

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:

**«Інтернет-магазин квітів та супутніх подарунків з
конструктором композицій та сервісом доставки
кур'єром»**

Виконав студент групи ІПЗ-22-4
СКОВЛЮК Матвій Сергійович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

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

«16» липня 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти **СКОВЛЮК Матвій Сергійович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Інтернет-магазин квітів та супутніх подарунків з конструктором композицій та сервісом доставки.»**,

затверджена Наказом закладу вищої освіти від **«16» липня 2026 р., № 396/с**

Вихідні дані для роботи: **вимоги до організації дистанційної торгівлі, функціональні можливості платформ електронної комерції, архітектурні рішення, інформаційні моделі та програмні засоби для створення інтернет-магазину квітів з конструктором композицій і сервісом доставки.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	16.07.2026р.	16.07.2026р
2	Власенко О.В.	16.07.2026р	16.07.2026р
3	Власенко О.В.	16.07.2026р	16.07.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра виконана на тему «Інтернет-магазин квітів та супутніх подарунків з конструктором композицій та сервісом доставки». Пояснювальна записка містить 105 сторінок, 96 рисунка, 33 таблиці, список із 30 використаних джерел та 3 додатки.

Метою роботи є розробка вебзастосунку інтернет-магазину для персоналізації замовлень та автоматизації логістичних процесів у сфері флористики.

У роботі застосовано методи системного аналізу предметної області, об'єктно-орієнтованого проєктування, моделювання реляційної бази даних та функціонального тестування програмного забезпечення.

У результаті розроблено архітектуру та реалізовано вебзастосунок, ядро якого становить інтерактивний конструктор букетів. Система підтримує динамічне оновлення вартості залежно від обраних інгредієнтів. Серверну частину реалізовано з урахуванням логіки обробки замовлень, інтеграції з платіжним шлюзом LiqPay та функціоналом сповіщення кур'єрської служби в режимі реального часу.

Практичне значення роботи полягає в можливості впровадження платформи підприємствами у сфері роздрібної торгівлі квітами для оптимізації процесів логістики та підвищення зручності персоналізації замовлень клієнтами.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, ВЕБПЛАТФОРМА, ДОСТАВКА, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН.

					ІПЗ.КР.Б-121-26-ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата	Інтернет-магазин квітів та супутніх подарунків з конструктором композицій та сервісом доставки				Літ.	Арк.	Аркушів	
Розроб.		Сковлюк М.С.										
Перевір.		Власенко О.В.								2	166	
									Житомирська політехніка, група ІПЗ-22-4			
Н. контр.		Тичина О.П.										
Зав. каф.		Вакалюк Т.А.										

ABSTRACT

This bachelor's degree final project is on the topic 'An online shop for flowers and related gifts, featuring a flower arrangement designer and a delivery service'. The explanatory note comprises 105 pages, 96 figures, 33 tables, a list of 30 references and 3 appendices.

The aim of the project is to develop a web application for an online shop to personalise orders and automate logistics processes in the field of floristry.

The thesis employs methods of domain analysis, object-oriented design, relational database modelling and functional software testing.

As a result, an architecture was developed and a web application was implemented, the core of which is an interactive bouquet builder. The system supports dynamic price updates depending on the selected ingredients. The back-end has been implemented taking into account order processing logic, integration with the LiqPay payment gateway, and real-time courier service notification functionality.

The practical significance of this work lies in the ability of businesses in the floral retail sector to implement the platform to optimise logistics processes and enhance the convenience of order personalisation for customers.

KEYWORDS: AUTOMATION, WEB PLATFORM, DELIVERY, E - COMMERCE, ONLINE SHOP, WEB PLATFORM.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс.

БД – база даних.

CRUD (Create, Read, Update, Delete) – набір базових операцій для керування даними.

DTO (Data Transfer Object) – об'єкт передачі даних між підсистемами.

EJS (Embedded JavaScript Templates) – засіб шаблонізації для створення динамічних HTML-сторінок.

HTTP / HTTPS – протокол передавання гіпертексту (з підтримкою шифрування).

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

JWT (JSON Web Token) – відкритий стандарт для створення токенів доступу.

MVC (Model-View-Controller) – схема розділення логіки додатка на три компоненти.

ORM (Object-Relational Mapping) – технологія програмування, що зв'язує бази даних із концепціями об'єктно-орієнтованих мов.

