв'язків між ними; на рівні бізнес-логіки — виконання правил оформлення замовлення, контролю зміни та розрахунку сум; на рівні доступу — автентифікація й авторизація; на рівні подання — зручне відображення інформації працівникам. Такий поділ за рівнями узгоджується з обраною архітектурою й допомагає рівномірно розподілити функціональність між модулями системи.

Слід зауважити, що частина задач не є самостійними функціями, а виступає передумовами для інших. Наприклад, автентифікація сама собою не приносить користувачеві результату, але без неї неможливі жодні захищені операції; так само облік робочих змін не є самоціллю, проте уможлиблює аналітику та контроль періоду роботи. Розуміння цих залежностей важливе для правильного планування послідовності реалізації: базові, забезпечувальні задачі реалізуються першими, а ті, що від них залежать, — згодом.

Визначивши перелік задач, що підлягають автоматизації, доцільно зіставити їх із можливостями наявних на ринку рішень. Для цього проаналізовано типові класи програмних продуктів, які використовуються для керування замовленнями, і з'ясовано, наскільки повно кожен із них покриває описані вище задачі.

Для визначення очікуваного функціоналу доцільно проаналізувати типові рішення, що використовуються для керування замовленнями: маркетплейси та інтернет-магазини, спеціалізовані сервіси доставки їжі, CRM-системи для малого бізнесу та модулі замовлень у популярних CMS. Кожен із цих класів по-різному покриває операційний процес і має власні сильні та слабкі сторони.

Маркетплейси та великі інтернет-магазини пропонують розвинений каталог, кошик, оплату й особистий кабінет клієнта. Вони добре масштабуються, проте орієнтовані передусім на самостійне оформлення замовлення клієнтом і є надлишковими для невеликого сервісу: значна частина їхнього функціоналу

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

залишається незатребуваною, а внутрішня операційна логіка прихована й не піддається тонкому налаштуванню під конкретну організацію роботи.

Спеціалізовані сервіси доставки їжі добре демонструють сценарії вибору позицій і відстеження доставки, однак їхня архітектура закрита, а можливості адаптації під власні процеси обмежені. CRM-системи для малого бізнесу зосереджені на обліку заявок, клієнтів і статусів, що близько до потреб диспетчерської системи, але вони рідко мають вбудований товарний каталог із цінами та акціями і не завжди надають зручний інструмент швидкого формування замовлення з позицій.

Модулі замовлень у системах керування контентом дозволяють швидко запустити базовий магазин на готових шаблонах, проте їх складно адаптувати під чітко розмежовану операційну модель з ролями оператора й кур'єра та обліком робочих змін. Узагальнені результати порівняння аналогів за чотирма критеріями — повнота операційного процесу, гнучкість налаштування, придатність для внутрішньої диспетчеризації та складність впровадження — наведено в таблиці 1.1.

Таблиця 1.1

Порівняння існуючих аналогів програмних рішень

Тип рішення	Сильні сторони	Обмеження для сервісу доставки
Інтернет-магазин	Розвинений каталог, кошик, оплата, кабінет клієнта, масштабованість	Орієнтація на клієнтський checkout, надлишковість, закрита операційна логіка
Сервіс доставки їжі	Зручний вибір позицій, відстеження, готові сценарії доставки	Закрита архітектура, слабка адаптація під власну організацію роботи
CRM для малого бізнесу	Облік заявок, клієнтів і статусів, звітність	Немає товарного каталогу з акціями, складне формування замовлення з позицій

Модуль замовлень у CMS	Швидкий запуск, готові шаблони, розширення плагінами	Складно реалізувати ролі оператор/кур'єр, зміни та чисту архітектуру
Власна розробка	Повний контроль логіки, ролі, зміни, аналітика, чиста архітектура	Менший набір готових функцій; потребує самостійної реалізації

Аналіз показує, що жоден із наявних класів рішень повністю не відповідає потребам внутрішньої диспетчерської системи невеликого сервісу доставки. Маркетплейси й магазини надлишкові та закриті, CRM не має каталогу, а CMS-модулі погано підтримують потрібну операційну модель. Тому доцільною є власна розробка, що свідомо займає проміжну позицію: вона поєднує товарний каталог з акціями, характерний для магазину, і керування заявками та статусами, характерне для CRM, але з акцентом на внутрішню операційну діяльність працівників.

Відмінністю власної розробки є рольова модель з оператором і кур'єром, облік робочих змін як одиниці роботи й аналітики, а також чистий поділ між бізнес-логікою та інфраструктурою. На основі огляду аналогів встановлено, що мінімально достатній функціонал має включати автентифікацію та ролі, каталог із категоріями й акціями, оформлення замовлення з позиціями, контроль статусів, призначення кур'єра, облік змін та адміністративне керування довідниками.

Доцільно докладніше розглянути кожен клас аналогів з погляду внутрішньої операційної логіки. Маркетплейси будуються навколо вітрини й самостійного оформлення замовлення клієнтом, а робота персоналу зводиться до обробки вже створених замовлень; такий підхід не передбачає сценарію, коли саме оператор формує замовлення від імені клієнта, що є ключовим для телефонних і месенджерних звернень. Сервіси доставки їжі, навпаки, мають розвинену операційну частину, проте надаються як готова закрыта платформа, що не дозволяє

адаптувати модель ролей або облік змін під конкретний заклад.

CRM-системи орієнтовані на роботу з угодами та контактами, тому добре підтримують статуси й відповідальних, але вимагають значного доналаштування, щоб перетворити узагальнену «угоду» на замовлення з товарними позиціями, цінами та знижками. Модулі замовлень у CMS успадковують архітектуру платформи, у якій вони працюють, тому реалізація чистого поділу домену й інфраструктури або власної рольової моделі в них суперечить шаблонам платформи й ускладнює супровід.

Проведений аналіз дозволяє сформулювати перелік ознак, які власна система має успадкувати від найкращих рішень і водночас уникнути їхніх недоліків. Від магазинів варто взяти структурований каталог з акціями, від сервісів доставки — зручні сценарії роботи з позиціями та відстеження, від CRM — облік статусів і відповідальних. Уникнути ж слід надлишковості, закритості архітектури та жорсткої прив'язки до клієнтського checkout, що й визначає проміжну, спеціалізовану нішу власної розробки.

1.3. Вимоги до системи та обґрунтування архітектурного підходу

На основі аналізу предметної галузі та аналогів сформульовано вимоги до системи. Відповідно до рекомендацій щодо оформлення, вимоги подано зв'язним текстом та у вигляді діаграми варіантів використання, а не маркованими переліками. Узагальнену діаграму варіантів використання наведено на рис. 1.1; вона відображає основних акторів і дії, які кожен з них може виконувати.



Рис. 1.1 – Діаграма варіантів використання системи

Функціональні вимоги визначають дії, які система повинна виконувати. Адміністратор має повний доступ: він керує товарним каталогом, категоріями та акціями, веде облік працівників і змінює їхні ролі, переглядає зведену статистику та має доступ до всіх замовлень. Оператор виконує операційну роботу: відкриває та закриває робочу зміну, оформлює замовлення з позицій каталогу, змінює їхні статуси й призначає кур'єра, але не керує працівниками. Кур'єр взаємодіє із системою через Telegram-бот: отримує призначені йому доставки, починає доставку, підтверджує її виконання або позначає проблему, не використовуючи вебінтерфейс.

Окремо до функціональних вимог належать автентифікація працівника за одноразовим кодом і видача токена доступу, автоматичний розрахунок суми замовлення з урахуванням акції, контроль єдиної відкритої зміни, автоматичне закриття простроченої зміни та формування аналітики за зміною. Ці вимоги безпосередньо впливають із бізнес-процесів, описаних вище, і визначають поведінку системи в типових сценаріях роботи.

Нефункціональні вимоги стосуються якостей системи в цілому. До них належать модульність і розширюваність архітектури, що дозволяють додавати нові функції без переписування ядра; захищеність доступу до API, за якої адміністративні дії недоступні без відповідної ролі; цілісність даних на рівні бази, що гарантується обмеженнями та зв'язками; повторюваність середовища запуску завдяки контейнеризації; а також зрозумілість і чутливість інтерфейсу для працівників.

Відповідність системи цим вимогам оцінюється з опорою на загальновизнану модель характеристик якості програмного забезпечення, яка виділяє функціональну придатність, надійність, зручність використання, захищеність і супроводжуваність [30]. Сформульовані вимоги є критеріями, за якими далі перевіряється повнота реалізації, і безпосередньо впливають на вибір технологічного стеку.

Поряд із функціональними та нефункціональними вимогами доцільно визначити обмеження, у межах яких розробляється система. До них належать

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

обмежений склад персоналу сервісу, відсутність потреби в самостійному оформленні замовлення клієнтом, орієнтація на роботу через браузер без встановлення клієнтського застосунку, а також залежність окремих функцій від зовнішніх сервісів (надсилання SMS, зберігання зображень, сповіщення месенджером). Ці обмеження не звужують цінності системи, а навпаки, допомагають сфокусувати її на реальних потребах і уникнути надлишкової функціональності.

Важливо також розрізняти вимоги поточної версії та перспективні можливості. У межах кваліфікаційної роботи реалізується ядро системи, достатнє для повного циклу керування замовленням, тоді як низку можливостей (розширена маршрутизація кур'єрів, інтеграція з оплатою, сповіщення клієнтів) свідомо винесено в перспективу. Таке розмежування відповідає принципу поступової розробки й гарантує, що опис системи в подальших розділах стосується саме реалізованого функціоналу, а не задекларованих намірів.

Сформульовані вимоги доцільно розглядати у взаємозв'язку, оскільки вони впливають одна на одну. Так, вимога захищеності доступу за ролями безпосередньо впливає з функціонального розмежування адміністратора, оператора й кур'єра, а вимога цілісності даних підкріплює коректність розрахунку суми замовлення. Нехтування будь-якою з нефункціональних вимог знижує практичну цінність системи: навіть за повного набору функцій незахищена або ненадійна система непридатна до реального використання.

Особливу увагу приділено вимозі щодо зручності використання, оскільки безпосередніми користувачами системи є працівники сервісу, які виконують типові дії багаторазово протягом зміни. Інтерфейс має мінімізувати кількість кроків для оформлення замовлення, наочно показувати стан замовлень і робочої зміни та запобігати помилковим діям, приховуючи недоступні операції. Ці міркування враховано під час проєктування інтерфейсу, описаного в третьому розділі.

Сформульовані функціональні та нефункціональні вимоги безпосередньо впливають на спосіб внутрішньої організації системи, тому завершальним питанням огляду предметної галузі є вибір архітектурного підходу. Нижче

					3-ІПЗ-22-1	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

розглянуто поширені архітектурні варіанти та обґрунтовано той із них, що найкраще відповідає визначеним вимогам і масштабу роботи.

Завершальним питанням огляду предметної галузі є вибір архітектурного підходу до побудови системи. Розглянуто три поширені варіанти: класичну шарову архітектуру, гексагональну архітектуру (порти й адаптери) та мікросервісну архітектуру. Кожен із них має свою сферу доцільного застосування, тому вибір ґрунтується на масштабі системи та вимогах кваліфікаційної роботи.

Класична шарова архітектура передбачає, що контролери звертаються до сервісів, сервіси — до репозиторіїв, а всі класи перебувають в одному застосунку. Вона проста для невеликого проєкту, проте швидко призводить до залежності бізнес-логіки від інфраструктурних деталей: сервіс безпосередньо залежить від конкретного репозиторію, а зміна сховища чи веб-фреймворку зачіпає ядро системи.

Мікросервісна архітектура поділяє систему на незалежні сервіси, що взаємодіють мережею. Вона забезпечує гнучке масштабування та незалежне розгортання, однак для бакалаврської роботи й поточного масштабу системи є надлишковою: вона потребує окремого розгортання, моніторингу, обробки мережеских відмов і суттєво ускладнює запуск та демонстрацію.

Гексагональна архітектура є компромісом між простотою та якістю структури. Її головна ідея полягає у відокремленні доменного ядра від зовнішніх механізмів — бази даних, контролерів, засобів безпеки та конфігурації — через порти. Домен не залежить від інфраструктури, а інфраструктура підключається до нього через адаптери. Система при цьому залишається монолітною для запуску, але внутрішньо має чіткий поділ відповідальностей, що полегшує тестування та розвиток.

Для розроблюваної системи обрано саме гексагональну архітектуру. Вона дозволяє не змішувати бізнес-логіку доставки з технічними деталями веб-фреймворку чи ORM, спрощує заміну сховища або каналу сповіщень і відповідає вимозі модульності. Детальний опис реалізації цього підходу засобами обраного

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

стеку наведено у другому розділі, де також обґрунтовано вибір конкретних технологій.

Вибір на користь гексагональної архітектури додатково підкріплюється міркуваннями щодо вартості подальших змін. У шаровій архітектурі бізнес-логіка з часом неминуче «прилипає» до інфраструктурних класів, тому навіть незначна заміна — інший механізм збереження, інша бібліотека веб-рівня чи новий канал сповіщень — потребує втручання в ядро. У гексагональній моделі ці зміни локалізуються в адаптерах, а доменні правила залишаються незмінними, що знижує ризик регресій і вартість супроводу впродовж усього життєвого циклу системи [32].

Важливим є й те, що обраний підхід не суперечить можливому майбутньому переходу до розподіленої архітектури. Якщо з часом окремі частини системи доведеться виділити в самостійні сервіси, чіткі межі портів стануть природними точками такого поділу: адаптер, що нині звертається до локального репозиторію, може бути замінений на адаптер віддаленого виклику без зміни доменної логіки. Таким чином, монолітна за способом запуску система зберігає потенціал до еволюційного масштабування, не сплачуючи наперед його організаційну й експлуатаційну ціну [34].

Слід також наголосити, що гексагональна архітектура є не самоціллю, а засобом досягнення сформульованих нефункціональних вимог. Модульність і розширюваність забезпечуються поділом на домен та адаптери, тестованість — можливістю підставляти тестові реалізації портів, а захищеність і цілісність — зосередженням бізнес-правил у єдиному ядрі, через яке проходять усі операції. Саме узгодженість архітектурного рішення з вимогами, а не його теоретична досконалість, є вирішальним аргументом на користь обраного підходу.

Підсумовуючи огляд архітектурних підходів, варто наголосити, що обраний підхід впливає не лише на внутрішню організацію коду, а й на весь життєвий цикл системи. Чіткий поділ на домен та інфраструктуру полегшує супровід, оскільки локалізує зміни: правка бізнес-правила зачіпає лише доменний модуль, зміна сховища — лише модуль збереження, а налаштування веб-доступу — лише веб-

					3-ІПЗ-22-1	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

модуль. Це знижує ризик випадково порушити несуміжну частину системи й робить проєкт зрозумілішим для перевірки та подальшого розвитку.

Таким чином, результати огляду предметної галузі формують цілісну картину: визначено характер діяльності сервісів доставки та проблеми її ручного ведення, описано бізнес-процеси й ролі, сформовано перелік задач автоматизації, проаналізовано аналоги та обрано архітектурний підхід. Ці результати є вхідними даними для обґрунтування технологічного стеку, який має забезпечити реалізацію сформульованих функціональних і нефункціональних вимог.

Висновки до розділу 1

У першому розділі виконано огляд предметної галузі сервісів доставки, обґрунтовано актуальність автоматизації операційної діяльності та визначено перелік задач, що підлягають автоматизації. Розглянуто бізнес-процеси сервісу та ролі його працівників, проаналізовано існуючі аналоги й показано, що жоден з них повністю не відповідає потребам внутрішньої диспетчерської системи.