ПЗ – програмне забезпечення.

PCI DSS (Payment Card Industry Data Security Standard) – стандарт безпеки даних індустрії платіжних карток.

СУБД – система управління базами даних.

WCAG (Web Content Accessibility Guidelines) – настанови з доступності вебконтенту.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						5

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	9
1.1. Загальна характеристика проблеми	9
1.2. Аналіз аналогів програмного продукту.....	12
1.3. Вибір архітектури системи	15
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	19
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНТЕРНЕТ-МАГАЗИНУ	24
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	24
2.2. Розробка бази даних системи	30
2.3. Проєктування та реалізація алгоритмів роботи системи.....	45
Висновки до другого розділу.....	53
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНТЕРНЕТ- МАГАЗИНОМ.....	55
3.1. Проєктування варіантів використання та архітектури системи	55
3.2. Структура інтерфейсу вебплатформи.....	58
3.3. Тестування вебплатформи	97
Висновки до третього розділу	99
ВИСНОВКИ	101
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	103
ДОДАТКИ.....	106

ВСТУП

Актуальність теми зумовлена стрімким переходом роздрібної торгівлі у сферу електронної комерції та зростаючим попитом на сервіси швидкої доставки подарунків. На ринку представлена велика кількість пропозицій від квіткових магазинів, проте процес вибору букета часто становить складну задачу, оскільки готові варіанти в каталогах не завжди відповідають індивідуальним побажанням замовника. Клієнти все частіше шукають можливості для персоналізації товару, але більшість платформ мають обмежений функціонал, що не дозволяє самостійно корегувати склад композиції. Окрім того, бракує надійних інструментів для координації кур'єрської доставки. Тому створення інтернет-магазину з конструктором квіткових композицій та сервісом доставки є значущим завданням, що дозволить поліпшити взаємодію з покупцями.

У межах кваліфікаційної роботи розроблено вебзастосунок для інтернет-магазину квітів FlowersGo. Система забезпечує функціонування інтерактивного конструктора букетів, керування каталогом товарів та оформлення замовлень, інтеграцію з платіжними сервісами, адміністрування складських залишків, контроль статусу доставки та технічну підтримку користувачів.

Метою роботи є проектування та реалізація вебзастосунку інтернет-магазину для персоналізації замовлень через конструктор композицій та автоматизації логістичних процесів у сфері флористики.

Завдання роботи включають аналіз предметної області та наявних сервісів доставки квітів, визначення функціональних вимог до системи, обґрунтування вибору архітектури та інструментальних засобів, проектування структури програмних компонентів і реляційної бази даних, реалізацію керування товарами, замовленнями та доступом користувачів, а також проведення функціонального тестування вебзастосунку.

Об'єктом дослідження є процеси функціонування систем дистанційної торгівлі та архітектура цифрової інфраструктури для забезпечення розташування флористичної продукції у онлайн-середовищі

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						7

Предметом дослідження є програмні інструменти, архітектурні підходи та вебтехнології, необхідні для розробки системи, яка забезпечує функціонування інтерактивного конструктора композицій, автоматизацію обробки замовлень та логістику кур'єрської доставки.

У роботі використано методи системного та порівняльного аналізу, об'єктно-орієнтованого проєктування, моделювання реляційної бази даних та функціонального тестування. Інформаційну базу дослідження становлять результати аналізу ринкових сервісів, офіційна документація технологій і матеріали, отримані під час розроблення програмного продукту.

Практичне значення одержаних результатів полягає у створенні функціонального вебзастосунку, який може використовуватися підприємствами у сфері роздрібної торгівлі квітами для оптимізації логістики та підвищення зручності персоналізації замовлень.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						8

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Загальна характеристика проблеми

Аналіз сфери онлайн-замовлення квітів та їх доставки дозволив визначити основні проблеми, які необхідно врахувати під час розробки FlowersGo.

Вебзастосунок, що розробляється, орієнтується на потреби кількох груп користувачів: клієнтів, які обирають і оплачують букет; працівників складу, що комплектують замовлення; кур'єрів, які доставляють їх за адресою; а також адміністраторів, які керують каталогом, цінами, замовленнями та модерацією контенту. На багатьох існуючих сайтах квіткових магазинів процеси перегляду каталогу, оформлення замовлення, оплати, відстеження статусу доставки та взаємодії зі службою підтримки реалізовані на різних сторінках або працюють неузгоджено, що ускладнює процес покупки, особливо під час термінових замовлень.

Головною проблемою, що потребує вирішення, є відсутність цілісного середовища для замовлення квітів. Розроблювана система має забезпечувати централізоване збереження даних про товари, замовлення, користувачів і статуси їх обробки. Необхідно реалізувати зручний каталог із категоріями та пошуком, кошик, можливість створення власного букета за допомогою конструктора, прозору систему оплати (у тому числі через LiqPay [2]), що відповідає вимогам стандарту безпеки платіжних карток PCI DSS [23], а також підтримку внутрішньої логістики від підтвердження замовлення на складі до його передачі кур'єру.»

Процеси вибору букета, розрахунку вартості, підтвердження оплати, зміни статусу замовлення та інформування клієнта на практиці часто реалізуються окремими засобами, які недостатньо інтегровані між собою. Унаслідок цього користувач витрачає більше часу на оформлення покупки, а магазин ризикує втратити замовлення через незрозумілий інтерфейс або відсутність актуальної інформації щодо наявності товару.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						9

Завданням випускної кваліфікаційної роботи бакалавра є розроблення програмного середовища, яке забезпечить підтримку основних бізнес-процесів інтернет-магазину квітів: від перегляду каталогу та формування замовлення до його обробки на складі, доставки й адміністрування.

Окрім технічних вимог існує й людський аспект проблеми. Квіти замовляють не лише з нагоди дня народження, побачення, вибачення чи подяки. Більшість сучасних вебресурсів пропонують готові композиції, однак не завжди надають можливість самостійно сформувати букет або оперативно отримати інформацію про поточний стан замовлення. Користувачеві важливо відчувати, що процес перебуває під контролем: коли буде зібрано букет, коли кур'єр вирушить на доставку та яким чином можна зв'язатися зі службою підтримки у разі зміни обставин. Саме тому доцільно поєднати функціональність повноцінного інтернет-магазину з елементами персоналізованого сервісу, системою сповіщень, чатом підтримки та можливістю залишати відгуки після покупки.

Для мінімізації ризиків та забезпечення стабільності бізнес-логіки в умовах реальних навантажень було обрано модульний підхід до проєктування, що дозволяє ізолювати окремі сервіси системи. Це сприяє швидкій адаптації платформи до змін вимог ринку без зупинки роботи всього інтернет-магазину.

Для перевірки коректності бізнес-логіки та маршрутів програмного інтерфейсу (API) на етапі розробки використовувався інструмент Postman [23], а для виконання HTTP-запитів між сервісами – бібліотека Axios [5]. Закладена архітектура (routes – controllers – services – models) забезпечує можливість подальшого розширення системи.

Під час розроблення проєкту особливу увагу приділено конструктору букетів і процесу обробки замовлення. Користувач повинен мати можливість формувати композицію без необхідності самостійно розраховувати її вартість або перевіряти наявність кожної складової. Аналогічно працівник складу та кур'єр повинні отримувати зрозумілий перелік завдань без виконання зайвих операцій у системі. Закладена архітектура (routes – controllers – services – models)

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				10

забезпечує можливість подальшого розширення функціональних можливостей системи шляхом додавання нових категорій товарів, способів доставки та аналітичних модулів адміністративної панелі [3].

Для прискорення виконання запитів у таблицях `orders`, `order_items` та `products` використовуються індекси за зовнішніми ключами. Окремі поля (`orders.payment_status`, `payment_deadline_at`, `admin_approved`, `courier_id`) визначають переходи між етапами обробки замовлення: без успішної оплати замовлення не передається на склад; без підтвердження адміністратора (`admin_approved = 1`) не здійснюється списання товарних залишків; без призначення кур'єра (`courier_id`) замовлення залишається в черзі диспетчера. Такий підхід разом із веденням журналу змін у таблиці `order_status_log` зменшує ризик помилок під час комплектації та забезпечує прозорість процесу для всіх учасників [19].

Довідники `statuses` і `status_transitions` винесено в окремі таблиці, що дає змогу додавати нові етапи логістики без внесення змін до програмного коду. Кожен перехід між статусами дублюється у таблиці `order_status_log` із зазначенням користувача та часу виконання операції, що забезпечує можливість відновлення повної історії замовлення під час вирішення спірних ситуацій між клієнтом, працівниками складу та кур'єром.