Сформульовано функціональні та нефункціональні вимоги до системи, подані діаграмою варіантів використання та зв'язним текстом, а також обрано гексагональну архітектуру як найбільш доцільну для обраного масштабу. Отримані результати є основою для обґрунтування технологічного стеку, що розглядається у другому розділі.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

РОЗДІЛ 2. ОПИС СТЕКУ ТЕХНОЛОГІЙ ТА АРГУМЕНТАЦІЯ ВИБОРУ

2.1. Загальна характеристика стеку та платформа Java і Spring Boot

Другий розділ присвячено опису технологічного стеку та аргументації його вибору. Для системи управління замовленнями стек має одночасно вирішувати кілька задач: забезпечувати надійний серверний застосунок, підтримувати модульну архітектуру, надавати засоби безпеки, стабільно працювати з реляційною базою даних, бути зручним для тестування й дозволяти розвивати клієнтський інтерфейс. Тому вибір технологій розглядається не як випадковий набір бібліотек, а як узгоджена сукупність, у якій кожен компонент має чітке місце в архітектурі.

У роботі використано мову Java 17, фреймворк Spring Boot із модулями Spring MVC, Spring Security та Spring Data JPA, бібліотеку об'єктно-реляційного відображення Hibernate, реляційну СКБД PostgreSQL, контейнеризацію через Docker Compose, систему збирання Apache Maven, токенну автентифікацію на основі JWT, хмарне сховище зображень Cloudinary, а також односторінковий клієнт на чистому JavaScript. Загальну взаємодію цих компонентів показано на рис. 2.1.

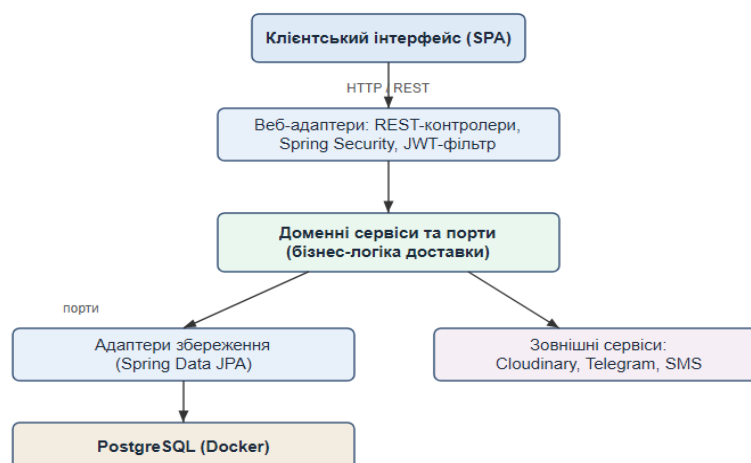


Рис. 2.1 – Узагальнена структура та технологічний стек системи

Стек відповідає характеру внутрішньої операційної системи доставки: серверна частина обробляє бізнес-правила, база даних зберігає структуровані об'єкти, веб-модуль приймає HTTP-запити, засоби безпеки контролюють доступ, а

односторінковий клієнт надає працівникам зручний інтерфейс. Ключовою вимогою до стеку є підтримка гексагональної архітектури — технології не повинні змушувати доменну логіку залежати від контролерів чи репозиторіїв.

Додатковою вимогою є актуальність технологій: усі компоненти обрано в сучасних версіях, а їх вибір обґрунтовано чинною офіційною документацією, що зменшує ризик застарілих рішень і відповідає вимозі до сучасності джерел [2]–[9]. У наступних підрозділах кожен елемент стеку розглянуто окремо з поясненням, яку саме задачу він вирішує в межах системи доставки.

Принциповим критерієм відбору технологій була їхня взаємна сумісність та наявність єдиної зрілої екосистеми. Замість того щоб поєднувати розрізнені бібліотеки різних постачальників, перевагу віддано платформі, у межах якої веб-рівень, доступ до даних, безпека й тестування інтегровані між собою та мають узгоджену конфігурацію. Це зменшує кількість потенційних точок несумісності, спрощує налаштування й знижує поріг входження для подальшого супроводу системи іншими розробниками.

Другим критерієм була підтримка довгострокового розвитку. Для навчально-інженерного проєкту, який може розвиватися й після захисту, важливо, щоб обрані технології мали тривалу підтримку, велику спільноту та докладну документацію. Це гарантує, що знайдені рішення залишатимуться актуальними, а проблеми, які виникають під час розробки, мають описані способи розв’язання у відкритих джерелах.

Принциповим для подальшого викладу є розуміння того, що кожен компонент стеку закріплено за конкретним архітектурним рівнем. Клієнтський рівень відповідає за подання та взаємодію з працівником, веб-рівень (Spring MVC та засоби безпеки) — за приймання й авторизацію запитів, доменний рівень (доменні сервіси та порти) — за виконання бізнес-правил, а рівень збереження (Spring Data JPA, Hibernate, PostgreSQL) — за надійне зберігання даних. Такий розподіл відповідальностей запобігає змішуванню логіки різних рівнів і робить систему передбачуванішою для аналізу та супроводу.

					3-ІПЗ-22-1	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

Окремо варто підкреслити перевагу єдиної мови та середовища виконання для всієї серверної частини. Оскільки доменна логіка, доступ до даних, безпека й інтеграції написані однією мовою та працюють у межах однієї віртуальної машини Java, зникає потреба в проміжних перетвореннях даних між різнорідними платформами, спрощується налагодження й узгоджується обробка помилок. Це зменшує кількість «стиків» між технологіями, кожен з яких є потенційним джерелом дефектів.

Нарешті, обраний стек свідомо орієнтований на просте розгортання та демонстрацію. Серверний застосунок збирається в єдиний виконуваний артефакт із вбудованим веб-сервером, а зовнішня залежність — база даних — піднімається однією командою через контейнер. Завдяки цьому весь програмний комплекс може бути запущений на типовому комп'ютері без складного попереднього налаштування, що є важливою практичною вимогою до навчально-інженерного проєкту та умовою його успішного захисту.

Окресливши склад технологічного стеку в цілому, розглянемо детальніше його фундамент — мову програмування та серверну платформу, на яких ґрунтується вся серверна частина системи.

Основною мовою реалізації серверної частини обрано Java 17 — версію платформи з тривалою підтримкою (LTS), яку супроводжує зріла екосистема Spring [23]. Java має статичну типізацію, розвинені засоби об'єктно-орієнтованого програмування та широку інфраструктуру бібліотек для безпеки, доступу до даних і тестування, що робить її доцільною для систем із тривалим життєвим циклом.

Для кваліфікаційної роботи Java зручна тим, що дозволяє явно показати об'єктно-орієнтоване моделювання предметної галузі: працівники, клієнти, товари, категорії, акції, замовлення, позиції, зміни, ролі та статуси описуються окремими класами або переліченнями. Строга типізація допомагає виявляти помилки ще на етапі компіляції, а не під час виконання, що підвищує надійність коду.

Серверний застосунок побудовано на основі Spring Boot, який автоматизує значну частину конфігурації, надає вбудований веб-сервер, механізм профілів,

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

підтримку зовнішньої конфігурації та інтеграцію з компонентами екосистеми. Завдяки цьому розробник може зосередитися на бізнес-логіці, а не на ручному налаштуванні кожного інфраструктурного компонента. Офіційна документація Spring Boot прямо орієнтована на створення придатних до промислового використання застосунків [2], [3].

Веб-рівень реалізовано засобами Spring MVC. Контролери приймають HTTP-запити, отримують об'єкти передачі даних, викликають вхідні порти доменного модуля й повертають відповіді. Завдяки цьому контролери не змішуються з доменними моделями й не перетворюються на місце зберігання бізнес-правил, а виконують роль тонких адаптерів, що перетворюють зовнішній запит на виклик доменного сервісу (приклад контролера замовлень наведено в додатку Е, адміністративні операції — у додатку І).

Допоміжним інструментом є бібліотека Lombok, що зменшує обсяг шаблонного коду в моделях і об'єктах передачі даних, генеруючи методи доступу та конструктори. Розробку проєкту виконано в інтегрованому середовищі розробки IntelliJ IDEA, яке надає засоби рефакторингу, налагодження та інтеграцію зі збиранням Maven і системою контролю версій [28]. Вона не є основою архітектури, а лише прискорює розробку; основну цінність системи становить правильний поділ на модулі, сервіси, порти та адаптери, який забезпечується саме платформою Spring.

Окремо слід відзначити роль контейнера інверсії керування (IoC), що є ядром платформи Spring. Замість того щоб класи самостійно створювали свої залежності, контейнер впроваджує їх ззовні через конструктори, що робить компоненти слабо зв'язаними та зручними для тестування: під час перевірки доменного сервісу замість реальних адаптерів легко підставляються їхні тестові реалізації. Саме механізм впровадження залежностей дозволяє технічно реалізувати інверсію залежностей, на якій ґрунтується гексагональна архітектура [35].

Платформа надає також єдиний механізм керування транзакціями через декларативну анотацію, завдяки якій бізнес-операція виконується атомарно: або всі зміни фіксуються в базі, або жодна. Це критично для системи з фінансовими

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

розрахунками, де часткове збереження замовлення призвело б до неузгодженості сум. Декларативний підхід звільняє доменний код від ручного керування з'єднаннями та межами транзакцій і зменшує ймовірність помилок у цій тонкій ділянці.

Статична типізація мови Java та суворі дисципліни іменування, рекомендовані для промислового коду, додатково підвищують надійність системи. Велику частину помилок — від невідповідності типів до звернення до неіснуючих полів — компілятор виявляє ще до запуску, а виразні назви класів і методів роблять код зрозумілим без додаткових пояснень. Дотримання усталених практик мови та платформи робить кодову базу придатною до тривалого супроводу й колективної розробки [31].

Механізм зовнішньої конфігурації та профілів Spring Boot дозволяє відокремити параметри середовища від коду: адреси бази даних, секрети підпису токенів, ключі доступу до зовнішніх сервісів задаються через конфігураційні файли та змінні середовища й відрізняються для середовищ розробки та промислової експлуатації. Завдяки цьому той самий артефакт застосунку запускається в різних середовищах без перекомпіляції, що відповідає сучасним підходам до розгортання програмного забезпечення.

Отже, поєднання Java та Spring Boot формує надійну основу серверної частини. Воно дозволяє реалізувати типізовану доменну модель, тонкий веб-рівень і автоматизовану конфігурацію, а широка підтримка цих технологій спрощує подальший супровід і розвиток системи.

2.2. Архітектура застосунку та організація доступу до даних

Для збирання проєкту використано систему Apache Maven із багатомодульною структурою. Кореневий модуль є агрегатором: він визначає склад реактора та версії залежностей, але не містить бізнес-логіки. Реактор Maven автоматично визначає порядок компіляції залежних модулів, що спрощує збирання та запуск тестів [6]. Така структура дозволяє показати модульність не лише на схемі, а й у фактичній організації проєкту.

					3-ІПЗ-22-1	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

Межі модулів збігаються з архітектурними межами системи. Доменний модуль містить бізнес-логіку, інфраструктурні модулі — збереження даних і веб-доступ, а головний модуль збирає все разом і слугує точкою запуску. Відповідальність кожного модуля узагальнено в таблиці 2.1.

Таблиця 2.1

Модульна структура програмного комплексу

Модуль	Відповідальність	Основний вміст
domain	Бізнес-логіка	Моделі, DTO, перетворювачі, сервіси, вхідні та вихідні порти
infra/postgres	Збереження даних	Spring Data JPA репозиторії та реалізація портів збереження
infra/web	Веб-доступ і безпека	REST-контролери, Spring Security, JWT, конфігурація, інтеграції
infra/app	Головний застосунок	Точка входу, application.yml, міграції Flyway, тести, збірка

Доменний модуль є центром системи. У ньому розміщено моделі предметної галузі, об'єкти передачі даних, перетворювачі, доменні сервіси та порти. Вхідні порти описують операції, доступні для веб-рівня (робота з товарами, категоріями, акціями, замовленнями й змінами), а вихідні порти описують, що домену потрібно від інфраструктури (збереження сутностей, сповіщення кур'єрів). Інфраструктурний модуль збереження реалізує вихідні порти засобами Spring Data JPA, тому доменна логіка не залежить від конкретного механізму збереження.

Перевагою такого поділу є можливість замінити або розширити будь-який адаптер без зміни ядра. Наприклад, у разі появи іншого клієнта (мобільного застосунку або зовнішнього API) доменний модуль не потребуватиме перепроєктування; так само канал сповіщення кур'єрів реалізовано як окремий вихідний порт, який можна замінити без втручання в логіку замовлень. Це безпосередньо відповідає нефункціональним вимогам щодо модульності та розширюваності, сформульованим у першому розділі.

Об'єкти передачі даних і перетворювачі формують межу між внутрішніми моделями та зовнішнім представленням. Запити містять лише потрібні поля з

анотаціями валідації, а відповіді не розкривають внутрішню структуру сутностей бази даних. Завдяки цьому клієнтська частина працює з прогнозованими структурами, а зміна внутрішньої моделі не порушує контракт API.

Багатомодульна організація має й практичні переваги під час командної роботи та супроводу. Межі модулів водночас є межами відповідальності: зміни в одному модулі мінімально зачіпають інші, а залежності між модулями описані явно у файлах збирання. Це полегшує розуміння проєкту новим розробником, який може вивчати систему помодульно, та спрощує повторне використання доменного модуля, що не містить інфраструктурних залежностей, у потенційних суміжних застосунках.

Гексагональна архітектура реалізується не лише поділом на модулі, а й напрямком залежностей між ними. Доменний модуль не залежить від жодного інфраструктурного модуля, тоді як інфраструктурні модулі залежать від домену, реалізуючи оголошені в ньому порти. Такий напрямок залежностей (інверсія залежностей) гарантує, що зміни в інфраструктурі не поширюються на ядро, і саме він є технічною основою для незалежності бізнес-логіки від конкретних технологій збереження чи доставки повідомлень.

Практичним наслідком цього є можливість тестувати доменну логіку ізольовано, підставляючи замість реальних репозиторіїв їхні тестові реалізації, а також спрощене внесення змін: додавання нового способу збереження або нового каналу сповіщень зводиться до створення ще одного адаптера, що реалізує наявний порт. Це безпосередньо підтримує нефункціональні вимоги щодо тестованості та розширюваності, які були сформульовані під час аналізу предметної галузі.

Узагальнено внутрішню будову доменного ядра системи у вигляді діаграми класів, наведеної на рис. 2.2. Діаграма відображає основні класи предметної галузі, їхні ключові атрибути та зв'язки між ними, а також перелічення (enum), якими описано скінченні множини станів і ролей. Саме ці класи реалізують бізнес-логіку доставки й не залежать від інфраструктурних механізмів, що відповідає принципам чистої архітектури [32], [36].