Вибір технологій обґрунтовано практичними міркуваннями: Node.js[20] і Express [10] для серверної логіки, EJS [10] для формування сторінок на сервері, MySQL [19] для реляційного зберігання замовлень та хмарне розгортання (Render[26], Aiven[3]) для стабільної публічної роботи системи. Відповідно до вимог ліцензійних умов [1], система забезпечує належний рівень прозорості та обліку операцій.

Обраний набір технологій відповідає функціональним вимогам FlowersGo та дозволяє реалізувати серверну логіку, роботу з базою даних і розгортання вебзастосунку.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						11

1.2. Аналіз аналогів програмного продукту

Для обґрунтування розробки вебзастосунку FlowersGo проведено порівняння наявних сервісів онлайн-продажу квітів та супутніх товарів. Під час аналізу розглянуто можливості каталогів, пошуку та фільтрації товарів, оформлення замовлень, організації доставки та персоналізації букетів.

Сфера онлайн-продажу квіткової продукції має певні особливості, пов'язані з необхідністю швидкого оформлення замовлення, вибором параметрів доставки та можливістю підібрати композицію відповідно до побажань клієнта. Тому під час розробки FlowersGo важливо врахувати функціональні можливості наявних сервісів та визначити, які з них доцільно реалізувати у власному вебзастосунку.

Для порівняння обрано три сервіси: LoraShen [16], Dicentra [7] та Flowers.ua [12]. Їхній функціонал використано для визначення основних можливостей та недоліків наявних рішень.

LoraShen орієнтований на продаж квіткових композицій та супутніх товарів. На головній сторінці представлено велике зображення квіткової продукції, а верхнє меню містить основні категорії магазину, зокрема флористику, день закоханих, подарунки, акції та оренду декору. Також передбачено пошук, кошик та вибір мовної версії сайту. Інтерфейс сайту LoraShen наведено на рис. 1.1.

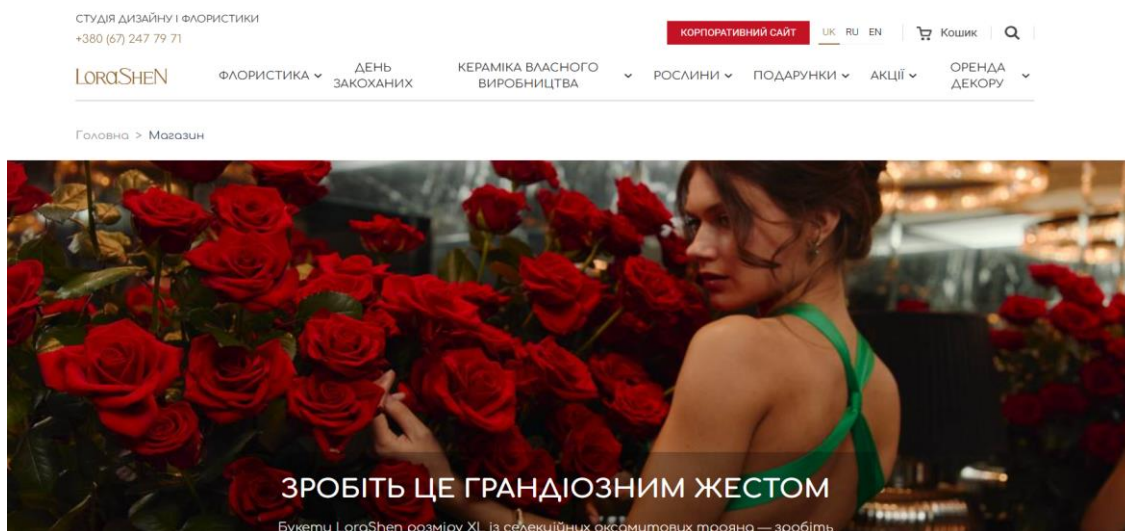


Рис. 1.1 – Інтерфейс сайту LoraShen

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				12

Основну частину головної сторінки займає візуальний блок із фотографією квітів та рекламною пропозицією. Навігація побудована за категоріями товарів і послуг, що дозволяє перейти до відповідного розділу магазину. Основний сценарій роботи користувача полягає у виборі вже сформованої композиції. Окремого інструменту для самостійного складання букета з доступних компонентів не передбачено.