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

Діаграма класів доменної моделі системи

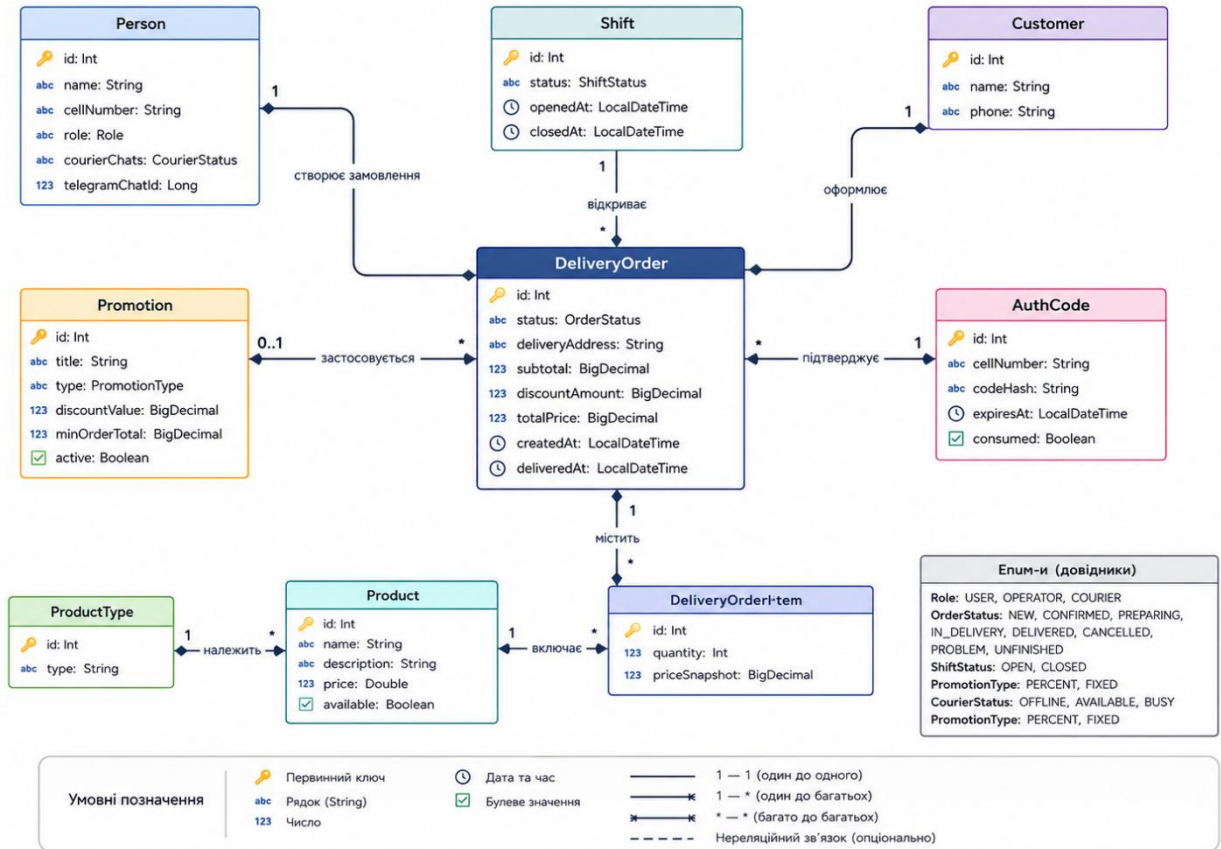


Рис. 2.2 – Діаграма класів доменної моделі системи

Центральним класом моделі є замовлення (DeliveryOrder), яке агрегує позиції замовлення (DeliveryOrderItem) у відношенні композиції «один-до-багатьох»: позиції не існують поза замовленням і видаляються разом із ним. Кожна позиція посилається на товар (Product), а товар належить до категорії (ProductType). Замовлення також пов'язане відношеннями «багато-до-одного» з клієнтом (Customer), працівником-оператором та кур'єром (обидва — екземпляри Person), робочою зміною (Shift) і, за наявності, акцією (Promotion). Така структура зв'язків безпосередньо відображає правила предметної галузі та узгоджується зі схемою бази даних, розглянутою далі.

Окремо на діаграмі винесено перелічення, які роблять модель самодокументованою та типобезпечною: роль працівника (Role), статус замовлення (OrderStatus), статус зміни (ShiftStatus), тип акції (PromotionType) та статус кур'єра (CourierStatus). Використання перелічень замість довільних рядків унеможливорює появу некоректних значень станів і ролей ще на етапі компіляції, а

клас одноразового коду автентифікації (AuthCode) зберігає лише геш коду та строк його дії, не розкриваючи самого секрету. Підхід до проєктування моделі через виразні типи й чіткі зв'язки відповідає рекомендаціям щодо створення надійних предметних моделей [31], [34].

Описана багатомодульна структура визначає не лише організацію коду, а й спосіб роботи з даними: інфраструктурний модуль збереження реалізує оголошені доменом порти. Тому далі розглянуто підсистему доступу до даних — обрану систему керування базами даних та засоби об'єктно-реляційного відображення, через які домен взаємодіє зі сховищем.

Системою керування базами даних обрано PostgreSQL — надійну реляційну СКБД із підтримкою транзакцій, зовнішніх ключів, індексів та обмежень цілісності [7], [21]. Реляційна модель природна для сервісу доставки, оскільки більшість об'єктів мають чіткі зв'язки: замовлення містить позиції, позиції посилаються на товари, товари належать до категорій, а замовлення прив'язане до клієнта, працівника, кур'єра, акції та зміни. Концептуальну схему бази даних наведено на рис. 2.3.

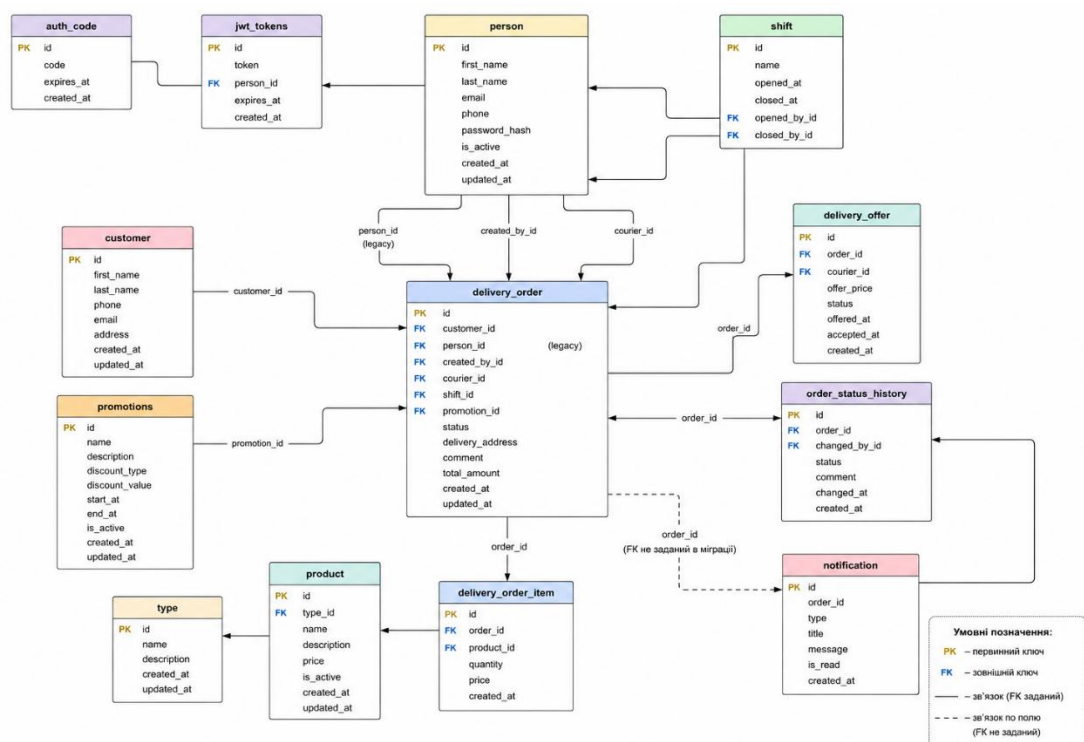


Рис. 2.3 – Схема бази даних системи

Зв'язки між сутностями відображаються анотаціями: у замовленні посилання на клієнта, працівника, кур'єра й зміну позначено зв'язком «багато-до-одного», а позиції — зв'язком «один-до-багатьох» із каскадним збереженням.

Доступ до даних реалізовано засобами Spring Data JPA у поєднанні з Hibernate. Цей підхід дозволяє описувати репозиторії як інтерфейси та працювати з доменними сутностями, не пишучи велику кількість шаблонного SQL-коду [5], [20].

Цілісність даних забезпечується на рівні самої бази. Для ключових полів визначено обмеження унікальності (зокрема, для номерів телефонів працівника та клієнта) і обов'язковості, а для кількості позицій застосовано перевіірочні обмеження, що гарантують додатне значення. Статуси й ролі зберігаються як рядкові значення відповідних перелічень, а для прискорення типових запитів — вибірки замовлень за зміною, статусом, кур'єром або датою — створено індекси.

Структуру бази даних створюють і супроводжують міграції Flyway — версіоновані SQL-сценарії, що застосовуються автоматично під час запуску застосунку. Це робить схему відтворюваною на чистому середовищі та узгодженою з кодом. Фрагмент сценарію створення таблиці замовлення наведено в лістингу 2.1, а фрагмент початкової схеми з основними таблицями — у додатку Н.

Лістинг 2.1 – Фрагмент міграції створення таблиці замовлення

```
CREATE TABLE delivery_order (
  id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  customer_id integer REFERENCES customer(id),
  created_by_id integer REFERENCES person(id),
  courier_id integer REFERENCES person(id),
  shift_id integer REFERENCES shift(id),
  promotion_id integer REFERENCES promotions(id),
  status varchar(255) NOT NULL DEFAULT 'NEW',
  delivery_address varchar(255) NOT NULL,
  subtotal numeric(10, 2),
  discount_amount numeric(10, 2) NOT NULL DEFAULT 0,
  total_price numeric(38, 2) NOT NULL DEFAULT 0,
  created_at timestamp(6) DEFAULT CURRENT_TIMESTAMP
);
```

Важливо, що використання Spring Data JPA не порушує гексагональної архітектури: репозиторії розташовані в інфраструктурному модулі та реалізують вихідні порти, а доменні сервіси звертаються саме до портів. Це відрізняє систему

від типового навчального CRUD-проєкту, у якому сервіс напряду залежав би від репозиторію, і робить бізнес-логіку незалежною від механізму збереження.

Доступ до бази даних організовано через пул з'єднань, який у Spring Boot за замовчуванням реалізовано бібліотекою HikariCP. Пул багаторазово використовує вже відкриті з'єднання замість того, щоб встановлювати нове на кожен запит, що суттєво зменшує накладні витрати під час пікового навантаження, характерного для сервісу доставки. Параметри пулу винесено в зовнішню конфігурацію, тому їх можна узгодити з обчислювальними ресурсами конкретного середовища без зміни коду.

Особливу увагу приділено керованому завантаженню пов'язаних даних. Зв'язки між сутностями за замовчуванням завантажуються відкладено, а на межі системи дані передаються не у вигляді сутностей бази, а через об'єкти передачі даних, які містять лише потрібні поля. Такий підхід запобігає типовій для ORM проблемі надмірної кількості запитів та неконтрольованого розгортання графа об'єктів, а також не дозволяє внутрішній структурі бази «протекти» у відповіді інтерфейсу [33].

Цілісність даних забезпечується спільно на рівні моделі та на рівні бази. У моделі зв'язки описано анотаціями об'єктно-реляційного відображення, а в базі ті самі правила підкріплено обмеженнями зовнішніх ключів, унікальності та перевірочними обмеженнями. Таке дублювання не є надлишковим: навіть якщо помилковий код спробує зберегти суперечливі дані, остаточним гарантом узгодженості залишається сама база даних, яка відхилить некоректну операцію.

Окремо слід наголосити на ролі грошових типів фіксованої точності. Усі суми — вартість позиції, проміжна сума, знижка та підсумок — зберігаються й обчислюються типом довільної точності з явним правилом округлення до копійок, що виключає похибки, властиві числам з плаваючою комою. Завдяки знімку ціни товару в момент оформлення замовлення подальша зміна ціни в каталозі не впливає на вже створені замовлення, що є важливою вимогою до достовірності фінансового обліку.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

2.3. Контейнеризація бази даних та підсистема безпеки

Локальне середовище бази даних розгортається через Docker Compose, що описує інфраструктуру як конфігурацію, а не як набір ручних кроків [8], [9], [26]. У файлі опису визначено сервіс PostgreSQL із базою, користувачем, паролем, томом для збереження даних і перевіркою готовності. Для уникнення конфлікту зі стандартним портом 5432, на якому може працювати інша інсталяція, використано локальний порт 55432.

Лістинг 2.2 – Опис сервісу бази даних (docker-compose.yml)

```
services:
  postgres:
    image: postgres:18.4-bookworm
    environment:
      POSTGRES_DB: delivery_store
      POSTGRES_USER: delivery_user
      POSTGRES_PASSWORD: delivery_password
    ports:
      - "55432:5432"
    volumes:
      - delivery-store-postgres-data:/var/lib/postgresql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U delivery_user -d delivery_store"]
      interval: 10s
      timeout: 5s
      retries: 5
```

Контейнеризацію обрано не лише як зручний спосіб запуску, а й як елемент якості: середовище бази даних не залежить від стану конкретного комп'ютера, і будь-який розробник або викладач може запустити його за тією самою інструкцією. Параметри підключення винесено в зовнішню конфігурацію та дубльовано змінними середовища, що дозволяє не закріплювати паролі й адреси безпосередньо в коді.

Такий підхід забезпечує повторюваність середовища — одну з нефункціональних вимог до системи. У перспективі конфігурацію Docker Compose можна розширити додатковими сервісами, наприклад кешем або тестовим контейнером бази даних для інтеграційних перевірок [22], не змінюючи логіку застосунку.

Окремої уваги заслуговує спосіб опису структури даних у міграціях. Зберігання схеми у вигляді послідовності версіонованих сценаріїв робить її частиною кодової бази: будь-яка зміна структури оформлюється як новий сценарій,

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

що застосовується автоматично й однаково в усіх середовищах. Це усуває розбіжності між схемою на машині розробника та на сервері, які часто виникають за ручного адміністрування бази, і забезпечує відтворюваність, заявлену в нефункціональних вимогах.

Числові поля грошових сум описано типом фіксованої точності, що є принциповим для фінансових розрахунків: він виключає похибки округлення, властиві числам з плаваючою комою. Відповідно й у коді суми обчислюються типом довільної точності з явним правилом округлення, завдяки чому підсумкова вартість замовлення, проміжна сума та знижка завжди узгоджені між собою та з даними, збереженими в базі.

Опис інфраструктури у вигляді конфігурації, а не послідовності ручних дій, дає кілька практичних переваг. По-перше, перевірка готовності бази даних (healthcheck) дозволяє застосунку коректно дочікуватися моменту, коли СКБД справді готова приймати з'єднання, а не лише запущена як процес. По-друге, іменований том для збереження даних відокремлює життєвий цикл даних від життєвого циклу контейнера: контейнер можна перестворити, оновити чи видалити, не втрачаючи накопиченого вмісту бази.

Контейнеризація також забезпечує паритет середовищ розробки та експлуатації. Та сама версія образу PostgreSQL із тими самими параметрами запускається й на машині розробника, і на сервері, що усуває клас помилок, які виникають через розбіжності у версіях чи налаштуваннях СКБД. Це безпосередньо підтримує нефункціональну вимогу повторюваності середовища та узгоджується із сучасними практиками хмарно-орієнтованої розробки [26].

Винесення параметрів підключення та секретів у змінні середовища робить конфігурацію гнучкою й безпечнішою: чутливі дані не зберігаються у вихідному коді, а адреси й облікові дані легко змінюються під конкретне середовище. У перспективі описаний підхід дозволяє без зміни логіки застосунку доповнити конфігурацію додатковими сервісами — наприклад, окремим контейнером бази даних для інтеграційних тестів або кеш-сервером, — що робить інфраструктуру розширюваною [22].