Dicentra пропонує каталог квітів, готових букетів та супутніх товарів. На сайті передбачено поділ продукції за категоріями, пошук, особистий кабінет та кошик. Окремо виділено акційні пропозиції. Інтерфейс сайту Dicentra наведено на рис. 1.2.

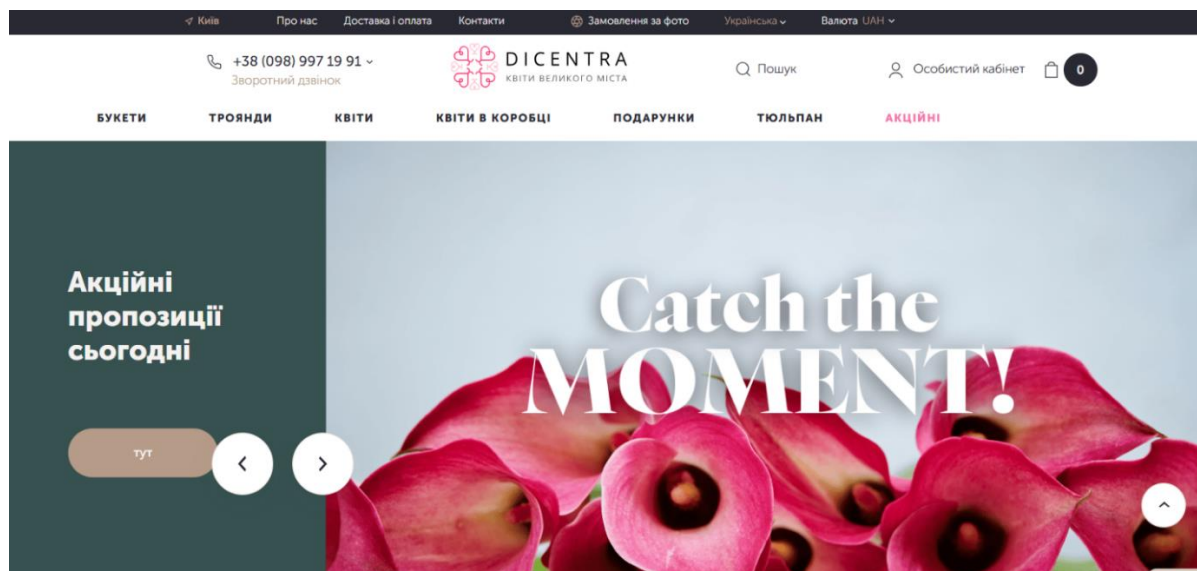


Рис. 1.2 – Інтерфейс сайту Dicentra

Головна сторінка містить навігаційне меню з основними категоріями товарів та великий блок із поточними пропозиціями. Користувач може перейти до потрібного типу продукції або переглянути акційні товари. Каталог переважно складається з уже сформованих композицій, які можна додати до замовлення. Можливість самостійно визначити склад букета з окремих компонентів не передбачена.

Flowers.ua містить каталог квітів, готових букетів та супутніх товарів. Інтерфейс передбачає пошук, кошик, перевірку статусу замовлення та навігацію за категоріями. На головній сторінці також представлено блок популярних

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				13

товарів із фотографіями, назвами, цінами та можливістю швидкого переходу до оформлення замовлення. Інтерфейс сайту Flowers.ua наведено на рис. 1.3.