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

Розгорнуте у контейнері середовище бази даних потребує надійного захисту доступу до інформації, яка в ньому зберігається. Тому наступним елементом стеку, що безпосередньо доповнює інфраструктуру, є підсистема безпеки, яка контролює автентифікацію працівників і розмежування їхніх прав.

Безпеку системи реалізовано засобами Spring Security, який забезпечує фільтрацію HTTP-запитів, автентифікацію та авторизацію [4]. Застосунок налаштовано на роботу без серверних сесій: стан автентифікації передається токеном, форму входу та захист CSRF вимкнено, оскільки використовується токена модель, а доступ до більшості ендпоінтів вимагає автентифікації. Обмеження прав на рівні методів контролерів реалізовано анотацією @PreAuthorize, що централізовано визначає, хто має право виконувати кожну операцію.

Автентифікація працівника виконується за номером телефону й одноразовим кодом (ОТР). Працівник запитує код, який генерується криптографічно стійким генератором, зберігається в базі лише у вигляді геш-значення, обчисленого алгоритмом BCrypt, і має обмежений строк дії. Під час перевірки система знаходить останній невикористаний код, контролює строк його дії, звіряє введені значення та, у разі успіху, видає токен доступу JWT і токен оновлення (лістинг 2.3).

Лістинг 2.3 – Перевірка одноразового коду та видача токена

```
@Transactional
public Map<String, Object> verifyCode(String phone, String code, HttpServletRequest
request) {
    Person person = findActivePerson(phone);
    AuthCode authCode = authCodeRepository

.findFirstByCellNumberAndConsumedFalseOrderByCreatedAtDesc(person.getCellNumber())
    .orElseThrow(() -> new AccessDeniedException("Code not found"));
    if (authCode.getExpiresAt().isBefore(LocalDate.now()))
        throw new AccessDeniedException("Code expired");
    if (!passwordEncoder.matches(code, authCode.getCodeHash()))
        throw new AccessDeniedException("Invalid code");
    authCode.setConsumed(true);
    authCodeRepository.save(authCode);
    PersonalDetails details = new PersonalDetails(person);
    String accessToken = jwtService.generateJwtAccessToken(details);
    jwtService.generateJwtRefreshToken(details, request);
    return Map.of("accessToken", accessToken, "role", person.getRole().name());
}
```

Токен доступу JWT передає інформацію про автентифікованого працівника між клієнтом і сервером, а токен оновлення дозволяє продовжити доступ без повторного входу. Таке рішення зручне для системи з REST-API, оскільки одна й

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		37

та сама серверна логіка може обслуговувати браузерний інтерфейс і потенційний мобільний клієнт. Зберігання секретів у зовнішній конфігурації та обмежений строк дії токенів відповідають рекомендаціям щодо безпечної роботи з токенами [13]. Конфігурацію безпеки наведено в додатку К, а запит і перевірку одноразового коду — в додатку М.

Рольовий доступ безпосередньо відображає бізнес-процес: адміністратор керує довідниками й персоналом, оператор оформлює та веде замовлення й зміни, кур'єр працює лише з власними доставками. Під час проєктування враховано типові ризики безпеки API — недостатній контроль авторизації, надмірне розкриття даних і небезпечну конфігурацію [10], [11]. Для їх мінімізації застосовано обмеження доступу за ролями, об'єкти передачі даних, що приховують внутрішню структуру, і збереження паролів та кодів лише у вигляді гешів [12].

Життєвий цикл токенів спроектовано так, щоб поєднати зручність роботи із захищеністю. Токен доступу має короткий строк дії та містить мінімально необхідні відомості про працівника й роль, тоді як токен оновлення дозволяє безшовно продовжити сеанс без повторного введення одноразового коду. Короткий строк дії токена доступу зменшує вікно можливої атаки в разі його перехоплення, а підпис токена секретом, що зберігається лише на сервері, унеможливорює його підробку на боці клієнта [13].

Розмежування прав реалізовано не лише на рівні маршрутів, а й на рівні окремих методів контролерів за допомогою декларативних анотацій авторизації. Завдяки цьому навіть у разі помилки в конфігурації маршрутів доступ до операції залишається обмеженим відповідною роллю, що реалізує принцип багаторівневого захисту. Одноразові коди та паролі зберігаються виключно у вигляді гешів, обчислених адаптивним алгоритмом, тому навіть за умови витоку вмісту бази відновити вихідні секрети практично неможливо [12].

2.4. Фронтенд, порівняння альтернатив та відповідність вимогам

Клієнтську частину реалізовано як односторінковий застосунок на чистому JavaScript із маршрутизацією через History API. Такий підхід не потребує важкого фронтенд-фреймворку, лишається зрозумілим для супроводу й дозволяє

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		38

відкривати потрібний розділ безпосередньо за адресою без перезавантаження сторінки. Стилiзацію виконано власними засобами CSS, а графіки аналітики побудовано бібліотекою візуалізації даних [15], [16], [25].

Взаємодія з сервером відбувається через захищені REST-запити, які виконуються засобами HTTP-клієнта Axios [17]. Клієнт сконфігуровано з базовою адресою API та обмеженням часу очікування, а перехоплювач запитів (request interceptor) автоматично додає до заголовка авторизації токен доступу після входу. Перехоплювач відповідей обробляє ситуацію недійсного токена: у разі відмови в доступі він прозоро оновлює токен через відповідний ендпоінт і повторює початковий запит, а якщо оновлення неможливе — повертає працівника на сторінку входу. Завдяки чіткому контракту об'єктів передачі даних інтерфейс працює з прогнозованими структурами, що спрощує його розвиток незалежно від доменного ядра (фрагмент HTTP-клієнта наведено в додатку Л).

Для пришвидшення повторних переходів застосовано кешування сторінок: уже завантажений екран показується миттєво, а дані за потреби оновлюються у фоні. Інтерфейс адаптовано до різної ширини екрана, а під час холодного відкриття сторінки відображаються каркасні елементи-заглушки, що зменшує відчуття затримки та відповідає рекомендаціям щодо зручності й доступності [24].

Розташування статичних файлів клієнта в межах того самого застосунку, який резолвить їх через конфігурацію MVC, спрощує запуск і демонстрацію: серверна й клієнтська частини не потребують двох окремих процесів збирання. За потреби фронтенд може бути винесений в окремий застосунок без зміни доменної логіки та API.

Вибір способу автентифікації за одноразовим кодом замість традиційної пари «логін — пароль» зумовлений специфікою користувачів системи. Працівники сервісу вже ідентифікуються за номером телефону, тому використання цього ж номера для входу є природним і не вимагає запам'ятовування додаткових паролів. Крім того, відсутність постійного пароля знижує ризики, пов'язані з його витоком або повторним використанням, а обмежений строк дії коду мінімізує вікно можливої атаки.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

Важливим рішенням є також відмова від зберігання серверних сесій на користь токенів. Безсесійний підхід спрощує горизонтальне масштабування, оскільки сервер не зберігає стану між запитами, і природно поєднується з REST-API, яке може обслуговувати різні типи клієнтів.

Водночас він вимагає уважного поводження з токенами — обмеження строку дії, захищеного зберігання секрету підпису та механізму оновлення, що й реалізовано в системі.

Клієнтську маршрутизацію реалізовано засобами History API, що дозволяє змінювати адресу сторінки та її вміст без повного перезавантаження. Кожному розділу інтерфейсу відповідає власний маршрут, тому потрібний екран можна відкрити безпосередньо за посиланням, а навігація браузера (кнопки «назад» і «вперед») працює очікувано. Маршрутизатор також виконує роль захисного бар'єра: перш ніж показати захищений розділ, він перевіряє наявність дійсного токена й за його відсутності перенаправляє працівника на сторінку входу.

Взаємодію із сервером інкапсульовано в єдиному HTTP-клієнті з перехоплювачами запитів і відповідей. Перехоплювач запитів автоматично додає токен доступу до кожного звернення, а перехоплювач відповідей централізовано обробляє ситуацію простроченого токена: він прозоро оновлює токен і повторює початковий запит, тож працівник не помічає переривання роботи. Така централізація звільняє код окремих сторінок від повторюваної логіки автентифікації та обробки помилок і робить поведінку клієнта узгодженою.

Зручність сприйняття інтерфейсу підвищено за рахунок кешування вже завантажених сторінок та відображення каркасних елементів-заглушок під час очікування даних, що зменшує відчуття затримки. Розмітку адаптовано до різної ширини екрана, а під час проєктування враховано загальні рекомендації щодо доступності вебінтерфейсів. Водночас уся клієнтська частина спирається лише на стабільний контракт об'єктів передачі даних, тому може розвиватися або навіть бути замінена іншим клієнтом без змін у доменному ядрі та інтерфейсі прикладного програмування [24].

					3-ІПЗ-22-1	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Розглянувши всі складові обраного стеку — від серверного ядра до клієнтської частини, — доцільно зіставити їх з альтернативними технологічними рішеннями, щоб показати обґрунтованість зробленого вибору, а не його випадковість.

Під час вибору стеку розглядалися й альтернативні серверні платформи. Узагальнене порівняння основних варіантів за критеріями типізації, екосистеми безпеки, підтримки модульності та придатності для проєкту наведено в таблиці 2.2.

Таблиця 2.2

Порівняння альтернативних технологічних рішень

Технологія	Сильні сторони	Чому не обрано
Java + Spring Boot	Статична типізація, зріла безпека, модульність, JPA	Обрано як основний стек
Node.js (Express/Nest)	Швидке створення API, єдина мова з фронтендом	Слабша статична типізація доменної моделі
Python (Django/FastAPI)	Швидка розробка, готова адмінка (Django)	Складніше реалізувати чисті порти й адаптери
C# + ASP.NET Core	Якісна безпека та продуктивність	Потребував би повного переписування проєкту

Java зі Spring Boot обрано тому, що для системи з акцентом на чисту архітектуру важливо поєднати статичну типізацію доменної моделі, зрілу екосистему безпеки, підтримку реальної багатомодульності та зручну інтеграцію з реляційною базою даних. Node.js зручний для швидкого API, але контроль структури портів і DTO потребує додаткової дисципліни; Python із Django тяжіє до власної монолітної структури; ASP.NET Core є сильним варіантом, проте зміна платформи означала б переписування системи.

Для бази даних розглядалися також MySQL і MongoDB. Реляційні MySQL і MariaDB придатні для багатьох проєктів, однак PostgreSQL має репутацію універсальної СКБД із розвиненими можливостями запитів та обмежень. Документо-орієнтована MongoDB була б доречною для слабоструктурованих даних, але предметна галузь доставки має багато чітких зв'язків, тому реляційна модель є природнішою.

Розглядаючи альтернативні серверні платформи докладніше, можна окреслити, за яких умов кожна з них була б доречною. Середовище Node.js доцільне там, де визначальною є висока кількість одночасних мережових з'єднань і єдина мова з фронтом, проте контроль структури портів та об'єктів передачі даних у ньому покладається переважно на дисципліну розробника, а не на компілятор. Платформа на основі Python пришвидшує початкову розробку й має готові адміністративні засоби, але її типова структура тяжіє до власних шаблонів, що ускладнює реалізацію чистого поділу домену й інфраструктури.

Платформа ASP.NET Core є сильним конкурентом за якістю засобів безпеки та продуктивністю й цілком придатна для подібних систем; вирішальним аргументом проти неї стала не технічна слабкість, а недоцільність повного переписування вже спроектованого на Java доменного ядра. Отже, вибір основного стеку є наслідком не абсолютної переваги однієї технології, а найкращої відповідності сукупності вимог роботи та вже ухваленим архітектурним рішенням.

Вибір реляційної СКБД також ґрунтується на характері даних. Предметна галузь доставки має багато чітко визначених зв'язків і потребує суворих гарантій узгодженості, тому транзакційна реляційна модель із підтримкою зовнішніх ключів і перевірочних обмежень підходить краще за документо-орієнтоване сховище. PostgreSQL обрано як універсальну й надійну СКБД, що поєднує сувору реляційність із розвиненими можливостями запитів, а за потреби — і з підтримкою слабоструктурованих даних, що залишає простір для подальшого розвитку без зміни сховища.

Таким чином, обраний стек не є єдино можливим, проте він найкраще узгоджується з вимогами роботи та гексагональною архітектурою, дозволяючи продемонструвати професійні інженерні рішення без надмірного ускладнення системи.

Завершальним питанням опису стеку є перевірка його придатності на практиці: організація тестування, свідомий облік ризиків кожної з обраних технологій та оцінка відповідності стеку сформульованим у першому розділі вимогам.

					3-ІПЗ-22-1	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

Для перевірки системи застосовано бібліотеку модульного тестування JUnit разом із засобами тестування Spring [19]. Тести розміщено в головному модулі застосунку, де зібрані всі залежності, що дозволяє перевіряти як окремі доменні сервіси, так і інтеграцію компонентів. Особливу увагу приділено бізнес-сценаріям робочої зміни та замовлення, а не лише формальній наявності тестів.

Будь-який стек має ризики, які слід враховувати. Для Spring Boot це спокуса змішати контролери, сервіси й репозиторії в одному шарі — її мінімізовано реальними модулями та портами. Для JPA ризиком є неконтрольовані зв'язки та зайві запити до бази, тому використано об'єкти передачі даних і обмежено вибірки. Для JWT ризиком є неправильне зберігання секрету чи занадто великий строк дії токенів, що враховано винесенням секретів у конфігурацію та обмеженням строків.

Для контейнеризації ризиком є залежність від локальної інсталяції Docker і можливі конфлікти портів, що пом'якшено використанням нестандартного порту бази даних. Загальним ризиком кваліфікаційної роботи є невідповідність тексту фактичній реалізації, тому в розділах описано лише реалізовані функції, а напрями подальшого розвитку позначено окремо.

Обраний стек відповідає вимогам кваліфікаційної роботи: він дозволяє показати повний цикл інженерної роботи — від моделювання домену й проектування модулів до реалізації веб-доступу, підключення бази даних, захисту ендпоінтів і тестування. Технології достатньо складні, щоб продемонструвати інженерні компетентності, але не надмірні для одного дипломного проєкту, що узгоджується з моделлю характеристик якості програмного забезпечення [30].

Доцільно окремо підкреслити роль автоматизованого тестування у супроводі обраного стеку. Наявність набору тестів дозволяє безпечно вносити зміни: після кожної правки тести підтверджують, що ключові бізнес-сценарії й надалі працюють коректно. Такий набір тестів придатний також для автоматичного запуску в системах неперервної інтеграції, наприклад GitHub Actions, що дає змогу перевіряти кожну зміну коду перед її інтеграцією [27]. Це особливо важливо для системи з фінансовими розрахунками та контролем доступу, де помилка може мати

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		43

відчутні наслідки, тому тестове покриття розглядається не як формальність, а як складова якості програмного продукту.

Стратегію тестування побудовано за принципом піраміди: основу складають численні швидкі модульні тести доменних сервісів і перетворювачів, над ними розташовано меншу кількість інтеграційних тестів, що піднімають контекст застосунку та перевіряють взаємодію компонентів із реальною конфігурацією, а ручну перевірку інтерфейсу прикладного програмування застосовано як завершальний контроль наскрізних сценаріїв. Такий розподіл забезпечує водночас швидкий зворотний зв'язок під час розробки й достатню впевненість у коректності системи загалом.

Особливу увагу приділено перевірці саме бізнес-правил, а не формальної наявності тестів: контролюється відмова у створенні замовлення без відкритої зміни, правильність розрахунку суми та знижки, коректність переходів статусів і дотримання прав доступу за ролями. Тестуються не лише очікувані, а й помилкові сценарії, оскільки саме коректна обробка некоректного вводу відрізняє надійну систему від крихкої. Виявлені під час тестування ризики стека свідомо задокументовано разом зі способами їх мінімізації, що робить аналіз надійності частиною інженерної роботи [30].

Узагальнюючи, можна стверджувати, що обраний стек технологій становить узгоджену сукупність, у якій кожен компонент вирішує конкретну задачу та не суперечить архітектурі. Java та Spring Boot формують серверне ядро, Maven забезпечує модульність, PostgreSQL і JPA відповідають за надійне збереження даних, Docker — за повторюване середовище, Spring Security з JWT — за захист, а односторінковий клієнт — за зручну взаємодію. Саме узгодженість, а не окремі переваги компонентів, робить цей стек придатним для реалізації системи.

Висновки до розділу 2

У другому розділі описано технологічний стек системи та наведено аргументацію його вибору. Обґрунтовано використання мови Java та платформи Spring Boot, багатомодульної структури Maven для реалізації гексагональної

					3-ІПЗ-22-1	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

архітектури, бази даних PostgreSQL із доступом через Spring Data JPA та супроводом міграціями, контейнеризації через Docker, засобів безпеки Spring Security з JWT та автентифікацією за одноразовим кодом, а також односторінкового клієнта на JavaScript.

Виконано порівняння з альтернативними технологічними рішеннями та показано, чому обраний стек найкраще узгоджується з вимогами роботи. Розглянуто ризики використання стеку й способи їх мінімізації. Обрані технології забезпечують технічну основу для реалізації функціоналу, описаного у третьому розділі.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		45

РОЗДІЛ 3. ОПИС ВИКОРИСТАННЯ ПРОГРАМИ

3.1. Встановлення системи та структура інтерфейсу

Перед використанням системи її потрібно встановити та налаштувати. Для роботи необхідні встановлені Java 17, система збирання Maven (або вбудований Maven Wrapper), Docker Desktop із підтримкою Docker Compose та сучасний браузер. Розгортання складається з двох основних кроків: запуску бази даних у контейнері та запуску серверного застосунку, після чого вебінтерфейс стає доступним у браузері.

Першим кроком запускається контейнер бази даних командою `docker compose up`. Контейнер піднімає PostgreSQL із заздалегідь визначеними параметрами та томом для збереження даних; готовність бази перевіряється вбудованою перевіркою стану. Локальний порт 55432 використано для уникнення конфлікту з можливою локальною інсталяцією СКБД. Під час першого запуску застосунку міграції Flyway автоматично створюють схему бази даних, тому окремих ручних дій зі створення таблиць не потрібно.

Параметри підключення до бази даних і секрети застосунку винесено в конфігурацію та дубльовано змінними середовища, що дозволяє змінювати їх без редагування коду. Перелік основних параметрів налаштування наведено в таблиці 3.1.

Таблиця 3.1

Основні параметри налаштування системи

Параметр	Призначення
SPRING_DATASOURCE_URL	Адреса підключення до PostgreSQL (за замовчуванням порт 55432)
SPRING_DATASOURCE_USERNAME	Облікові дані доступу до бази даних
JWT-секрет та строки дії токенів	Підпис і час життя токенів доступу й оновлення
SMS_TOKEN, SENDER_NAME, sender.url	Параметри сервісу надсилання SMS-кодів
Параметри Cloudinary	Доступ до хмарного сховища зображень товарів і акцій
Параметри Telegram-бота	Токен бота для сповіщення кур'єрів про нові замовлення

Другим кроком збирається й запускається серверний застосунок засобами Maven Wrapper: команда збирання компілює всі модулі в правильному порядку, а запуск головного модуля піднімає вбудований вебсервер. Після старту застосунок обслуговує як REST-API, так і статичні файли інтерфейсу, тож система стає доступною за адресою сервера у браузері.

Першого працівника з роллю адміністратора створюють безпосередньо в базі даних або через початкові дані, після чого решта працівників додаються вже через інтерфейс адміністратора з підтвердженням номера телефону одноразовим кодом. Після цього система готова до роботи: адміністратор наповнює каталог, а оператор може відкривати зміну та приймати замовлення.

Під час налаштування важливо враховувати залежність системи від зовнішніх сервісів. Для надсилання одноразових кодів використовується сторонній SMS-шлюз, для зберігання зображень — хмарний сервіс, а для сповіщення кур'єрів — бот месенджера. Перед початком експлуатації відповідні параметри доступу мають бути коректно задані у конфігурації; за їх відсутності пов'язаний функціонал буде недоступним, тоді як основні операції із замовленнями залишаються працездатними.

Рекомендований порядок першого запуску такий: спочатку піднімається контейнер бази даних і перевіряється його готовність, потім запускається застосунок, який застосовує міграції та створює схему, після чого перевіряється доступність вебінтерфейсу у браузері. Такий покроковий запуск дозволяє локалізувати можливі проблеми: якщо застосунок не стартує, причину слід шукати в підключенні до бази або конфігурації, а не в самому коді. Після успішного старту система готова до наповнення довідників і початку роботи.

Для зручності діагностування можливих проблем розгортання передбачено чітке розмежування джерел помилок. Якщо застосунок не стартує, причину насамперед слід шукати в доступності бази даних і коректності параметрів підключення, а не в самому коді: послідовний запуск спочатку контейнера бази, а потім застосунку дозволяє локалізувати збій уже на першому кроці. Журнал

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		47

запуску застосунку фіксує застосування міграцій та ініціалізацію компонентів, що допомагає швидко встановити етап, на якому виникла проблема.

Залежність окремих функцій від зовнішніх сервісів — шлюзу надсилання SMS, хмарного сховища зображень і месенджера для сповіщення кур'єрів — враховано в порядку налаштування. За відсутності відповідних параметрів пов'язаний функціонал стає недоступним, проте основні операції із замовленнями залишаються працездатними, тому систему можна розгортати поетапно, під'єднуючи зовнішні інтеграції в міру готовності. Першого працівника з роллю адміністратора створюють під час початкової ініціалізації, після чого решта облікових записів додаються вже штатним шляхом — через інтерфейс адміністратора з підтвердженням номера телефону одноразовим кодом.

Після встановлення та налаштування системи можна перейти до опису безпосередньої роботи з нею. Користування програмою починається з авторизації працівника та ознайомлення зі структурою інтерфейсу, яка визначає доступний йому набір розділів.

Робота з програмою починається з авторизації. На сторінці входу працівник вводить номер телефону, отримує одноразовий код у SMS-повідомленні та підтверджує його. Після успішної перевірки система видає токен доступу й перенаправляє працівника на стартовий екран відповідно до його ролі. Зовнішній вигляд сторінки входу наведено на рис. 3.1.

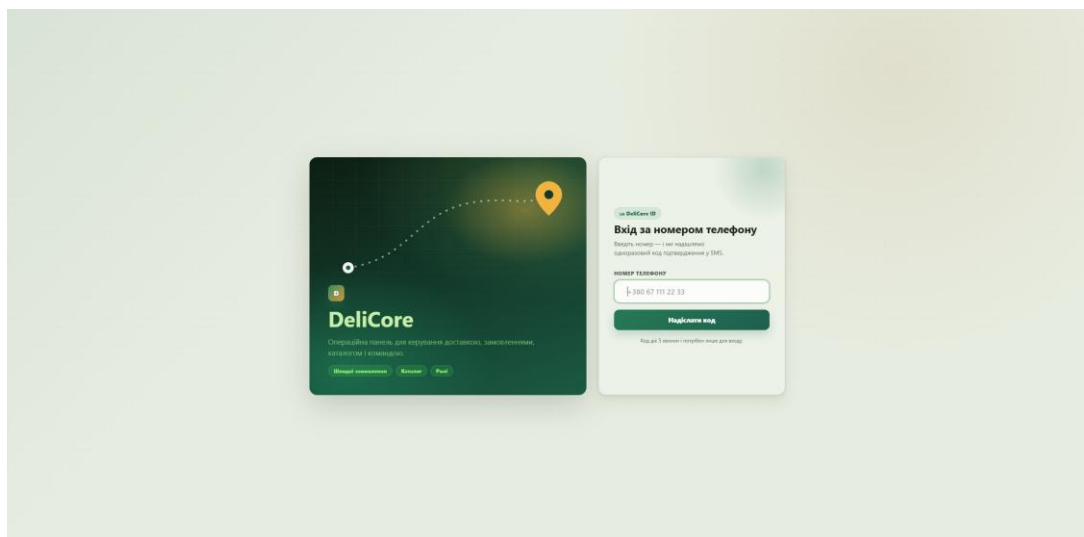


Рис. 3.1 – Сторінка входу до системи

					3-ІПЗ-22-1	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

Інтерфейс системи побудовано як односторінковий застосунок із бічною панеллю навігації, верхньою панеллю та робочою областю. Вебінтерфейс призначений для адміністратора й оператора; стартовим розділом для них є список замовлень, а доступний набір розділів залежить від ролі. Кур'єр у вебінтерфейсі не працює — його взаємодію із системою винесено в Telegram-бот (див. підрозділ 3.3). Перехід між розділами виконується без перезавантаження сторінки.

Права доступу до розділів і операцій визначаються роллю та контролюються на рівні сервера. Розподіл прав за ролями узагальнено в таблиці 3.2; саме він визначає, які розділи інтерфейсу й які дії стають доступними після входу. Спроба виконати дію без відповідної ролі відхиляється сервером, тому інтерфейс приховує недоступні розділи, а серверна перевірка унеможливорює обхід обмежень.

Таблиця 3.2

Розподіл прав доступу за ролями

Операція	ADMIN	OPERATOR	COURIER
Перегляд та оформлення замовлень	+	+	–
Зміна статусу, призначення кур'єра	+	+	–
Відкриття/закриття зміни, аналітика	+	+	–
Керування каталогом, акціями	+	–	–
Керування працівниками та ролями	+	–	–
Операції кур'єра через Telegram (власні доставки)	–	–	+

Така організація інтерфейсу відповідає рольовій моделі системи: кожен працівник бачить лише ту частину функціоналу, яка потрібна йому для роботи. Це спрощує користування програмою, зменшує ймовірність помилкових дій і підвищує захищеність, оскільки адміністративні операції недоступні оператору чи кур'єру навіть на рівні відображення.

Структурно інтерфейс побудовано як стійкий каркас односторінкового засто-

стосунку: бічна панель навігації, верхня панель з відомостями про поточного працівника та робоча область, у якій змінюється лише вміст активного розділу. Завдяки цьому перехід між розділами не призводить до перезавантаження сторінки й відбувається миттєво, а спільні елементи інтерфейсу залишаються незмінними. Набір пунктів навігації формується динамічно залежно від ролі: працівник бачить лише ті розділи, до яких має доступ, що спрощує орієнтування та зменшує ймовірність помилкових дій.

Захист доступу реалізовано узгоджено на клієнті та сервері. На клієнті недоступні розділи приховуються, а маршрутизатор не дозволяє відкрити їх безпосередньо за адресою без дійсного токена; на сервері кожна операція додатково перевіряється за роллю, тому приховування елементів інтерфейсу є зручністю, а не єдиним механізмом захисту. Навіть якщо звернутися до закритого ресурсу в обхід інтерфейсу, сервер відхилить запит, що унеможливило б обхід обмежень і робить рольову модель надійною.

Процес входу побудовано як послідовність двох кроків, що відповідає обраному способу автентифікації за одноразовим кодом. На першому кроці працівник вводить номер телефону й отримує код у SMS-повідомленні, на другому — підтверджує його, після чого сервер видає токен доступу та перенаправляє працівника на стартовий екран відповідно до його ролі. Стан автентифікації зберігається на боці клієнта у вигляді токена, який автоматично додається до кожного наступного запиту, тому під час роботи в межах сеансу повторне введення коду не потрібне. Якщо токен втрачає чинність, клієнт прозоро оновлює його, а за неможливості оновлення повертає працівника на сторінку входу, що робить поведінку системи передбачуваною навіть за тривалої роботи.

3.2. Оформлення замовлень та керування їх виконанням

Центральним розділом для оператора є список замовлень, оформлений у вигляді канбан-дошки зі стовпцями за статусами. Кожне замовлення відображається карткою з номером, клієнтом, сумою та адресою, а перехід картки між стовпцями відповідає зміні статусу. Над дошкою розміщено банер стану робочої зміни. Вигляд екрана замовлень наведено на рис. 3.2.

					3-ІПЗ-22-1	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

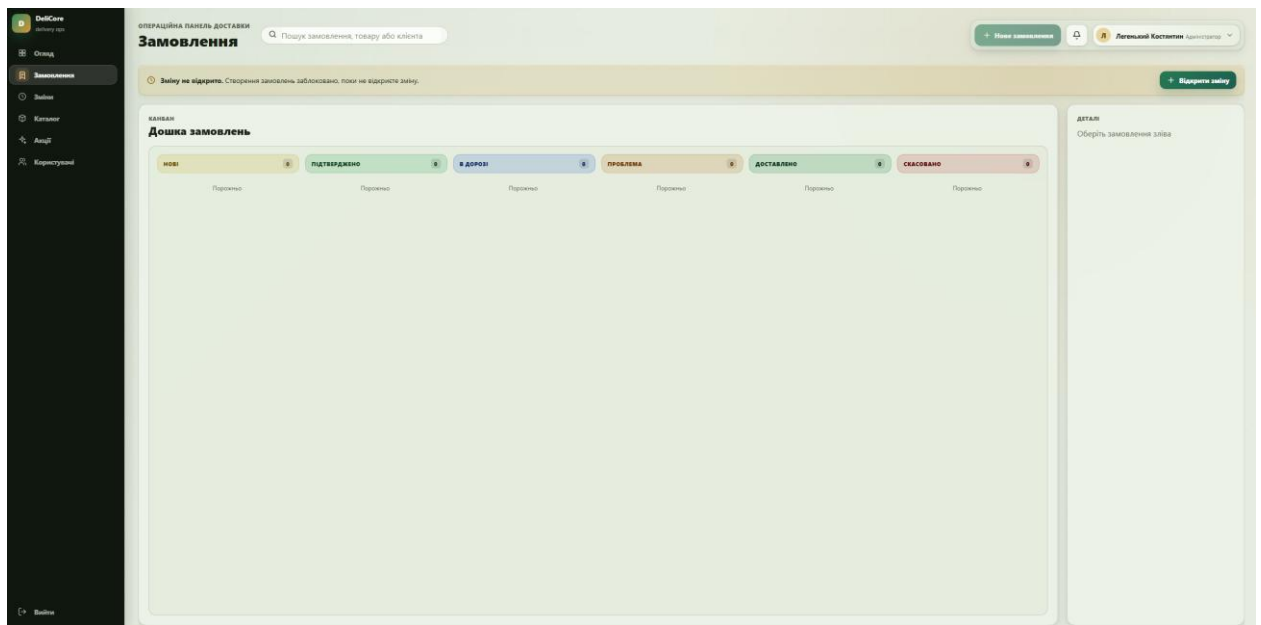


Рис. 3.2 – Екран замовлень (канбан-дошка)

Канбан-подання обрано для списку замовлень не випадково: воно наочно відображає рух замовлення етапами виконання й дозволяє оператору одним поглядом оцінити, скільки замовлень перебуває на кожній стадії. Замовлення, що потребують уваги, візуально виділяються, тому оператор одразу бачить пріоритети, не переглядаючи весь список. Таке подання відповідає характеру операційної роботи, де важлива саме поточна картина стану всіх замовлень.

Якщо робоча зміна не відкрита, кнопка створення замовлення заблокована, а банер пропонує відкрити зміну. Це відповідає бізнес-правилу, за яким замовлення приймаються лише в межах відкритої зміни. Після відкриття зміни оператор може створювати замовлення; спроба створити замовлення без відкритої зміни відхиляється сервером із відповідним повідомленням.

Створення замовлення виконується в модальному вікні, що відіграє роль внутрішнього операторського оформлення. Оператор вводить ім'я й телефон клієнта, адресу доставки та коментар, додає товари з каталогу із зазначенням кількості, за потреби обирає акцію, а система показує підрахунок суми. Послідовність дій системи під час створення замовлення наведено на рис. 3.3.



Рис. 3.3 – Схема послідовності створення замовлення

Під час збереження система перевіряє наявність відкритої зміни, знаходить або створює клієнта за номером телефону, формує позиції зі знімком ціни кожного товару, обчислює проміжну суму, застосовує акцію та розраховує підсумкову вартість. Якщо товар не знайдено, кількість некоректна або акція неактивна, операція переривається зрозумілою помилкою, і замовлення не зберігається в неповному стані. Це гарантує цілісність обліку та правильність розрахунків.

Знижка за акцією розраховується автоматично: для відсоткової акції вона обчислюється як частка від суми з округленням до копійок, а для фіксованої береться вказане значення; при цьому знижка не може бути від'ємною й не перевищує суму замовлення. Завдяки автоматизації цих обчислень оператор позбавлений потреби рахувати суму вручну, що прискорює оформлення та виключає арифметичні помилки.

Для зручності оператора канбан-дошка передбачає не лише перегляд, а й швидкі дії над замовленням безпосередньо з картки. Натиснувши на картку, оператор відкриває деталі замовлення зі складом позицій, сумою, даними клієнта та поточним статусом, звідки може змінити статус або призначити кур'єра. Таке

поєднання загального огляду на дошці й детального перегляду в картці відповідає типовому сценарію роботи, коли оператор спершу оцінює загальну картину, а потім працює з конкретним замовленням.

Окремо варто пояснити поведінку системи у разі помилкових дій. Якщо оператор намагається створити замовлення без жодної позиції, із неіснуючим товаром або без обов'язкових контактних даних, система не зберігає замовлення й показує зрозуміле повідомлення про причину відмови. Так само контролюються операції зі зміною, тому стан системи завжди залишається узгодженим, а працівник отримує підказку, як виправити введені дані, замість незрозумілої технічної помилки.

Правила перевірки даних під час оформлення замовлення задано так, щоб у систему не потрапляли суперечливі записи. Контролюється наявність відкритої зміни, існування й доступність кожного обраного товару, додатність кількості позицій та повнота контактних даних клієнта. Лише за умови проходження всіх перевірок замовлення зберігається; інакше операція переривається, а працівник отримує зрозуміле повідомлення про причину відмови та підказку, як виправити введені дані.

Важливим проєктним рішенням є фіксація ціни товару у вигляді знімка в момент оформлення замовлення. Кожна позиція зберігає ту ціну, що діяла на момент створення, тому подальша зміна ціни в каталозі не спотворює вже оформлені замовлення та збережену історію розрахунків. Клієнт ідентифікується за номером телефону: за повторного звернення система знаходить наявний запис замість створення дублікату, завдяки чому накопичується коректна база контактів і стає можливим підрахунок кількості унікальних клієнтів за зміну.

Оформленням замовлення робота з ним не завершується: створене замовлення проходить контрольовані етапи виконання. Тому далі розглянуто керування статусами замовлення та призначення кур'єра, які забезпечують його супроводження аж до моменту доставки.

Після створення замовлення розпочинається процес його обробки. Оператор переглядає нові замовлення, підтверджує їх, переводить у стан підготовки та

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		53

передавання в доставку. Статус замовлення є контрольованим переліком, а не довільним текстом, що дозволяє у будь-який момент бачити стан кожного замовлення й фільтрувати їх за етапами. Схему станів замовлення наведено на рис. 3.4.



Рис. 3.4 – Схема станів замовлення

Перелік станів та їх зміст наведено в таблиці 3.3. Під час переходу замовлення у стан виконаного або скасованого система фіксує відповідну часову позначку, яка згодом використовується в аналітиці зміни. Окремі стани відображають проблему під час доставки та незавершеність замовлення на момент закриття зміни.

Таблиця 3.3

Статуси замовлення

Статус	Зміст
NEW	Замовлення створено оператором, очікує підтвердження
CONFIRMED	Замовлення підтверджено й прийнято в роботу
PREPARING	Замовлення готується до передачі кур'єру
IN_DELIVERY	Кур'єр розпочав доставку
DELIVERED	Доставку успішно завершено
CANCELLED	Замовлення скасовано
PROBLEM	Кур'єр позначив проблему під час доставки
UNFINISHED	Замовлення не завершено вручну на момент закриття зміни

Призначення кур'єра виконується оператором вручну зі списку доступних кур'єрів або автоматично — системою диспетчеризації, яка обирає кур'єра зі статусом «онлайн» і надсилає йому пропозицію доставки. Сповіщення кур'єра реалізовано через окремий канал (Telegram-бот із кнопками прийняття), що

дозволяє кур'єру оперативно отримати інформацію про нове замовлення з адресою та сумою.

Завдяки контрольованому переліку статусів і чіткому призначенню кур'єра кожне замовлення має однозначний стан і відповідального виконавця. Це усуває невизначеність, характерну для ручного ведення замовлень, і дає змогу будувати достовірну аналітику за результатами зміни.

Доступ до призначення кур'єра та зміни статусів мають лише адміністратор і оператор, тоді як кур'єр впливає на стан замовлення опосередковано — через дії у Telegram-боті. Такий розподіл відповідальності запобігає конфліктам: оператор контролює загальний потік і не може випадково втрутитися в роботу, закріплену за кур'єром, а кур'єр не має доступу до чужих замовлень чи адміністративних операцій. Це робить поведінку системи передбачуваною й узгодженою з реальним розподілом обов'язків у сервісі.

Перехід замовлення між статусами супроводжується фіксацією супутніх даних: під час підтвердження доставки зберігається час її виконання, а під час позначення проблеми — її причина. Завдяки цьому система зберігає не лише поточний стан замовлення, а й контекст його зміни, що згодом використовується в аналітиці та допомагає розбирати спірні ситуації, спираючись на зафіксовані факти, а не на усні пояснення.

Призначення кур'єра може виконуватися як вручну, так і автоматично — через підсистему диспетчеризації. У режимі автоматичного підбору система формує пропозицію доставки кур'єру зі статусом «доступний» і надсилає її в месенджер; якщо кур'єр відхиляє пропозицію або не відповідає протягом відведеного часу, пропозиція автоматично передається наступному вільному кур'єру. Такий механізм послідовного оферування з обмеженням часу очікування запобігає «зависанню» замовлення на одному виконавці й прискорює його передавання в доставку.

Перелік статусів замовлення є контрольованим, а переходи між ними підпорядковані бізнес-логіці, а не довільним діям користувача. Окремі статуси відображають нештатні ситуації — позначення проблеми під час доставки та

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		55

незавершеність замовлення на момент закриття зміни, — завдяки чому жодне замовлення не залишається в невизначеному стані. Кожен перехід супроводжується фіксацією супутніх даних: під час підтвердження доставки зберігається час її виконання, а під час позначення проблеми — її причина, що формує достовірну основу для подальшої аналітики та розбору спірних випадків.

3.3. Робота кур'єра, сповіщення та аналітика робочих змін

На відміну від адміністратора й оператора, кур'єр не працює у вебінтерфейсі системи й не автентифікується за одноразовим кодом — для ролі кур'єра вхід через браузер свідомо вимкнено. Натомість кур'єр взаємодіє із системою через Telegram-бот, що є природним для роботи «в полі»: кур'єр уже користується смартфоном, а месенджер не потребує встановлення окремого застосунку та миттєво доставляє сповіщення про нові замовлення.

Після того як оператор призначає замовлення кур'єру (або спрацьовує автоматична диспетчеризація), система надсилає йому в Telegram повідомлення з основними даними замовлення — номером, адресою доставки, сумою та коментарем клієнта — і кнопками дії. Зовнішній вигляд такого повідомлення наведено на рис. 3.5.



Рис. 3.5 – Сповіщення кур'єра та кнопки дій у Telegram-боті

					3-ІПЗ-22-1	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

Кур'єр керує доставкою натисканням кнопок під повідомленням. Приймавши замовлення, він розпочинає доставку, після чого замовлення переходить у стан «у доставці»; після успішного вручення кур'єр натискає «Доставлено», і замовлення набуває стану виконаного з фіксацією часу. Якщо виникає проблема (клієнт недоступний, відмова від замовлення тощо), кур'єр позначає її, а оператор отримує змогу оперативного відреагувати. Технічно натискання кнопок надходять боту, який викликає відповідні захищені операції сервера (повний код контролера кур'єрських операцій наведено в додатку Ж).

На стороні сервера ці дії обслуговує окремий контролер під роллю COURIER з операціями перегляду власних доставок, початку доставки, підтвердження виконання та позначення проблеми, а вихідний порт сповіщення формує й надсилає повідомлення кур'єру (його реалізацію наведено в додатку Г). Завдяки тому, що бот і вебінтерфейс працюють з однією базою даних через спільний сервер, будь-яка дія кур'єра одразу стає видимою оператору в списку замовлень, і навпаки.

Такий підхід відображає реальний розподіл відповідальності: оператор контролює весь потік замовлень і приймає рішення, а кур'єр зосереджений на виконанні конкретних доставок і взаємодії із системою мінімальним, але достатнім набором дій просто з месенджера. Це спрощує роботу кур'єра, не вимагає від нього окремого вебкабінету й зменшує ймовірність помилок.

Технічно взаємодію з месенджером реалізовано за моделлю довготривалого опитування (long polling): окремий компонент серверної частини періодично запитує у служби месенджера нові події та передає їх на обробку. Натискання кнопок під повідомленням надходять як службові події, які застосунок зіставляє з конкретним кур'єром та замовленням і перетворює на виклики відповідних захищених операцій сервера. Такий підхід не потребує зовнішньої доступної адреси для приймання зворотних викликів, що спрощує розгортання системи в навчальному середовищі.

Прив'язку кур'єра до його облікового запису в месенджері реалізовано через збереження ідентифікатора чату в моделі працівника. Завдяки цьому система точно знає, кому надсилати пропозицію доставки, а кур'єр не потребує жодних

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

додаткових облікових даних, окрім уже наявного акаунта месенджера. Якщо ідентифікатор чату ще не прив'язано, подія сповіщення безпечно протоколюється без переривання основного процесу, що робить інтеграцію стійкою до неповних налаштувань.

Вибір месенджера як каналу взаємодії кур'єра є свідомим проєктним рішенням, що відповідає реальним умовам роботи «в полі». Кур'єр уже користується смартфоном і месенджером, тому не потребує встановлення окремого застосунку чи запам'ятовування додаткових паролів, а сповіщення про нове замовлення надходить миттєво. Оскільки бот і вебінтерфейс працюють зі спільною базою даних через єдиний сервер, будь-яка дія кур'єра одразу стає видимою оператору, і навпаки, що забезпечує узгодженість стану без окремого каналу узгодження.

Виконані кур'єром доставки та пройдені замовленнями статуси формують дані, які система узагальнює у вигляді аналітики. Тому, описавши роботу кур'єра, доцільно розглянути аналітику робочих змін, що перетворює накопичені під час роботи дані на показники для прийняття управлінських рішень.

Окремий розділ системи присвячено робочим змінам та аналітиці. На сторінці зміни відображається картка поточної зміни з можливістю її відкриття або закриття, ключові показники та історія попередніх змін. Вигляд сторінки аналітики зміни наведено на рис. 3.6.

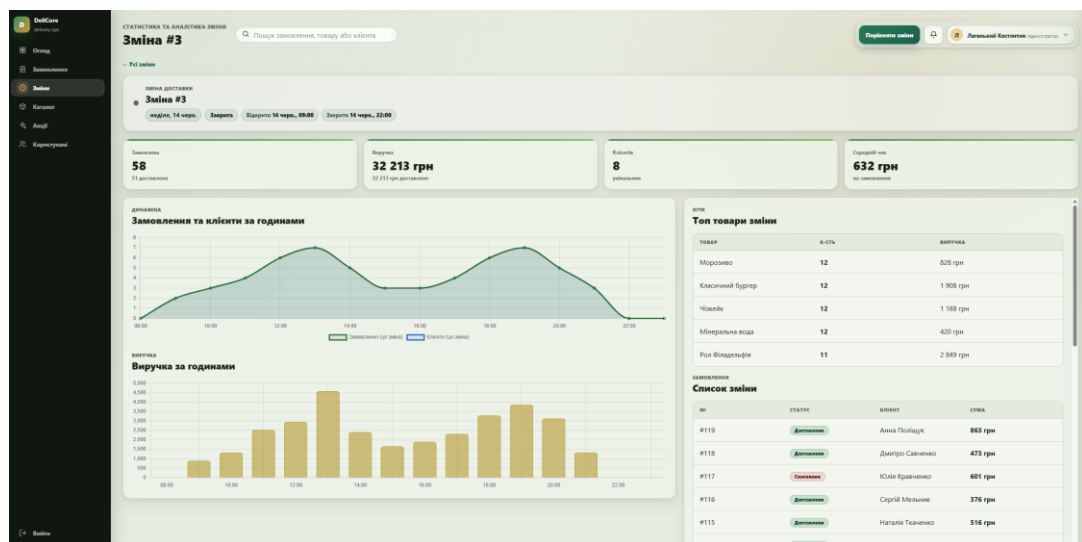


Рис. 3.6 – Сторінка статистики робочої зміни

Деталі окремої зміни містять кількість замовлень, клієнтів, виручку та середній чек, перелік найпопулярніших товарів, а також погодинні графіки кількості замовлень і клієнтів та стовпчикову діаграму виручки. Графіки побудовано на основі даних, згрупованих за годинами доби, що дозволяє оцінити розподіл навантаження протягом зміни.

Передбачено можливість накладання кривих іншої зміни для порівняння, а також окрему сторінку порівняння обраних змін за різними метриками. Це дає керівнику інструмент для аналізу динаміки: можна порівняти завантаженість у різні дні тижня, оцінити вплив акцій або змін у роботі сервісу.

Під час закриття зміни всі її незавершені замовлення автоматично позначаються як невиконані, а якщо зміну забули закрити, вона автоматично закривається після перевищення максимальної тривалості. Це гарантує, що аналітика будується на коректних даних і жодне замовлення не залишається в невизначеному стані поза обліковим періодом.

Корисною властивістю аналітичного модуля є те, що він не потребує окремого введення даних: усі показники обчислюються на основі вже наявних замовлень і змін. Це означає, що достовірність аналітики безпосередньо залежить від акуратності оперативної роботи — правильно оформлені замовлення та своєчасно змінені статуси автоматично дають коректну статистику, без додаткових зусиль з боку працівників.

Графічне подання показників обрано свідомо: числові таблиці важко сприймати під час роботи, тоді як погодинні криві та стовпчикові діаграми дозволяють миттєво оцінити динаміку й виявити пікові періоди. Поєднання кількох метрик на одному графіку (замовлення та клієнти) додатково показує співвідношення між загальною кількістю замовлень і числом унікальних клієнтів, що є важливим індикатором повторних звернень.

3.4. Адміністрування системи та тестування програмного комплексу

Адміністративні розділи доступні лише адміністратору й реалізовані у вигляді таблиць із формами створення та редагування. У розділі каталогу адміністратор додає й редагує товари, зазначаючи назву, опис, ціну, категорію,

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

доступність і зображення. Категорії товарів керуються в окремому розділі та використовуються для впорядкування каталогу. Вилучення товару виконується як логічне (м'яке) видалення: позиція позначається як видалена, але зберігається в базі, що не порушує цілісності вже оформлених замовлень, які на неї посилаються.

Розділ акцій дозволяє створювати маркетингові пропозиції з відсотковою або фіксованою знижкою та мінімальною сумою замовлення. Створена акція може бути застосована під час оформлення замовлення, після чого знижка автоматично враховується в підсумковій вартості. Завантаження зображень товарів і акцій виконується через хмарний сервіс [18], а в інтерфейсі зображення підвантажуються відкладено з плейсхолдером, що покращує сприйняття під час завантаження каталогу.

У розділі працівників адміністратор додає нових працівників, надсилає їм код підтвердження номера телефону та змінює ролі. Саме тут визначається, хто є оператором, кур'єром або іншим адміністратором. Усі форми перевіряють введені дані й показують повідомлення про помилки, що повертає сервер, тому некоректні дані не потрапляють у систему.

Адміністративні операції є основою для роботи решти системи: без актуального каталогу оператор не може оформити замовлення, а без призначених ролей працівники не отримують доступ до своїх кабінетів. Тому адміністрування розглядається як невід'ємна частина повсякденного використання програми, а не лише як початкове налаштування.

Розглянувши адміністративні розділи, які забезпечують роботу решти системи, доцільно завершити опис перевіркою працездатності реалізованого програмного комплексу. Тому далі описано тестування системи, спрямоване на підтвердження коректності її основних функцій.

Тестування системи спрямоване на перевірку основного функціоналу, а не лише процедури входу. Застосовано поєднання автоматизованого та ручного тестування: доменну логіку перевірено модульними тестами, наскрізні сценарії — інтеграційними тестами, що піднімають контекст застосунку, а ручну перевірку API виконано в середовищі тестування запитів [19], [29].

					3-ІПЗ-22-1	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Основну увагу приділено бізнес-сценаріям робочої зміни та замовлення. Перелік основних тестових сценаріїв та очікуваних результатів наведено в таблиці 3.4.

Таблиця 3.4

Основні сценарії тестування системи

Сценарій	Очікуваний результат
Створення замовлення без відкритої зміни	Відмова з помилкою стану (HTTP 409)
Відкриття зміни, коли вже є відкрита	Відмова з помилкою стану (HTTP 409)
Створення замовлення з кількох позицій	Замовлення створено, суму обчислено правильно
Застосування відсоткової акції	Знижку враховано, підсумок зменшено коректно
Зміна статусу до «виконано»	Статус змінено, зафіксовано час доставки
Закриття зміни з активними замовленнями	Незавершені замовлення позначено як невиконані
Доступ оператора до адмінрозділу	Відмова в доступі (HTTP 403)
Вхід за невірним одноразовим кодом	Відмова в автентифікації

Перевірку прав доступу виконано для кожної ролі: підтверджено, що адміністратор має доступ до всіх розділів, оператор — до операцій із замовленнями, змінами та переглядом каталогу, але не до керування працівниками, а кур'єр — лише до власних доставок. Спроби звернення до закритих ендпоінтів із токеном недостатньої ролі коректно відхиляються, що підтверджує правильність налаштування рольової моделі [10].

Окремо перевірено наскрізний сценарій повного циклу: вхід оператора, відкриття зміни, оформлення замовлення з кількох позицій, призначення кур'єра, перехід замовлення через статуси до виконання та відображення результату в аналітиці зміни. Сценарій виконувався як через інтерфейс, так і через прямі виклики API, і в обох випадках дані залишалися узгодженими.

Результати тестування підтверджують, що реалізований програмний комплекс коректно виконує основні функції відповідно до сформульованих вимог: він забезпечує захищений доступ за ролями, оформлення та супроводження замовлень, контроль робочих змін з автоматичним закриттям і формування аналітики. Це дає підстави вважати систему придатною до практичного використання та демонстрації.

Описаний порядок користування програмою охоплює повний робочий цикл усіх ролей: від встановлення та налаштування системи адміністратором, наповнення каталогу й додавання працівників — до повсякденної роботи оператора з оформлення та супроводження замовлень і виконання доставок кур'єром. Кожен екран інтерфейсу відповідає конкретному етапу бізнес-процесу, описаного в першому розділі, що забезпечує узгодженість між предметною галуззю та реалізованою програмою.

Важливо, що інтерфейс спроектовано з урахуванням реальних умов роботи: часті дії зроблено швидкими, стан замовлень і зміни — наочним, а помилкові дії — контрольованими. Завдяки цьому програма не лише реалізує необхідний функціонал, а й залишається зручною для працівників, які користуються нею багаторазово протягом зміни, що відповідає сформульованим раніше вимогам щодо зручності використання.

Перевірка не обмежується позитивними сценаріями: окремо тестуються помилкові ситуації, оскільки саме коректна обробка помилок відрізняє надійну систему від крихкої. Перевіряється, що система відхиляє замовлення з некоректними даними, не дозволяє виконувати дії без відповідної ролі та повертає зрозумілі повідомлення замість технічних винятків. Такий підхід підтверджує, що захищеність і цілісність даних забезпечуються не лише в очікуваних умовах, а й за спроб некоректного використання системи.

Результати тестування фіксуються та слугують основою для подальшого супроводу. Наявний набір перевірок дозволяє після кожної зміни в коді швидко переконатися, що раніше реалізована функціональність не порушена, що особливо важливо під час розвитку системи. Таким чином, тестування розглядається не як

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		62

разова дія перед захистом, а як постійна складова процесу розробки, що підтримує якість програмного комплексу впродовж його життєвого циклу.

Висновки до розділу 3

У третьому розділі описано порядок користування програмою. Розглянуто встановлення та налаштування системи з використанням контейнеризованої бази даних і автоматичних міграцій, вхід до системи та структуру інтерфейсу, а також роботу оператора з оформлення замовлення, керування статусами й призначення кур'єра, роботу кур'єра через Telegram-бот, аналітику робочих змін та адміністрування довідників.

Виконано тестування реалізованого функціоналу із застосуванням модульних та інтеграційних тестів і ручної перевірки API. Підтверджено коректність ключових бізнес-сценаріїв та правильність розмежування прав доступу, що свідчить про працездатність і практичну придатність розробленого програмного комплексу.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		63

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставленої мети — розроблено алгоритми роботи та реалізовано програмний комплекс веб-орієнтованої системи управління замовленнями для сервісу доставки, який автоматизує приймання, обробку та супроводження замовлень у межах внутрішньої операційної діяльності сервісу. Усі визначені у вступі завдання виконано в повному обсязі.

Розроблена система забезпечує автентифікацію працівників за ролями, ведення каталогу товарів, категорій та акцій, оформлення замовлень оператором з автоматичним розрахунком вартості й знижки, контроль статусів виконання, призначення кур'єрів та роботу кур'єра через Telegram-бот, облік робочих змін з автоматичним закриттям, а також формування аналітики за результатами зміни. Реалізований функціонал повністю відповідає сформульованому завданню.

Систему побудовано на сучасному технологічному стеку з дотриманням принципів чистої архітектури, що забезпечує надійність, захищеність доступу та цілісність даних. Практична цінність роботи полягає в тому, що отриманий програмний комплекс може використовуватися як основа операційної системи невеликого сервісу доставки, замінюючи розрізнені ручні дії єдиним інструментом і надаючи керівнику фактичні показники роботи для ухвалення рішень.

Завдяки модульній організації система відкрита до подальшого розвитку без переписування ядра. Перспективними напрямками вдосконалення є розширення автоматичної диспетчеризації та маршрутизації кур'єрів, ведення повної історії зміни статусів замовлення, впровадження інтеграцій з платіжними сервісами та засобами сповіщення клієнтів, а також підвищення рівня автоматизованого тестового покриття. Реалізація цих напрямів дозволить розвинути систему до повноцінної промислової платформи керування доставкою.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		64

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Державний університет «Житомирська політехніка». Методичні рекомендації до виконання кваліфікаційної роботи для здобувачів освітнього ступеня «бакалавр» спеціальності 121 «Інженерія програмного забезпечення». Житомир, 2024. 48 с.
2. Spring Boot Reference Documentation. Version 3.5. VMware Tanzu, 2025. URL: <https://docs.spring.io/spring-boot/3.5/documentation.html> (дата звернення: 07.02.2026).
3. Spring Framework Reference Documentation. Version 6.2. VMware Tanzu, 2025. URL: <https://docs.spring.io/spring-framework/reference/> (дата звернення: 12.02.2026).
4. Spring Security Reference Documentation. Version 6.5. VMware Tanzu, 2025. URL: <https://docs.spring.io/spring-security/reference/> (дата звернення: 18.02.2026).
5. Spring Data JPA Reference Documentation. VMware Tanzu, 2025. URL: <https://docs.spring.io/spring-data/jpa/reference/> (дата звернення: 23.02.2026).
6. Apache Maven Project. Guide to Working with Multiple Modules. The Apache Software Foundation, 2024. URL: <https://maven.apache.org/guides/mini/guide-multiple-modules.html> (дата звернення: 27.02.2026).
7. PostgreSQL Global Development Group. PostgreSQL 18 Documentation. 2025. URL: <https://www.postgresql.org/docs/current/index.html> (дата звернення: 03.03.2026).
8. Docker Inc. Docker Compose Documentation. 2025. URL: <https://docs.docker.com/compose/> (дата звернення: 08.03.2026).
9. Docker Inc. Compose file reference: services. 2025. URL: <https://docs.docker.com/reference/compose-file/services/> (дата звернення: 11.03.2026).
10. OWASP Foundation. OWASP API Security Top 10. 2023. URL: <https://owasp.org/www-project-api-security/> (дата звернення: 16.03.2026).
11. OWASP Foundation. Application Security Verification Standard. Version 5.0. 2024. URL: <https://owasp.org/www-project-application-security-verification-standard/> (дата звернення: 20.03.2026).
12. OWASP Foundation. Cheat Sheet Series: Password Storage. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 25.03.2026).

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

13. OWASP Foundation. Cheat Sheet Series: JSON Web Token for Java. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html (дата звернення: 29.03.2026).
14. DataReportal. Digital 2025: Ukraine. We Are Social; Meltwater, 2025. URL: <https://datareportal.com/reports/digital-2025-ukraine> (дата звернення: 02.04.2026).
15. MDN Web Docs. HTML: HyperText Markup Language. Mozilla, 2025. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (дата звернення: 06.04.2026).
16. MDN Web Docs. CSS: Cascading Style Sheets. Mozilla, 2025. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (дата звернення: 09.04.2026).
17. Axios. Getting Started: Introduction. 2024. URL: <https://axios-http.com/docs/intro> (дата звернення: 13.04.2026).
18. Cloudinary. Java SDK Documentation. 2025. URL: https://cloudinary.com/documentation/java_integration (дата звернення: 17.04.2026).
19. The JUnit Team. JUnit 5 User Guide. 2025. URL: <https://junit.org/junit5/docs/current/user-guide/> (дата звернення: 21.04.2026).
20. Red Hat. Hibernate ORM User Guide. 2025. URL: https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html (дата звернення: 25.04.2026).
21. PostgreSQL JDBC Driver Documentation. 2025. URL: <https://jdbc.postgresql.org/documentation/> (дата звернення: 28.04.2026).
22. AtomicJar. Testcontainers Documentation. 2025. URL: <https://testcontainers.com/guides/> (дата звернення: 01.05.2026).
23. Oracle. Java SE 17 Documentation. 2024. URL: <https://docs.oracle.com/en/java/javase/17/> (дата звернення: 04.05.2026).
24. World Wide Web Consortium. Web Content Accessibility Guidelines (WCAG) 2.2. 2023. URL: <https://www.w3.org/TR/WCAG22/> (дата звернення: 07.05.2026).
25. WHATWG. HTML Living Standard. 2025. URL: <https://html.spec.whatwg.org/> (дата звернення: 10.05.2026).
26. Cloud Native Computing Foundation. Cloud Native Glossary. 2025. URL: <https://glossary.cncf.io/> (дата звернення: 12.05.2026).

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

27. GitHub Inc. Understanding GitHub Actions. 2025. URL: <https://docs.github.com/en/actions/about-github-actions/understanding-github-actions> (дата звернення: 15.05.2026).
28. JetBrains. IntelliJ IDEA Documentation. 2025. URL: <https://www.jetbrains.com/help/idea/> (дата звернення: 18.05.2026).
29. Postman Inc. Postman Learning Center: API testing. 2025. URL: <https://learning.postman.com/docs/> (дата звернення: 20.05.2026).
30. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE). Geneva: ISO, 2023. 42 p.
31. Bloch J. Effective Java. 3rd ed. Boston: Addison-Wesley, 2018. 416 p.
32. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
33. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 533 p.
34. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2003. 560 p.
35. Walls C. Spring in Action. 6th ed. Shelter Island: Manning, 2022. 520 p.
36. Cockburn A. Hexagonal Architecture (Ports and Adapters). 2005. URL: <https://alistair.cockburn.us/hexagonal-architecture/> (дата звернення: 22.05.2026).

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		67

ДОДАТКИ

У додатках наведено фрагменти вихідного коду програмного комплексу, що становлять найбільший інтерес з погляду реалізованих проєктних рішень. Повний вихідний код розміщено в репозиторії системи контролю версій.

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

ДОДАТОК А

Доменна модель замовлення

Лістинг А.1 містить доменну сутність замовлення з відображенням зв'язків засобами JPA та методом життєвого циклу для проставлення часових позначок і початкового статусу.

Лістинг А.1 – Клас *DeliveryOrder*

```
@Entity
@Table(name = "delivery_order")
@Getter @Setter @AllArgsConstructor @NoArgsConstructor
public class DeliveryOrder {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @ManyToOne @JoinColumn(name = "customer_id") private Customer customer;
    @ManyToOne @JoinColumn(name = "created_by_id") private Person createdBy;
    @ManyToOne @JoinColumn(name = "courier_id") private Person courier;
    @ManyToOne @JoinColumn(name = "shift_id") private Shift shift;
    @ManyToOne @JoinColumn(name = "promotion_id") private Promotion promotion;
    @Enumerated(EnumType.STRING) private OrderStatus status;
    private String deliveryAddress;
    private BigDecimal subtotal;
    private BigDecimal discountAmount;
    private BigDecimal totalPrice;
    private LocalDateTime deliveredAt;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<DeliveryOrderItem> items = new ArrayList<>();

    @PrePersist void prePersist() {
        LocalDateTime now = LocalDateTime.now();
        this.createdAt = now; this.updatedAt = now;
        if (this.status == null) this.status = OrderStatus.NEW;
    }
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		69

ДОДАТОК Б

Створення замовлення та розрахунок знижки

Лістинг Б.1 містить метод створення замовлення, а лістинг Б.2 — алгоритм розрахунку знижки за акцією.

Лістинг Б.1 – Метод *create* (*DeliveryOrderServiceImpl*)

```
@Override
@Transactional
public DeliveryOrderDto create(CreateDeliveryOrderRequest request, String operatorPhone) {
    Shift shift = resolveOpenShift();
    Customer customer = resolveCustomer(request);
    Person createdBy = resolveOperator(operatorPhone);
    DeliveryOrder order = new DeliveryOrder();
    order.setCustomer(customer);
    order.setCreatedBy(createdBy);
    order.setShift(shift);
    order.setDeliveryAddress(request.getDeliveryAddress());
    order.setStatus(OrderStatus.NEW);
    List<DeliveryOrderItem> items = request.getItems().stream()
        .map(itemRequest -> toOrderItem(order, itemRequest)).toList();
    order.getItems().addAll(items);
    BigDecimal subtotal = items.stream()
        .map(i -> i.getPriceSnapshot().multiply(BigDecimal.valueOf(i.getQuantity())))
        .reduce(BigDecimal.ZERO, BigDecimal::add);
    Promotion promotion = resolvePromotion(request.getPromotionId(), subtotal);
    BigDecimal discount = discountFor(promotion, subtotal);
    order.setSubtotal(subtotal);
    order.setDiscountAmount(discount);
    order.setTotalPrice(subtotal.subtract(discount));
    return deliveryOrderMapper.toDto(deliveryOrderRepository.save(order));
}
```

Лістинг Б.2 – Метод *discountFor*

```
private BigDecimal discountFor(Promotion promotion, BigDecimal subtotal) {
    if (promotion == null || promotion.getDiscountValue() == null) return BigDecimal.ZERO;
    BigDecimal value = promotion.getDiscountValue();
    BigDecimal discount = promotion.getType() == PromotionType.PERCENT
        ? subtotal.multiply(value).divide(BigDecimal.valueOf(100), 2,
        RoundingMode.HALF_UP)
        : value;
    if (discount.compareTo(BigDecimal.ZERO) < 0) discount = BigDecimal.ZERO;
    return discount.min(subtotal);
}
```

					3-ІПЗ-22-1	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТОК В

Контроль та автоматичне закриття зміни

Лістинг В.1 містить механізм автоматичного закриття простроченої зміни, а лістинг В.2 — метод обчислення статистики зміни.

Лістинг В.1 – Автозакриття зміни (ShiftServiceImpl)

```
public static final int MAX_SHIFT_HOURS = 14;

@Scheduled(fixedRate = 300_000)
@Transactional
public void autoCloseExpiredShifts() {
    shiftRepository.findFirstByStatus(ShiftStatus.OPEN)
        .filter(this::isExpired).ifPresent(this::autoClose);
}

private boolean isExpired(Shift shift) {
    return shift.getOpenedAt() != null
        &&
        shift.getOpenedAt().plusHours(MAX_SHIFT_HOURS).isBefore(LocalDateTime.now());
}
```

Лістинг В.2 – Метод getStats

```
@Override
@Transactional(readOnly = true)
public ShiftStatsDto getStats(int id) {
    Shift shift = findShift(id);
    List<DeliveryOrder> orders = deliveryOrderRepository.findByShiftId(id);
    long delivered = orders.stream().filter(o -> o.getStatus() ==
        OrderStatus.DELIVERED).count();
    BigDecimal revenue = orders.stream()
        .filter(o -> o.getStatus() != OrderStatus.CANCELLED)
        .map(this::orderTotal).reduce(BigDecimal.ZERO, BigDecimal::add);
    long billable = orders.stream().filter(o -> o.getStatus() !=
        OrderStatus.CANCELLED).count();
    BigDecimal avg = billable == 0 ? BigDecimal.ZERO
        : revenue.divide(BigDecimal.valueOf(billable), 2, RoundingMode.HALF_UP);
    ShiftStatsDto dto = new ShiftStatsDto();
    dto.setOrdersCount(orders.size());
    dto.setDeliveredCount(delivered);
    dto.setRevenue(revenue);
    dto.setAvgOrderValue(avg);
    dto.setTopProducts(topProducts(orders));
    dto.setHourly(hourlySeries(orders));
    return dto;
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

ДОДАТОК Г

Сповіщення кур'єра через Telegram

Лістинг Г.1 містить реалізацію вихідного порту сповіщення кур'єра про нове замовлення.

Лістинг Г.1 – Метод notifyOffer (TelegramCourierNotifier)

```
@Override
public void notifyOffer(Person courier, DeliveryOrder order, int offerId) {
    if (courier == null || courier.getTelegramChatId() == null) {
        log.info("[COURIER-NOTIFY] offer #{} for order #{}", offerId, order.getId());
        return;
    }
    String text = "Нове замовлення #" + order.getId()
        + "\nАдреса: " + order.getDeliveryAddress()
        + "\nСума: " + order.getTotalPrice()
        + "\n\nПрийняти замовлення?";
    sender.sendMessage(courier.getTelegramChatId(), text, sender.offerKeyboard());
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

ДОДАТОК Д

Обмеження цілісності бази даних

Лістинг Д.1 містить фрагмент міграції зі створенням обмежень та індексів бази даних.

Лістинг Д.1 – Обмеження та індекси

```
ALTER TABLE delivery_order_item
    ADD CONSTRAINT chk_doi_quantity_positive CHECK (quantity >= 1);
ALTER TABLE cart_item
    ADD CONSTRAINT chk_cart_item_quantity_positive CHECK (quantity >= 1);

CREATE UNIQUE INDEX uq_person_cell_number ON person(cell_number);
CREATE UNIQUE INDEX uq_customer_phone ON customer(phone);

CREATE INDEX idx_delivery_order_courier_id ON delivery_order(courier_id);
CREATE INDEX idx_delivery_order_created_at ON delivery_order(created_at DESC);
CREATE INDEX idx_shift_status ON shift(status);
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

ДОДАТОК Е

REST-контролер замовлень

Лістинг Е.1 містить контролер замовлень, доступний адміністратору й оператору, з операціями перегляду, створення, зміни статусу та призначення кур'єра.

Лістинг Е.1 – Клас *DeliveryOrderController*

```
@RestController
@RequestMapping("/api/orders")
@PreAuthorize("hasAnyRole('ADMIN', 'OPERATOR')")
@RequiredArgsConstructor
public class DeliveryOrderController {
    private final DeliveryOrderService deliveryOrderService;
    private final CourierDispatchService courierDispatchService;

    @GetMapping
    public ResponseEntity<List<DeliveryOrderDto>> findAll() {
        return ResponseEntity.ok(deliveryOrderService.findAll());
    }

    @PostMapping
    public ResponseEntity<DeliveryOrderDto> create(
        @Valid @RequestBody CreateDeliveryOrderRequest request,
        @AuthenticationPrincipal PersonalDetails principal) {
        String phone = principal != null ? principal.getUsername() : null;
        return ResponseEntity.ok(deliveryOrderService.create(request, phone));
    }

    @PatchMapping("/{id}/status")
    public ResponseEntity<DeliveryOrderDto> updateStatus(
        @PathVariable int id, @Valid @RequestBody UpdateOrderStatusRequest request) {
        return ResponseEntity.ok(deliveryOrderService.updateStatus(id,
            request.getStatus()));
    }

    @PatchMapping("/{id}/courier")
    public ResponseEntity<DeliveryOrderDto> assignCourier(
        @PathVariable int id, @Valid @RequestBody AssignCourierRequest request) {
        return ResponseEntity.ok(deliveryOrderService.assignCourier(id,
            request.getCourierId()));
    }
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		74

ДОДАТОК Ж

Контролер кур'єрських операцій

Лістинг Ж.1 містить контролер кур'єрських операцій, доступний лише ролі COURIER; його методи викликаються під час обробки дій кур'єра, що надходять із Telegram-бота.

Лістинг Ж.1 – Клас CourierController

```
@RestController
@RequestMapping("/api/courier")
@PreAuthorize("hasRole('COURIER')")
@RequiredArgsConstructor
public class CourierController {
    private final DeliveryOrderService deliveryOrderService;
    private final PersonService personService;

    @GetMapping("/deliveries")
    public ResponseEntity<List<DeliveryOrderDto>> getMyDeliveries(
        @AuthenticationPrincipal PersonalDetails principal) {
        int courierId = personService.findByCellNumber(principal.getUsername()).getId();
        return ResponseEntity.ok(deliveryOrderService.findByCourierId(courierId));
    }

    @PatchMapping("/deliveries/{id}/start")
    public ResponseEntity<DeliveryOrderDto> startDelivery(
        @PathVariable int id, @AuthenticationPrincipal PersonalDetails principal) {
        int courierId = personService.findByCellNumber(principal.getUsername()).getId();
        return ResponseEntity.ok(deliveryOrderService.startDelivery(id, courierId));
    }

    @PatchMapping("/deliveries/{id}/confirm")
    public ResponseEntity<DeliveryOrderDto> confirmDelivery(
        @PathVariable int id, @AuthenticationPrincipal PersonalDetails principal) {
        int courierId = personService.findByCellNumber(principal.getUsername()).getId();
        return ResponseEntity.ok(deliveryOrderService.confirmDelivery(id, courierId));
    }

    @PatchMapping("/deliveries/{id}/problem")
    public ResponseEntity<DeliveryOrderDto> reportProblem(@PathVariable int id,
        @Valid @RequestBody ReportDeliveryProblemRequest request,
        @AuthenticationPrincipal PersonalDetails principal) {
        int courierId = personService.findByCellNumber(principal.getUsername()).getId();
        return ResponseEntity.ok(deliveryOrderService.reportProblem(id, courierId,
            request.getReason()));
    }
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		75

ДОДАТОК И

Адміністративний контролер

Лістинг И.1 містить фрагмент адміністративного контролера з операціями створення товару, додавання працівника та зміни його ролі.

Лістинг И.1 – Фрагмент класу AdminController

```
@RestController
@RequestMapping("/api/admin")
@PreAuthorize("hasRole('ADMIN')")
@RequiredArgsConstructor
public class AdminController {
    private final ProductService productService;
    private final PersonService personService;
    private final AuthCodeService authCodeService;

    @PostMapping(value = "/products", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    @ResponseStatus(HttpStatus.CREATED)
    public void createProduct(@RequestParam String name, @RequestParam String description,
        @RequestParam Double price, @RequestParam int productId,
        @RequestParam boolean available, @RequestParam(required = false) MultipartFile
image) {
        ProductType type = productService.findById(productId);
        productService.save(new ProductDto(name, description, price, type, image, null,
available));
    }

    @PostMapping("/users")
    @ResponseStatus(HttpStatus.CREATED)
    public Map<String, Object> createUser(@Valid @RequestBody PersonDto dto) {
        personService.save(dto);
        authCodeService.issueVerificationCode(dto.getCellNumber());
        return Map.of("phone", dto.getCellNumber(), "sent", true);
    }

    @PatchMapping("/users/{id}/role")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void changeUserRole(@PathVariable int id, @Valid @RequestBody ChangeRoleRequest
request) {
        personService.changeRole(id, request.getRole());
    }
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

ДОДАТОК К

Конфігурація безпеки

Лістинг К.1 містить конфігурацію безпеки: безсесійний режим, відкриті публічні маршрути та додавання JWT-фільтра до ланцюга фільтрів.

Лістинг К.1 – Клас SecurityCfg

```
@Configuration
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityCfg {
    private final JwtAuthenticateFilter jwtAuthenticateFilter;

    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
            .sessionManagement(s ->
s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .csrf(AbstractHttpConfigurer::disable)
            .formLogin(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(r -> r
                .requestMatchers("/", "/login", "/app/**", "/assets/**", "/src/**",
                    "/api/auth/request-code", "/api/auth/verify-code").permitAll()
                .anyRequest().authenticated())
            .addFilterBefore(jwtAuthenticateFilter,
UsernamePasswordAuthenticationFilter.class)
            .build();
    }
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		77

ДОДАТОК Л

HTTP-клієнт фронтенду

Лістинг Л.1 містить налаштування HTTP-клієнта Axios із перехоплювачами, що додають токен доступу та оновлюють його у разі помилки авторизації.

Лістинг Л.1 – HTTP-клієнт (http.js)

```
const http = axios.create({ baseURL: "/api", timeout: 9000 });

http.interceptors.request.use((config) => {
  const token = getToken();
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

http.interceptors.response.use(
  (response) => response,
  async (error) => {
    const original = error.config;
    if (error.response?.status === 401 && original && !original._retry) {
      original._retry = true;
      try {
        const token = await refreshAccessToken();
        original.headers.Authorization = `Bearer ${token}`;
        return http(original);
      } catch { redirectToLogin(); }
    } else if (error.response?.status === 401) {
      redirectToLogin();
    }
    return Promise.reject(error);
  }
);
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		78

ДОДАТОК М

Автентифікація за одноразовим кодом

Лістинг М.1 містить запит одноразового коду, а лістинг М.2 — його перевірку й видачу токенів.

Лістинг М.1 – Method requestCode (AuthService)

```
@Transactional
public Map<String, Object> requestCode(String phone) {
    Person person = findActivePerson(phone);
    String code = generateCode();
    AuthCode authCode = new AuthCode();
    authCode.setCellNumber(person.getCellNumber());
    authCode.setCodeHash(passwordEncoder.encode(code));
    authCode.setCreatedAt(LocalDateTime.now());
    authCode.setExpiresAt(LocalDateTime.now().plusMinutes(CODE_TTL_MINUTES));
    authCode.setConsumed(false);
    authCodeRepository.save(authCode);
    smsCodeSender.sendCode(person.getCellNumber(), code);
    return Map.of("sent", true, "codeTtlSeconds", CODE_TTL_MINUTES * 60);
}

private String generateCode() {
    return String.valueOf(100000 + secureRandom.nextInt(900000));
}
```

Лістинг М.2 – Method verifyCode

```
@Transactional
public Map<String, Object> verifyCode(String phone, String code, HttpServletRequest
request) {
    Person person = findActivePerson(phone);
    AuthCode authCode = authCodeRepository

.findFirstByCellNumberAndConsumedFalseOrderByCreatedAtDesc(person.getCellNumber())
    .orElseThrow(() -> new AccessDeniedException("Code not found"));
    if (authCode.getExpiresAt().isBefore(LocalDateTime.now()))
        throw new AccessDeniedException("Code expired");
    if (!passwordEncoder.matches(code, authCode.getCodeHash()))
        throw new AccessDeniedException("Invalid code");
    authCode.setConsumed(true);
    authCodeRepository.save(authCode);
    PersonalDetails details = new PersonalDetails(person);
    String accessToken = jwtService.generateJwtAccessToken(details);
    jwtService.generateJwtRefreshToken(details, request);
    return Map.of("accessToken", accessToken, "role", person.getRole().name());
}
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		79

ДОДАТОК Н

Фрагмент схеми бази даних

Лістинг Н.1 містить фрагмент початкової міграції зі створенням основних таблиць системи.

Лістинг Н.1 – Фрагмент схеми (міграція VI)

```
CREATE TABLE person (  
    id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
    name varchar(255),  
    cell_number varchar(255) NOT NULL UNIQUE,  
    role varchar(255) NOT NULL,  
    is_verified boolean NOT NULL DEFAULT false,  
    telegram_chat_id bigint,  
    courier_status varchar(20) NOT NULL DEFAULT 'OFFLINE'  
);  
  
CREATE TABLE product (  
    id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
    name varchar(255) NOT NULL,  
    description varchar(255) NOT NULL,  
    price double precision NOT NULL,  
    type_id integer REFERENCES type(id),  
    image_public_id varchar(255),  
    is_available boolean DEFAULT true,  
    deleted boolean NOT NULL DEFAULT false  
);  
  
CREATE TABLE shift (  
    id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
    status varchar(20) NOT NULL DEFAULT 'OPEN',  
    opened_by_id integer REFERENCES person(id),  
    closed_by_id integer REFERENCES person(id),  
    opened_at timestamp(6) DEFAULT CURRENT_TIMESTAMP,  
    closed_at timestamp(6)  
);
```

					3-ІПЗ-22-1	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		80