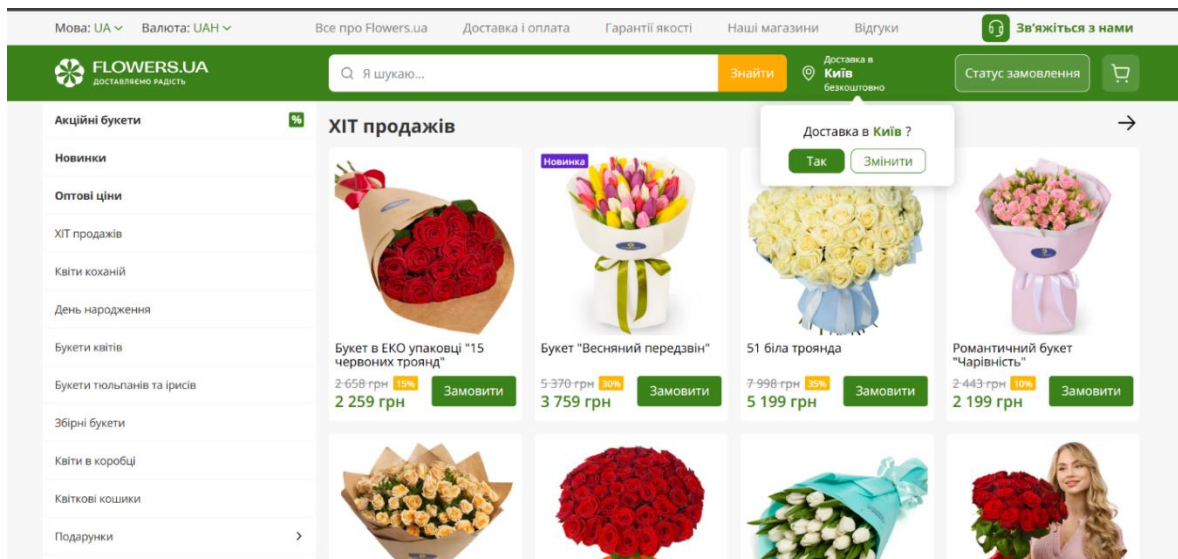


Рис. 1.3 – Інтерфейс сайту Flowers.ua

Каталог має розгалужену структуру: окремі категорії винесено в бічне меню, а товари представлені у вигляді карток із фотографією, назвою, ціною та кнопкою замовлення. Такий підхід спрощує перегляд великої кількості товарів. Користувач також може перевірити статус уже оформленого замовлення. Водночас формування власної композиції з окремих компонентів у представленому інтерфейсі не передбачено.

Узагальнене порівняння розглянутих платформ наведено в табл. 1.1.

Таблиця 1.1

Порівняльна характеристика програмних рішень

Характеристика	LoraShen	Dicentra	Flowers.ua	FlowersGo	Коментар
Візуальне представлення товарів	+	—	—	+	Акцент на представленні квіткових композицій
Швидкість обслуговування	—	+	+	+	Ключовий показник для доставки

продовж табл. 2.1

Детальна фільтрація каталогу	+	+	—	+	Необхідна для зручного вибору
Інтерактивний конструктор букетів	—	—	—	+	Унікальна конкурентна перевага
Система лояльності	+	+	+	+	Інструмент для утримання клієнтів
Історія замовлень	+	+	+	+	Обов'язковий елемент користувацького кабінету
Персоналізація пропозицій	+/-	—	+/-	+	Найвищий рівень через конструктор
Аналітичні модулі	+	+	+	+	Критичні для управління продажами
Адаптивність інтерфейсу	+	+	+	+	Стандарт для сучасного вебу

Результати порівняльного аналізу показують, що наявні на ринку рішення не повністю задовольняють специфічні потреби цільової аудиторії та вимоги сучасного флористичного ритейлу. Значна частина конкурентних сервісів орієнтована на продаж готових композицій без достатніх можливостей для їх персоналізації, що обмежує вибір користувача під час формування замовлення.

1.3. Вибір архітектури системи

Під час розробки FlowersGo необхідно визначити архітектурний підхід, який відповідає функціональним вимогам системи та забезпечує зручність подальшого супроводу. При виборі архітектури враховано особливості вебзастосунку, кількість функціональних модулів і характер взаємодії між ними [10]. Обрана структура безпосередньо впливає не лише на логіку взаємодії між внутрішніми компонентами системи, а й на перспективність подальшої

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						15

підтримки та розвитку проєкту. При формуванні архітектурних рішень для FlowersGo враховано кращі практики та патерни побудови хмарних застосунків, описані в Azure Architecture Center [17].

Зважаючи на технічну специфіку проєкту FlowersGo, а саме необхідність забезпечення високої швидкості відгуку при роботі користувача з візуальним конструктором та оновленні логістичних даних у реальному часі, було проведено аналіз сучасних підходів до побудови серверної частини. Для визначення оптимальної структури вебзастосунку розглядалися два основні архітектурні сценарії:

Використання монолітної архітектури передбачає створення системи як єдиного суцільного блоку. У такому випадку всі компоненти (користувацький інтерфейс, серверна логіка та робота реляційною базою даних MySQL) знаходяться разом і тісно пов'язані всередині одного застосунку [10].

Узагальнену архітектуру вебплатформи наведено на рис. 1.4.

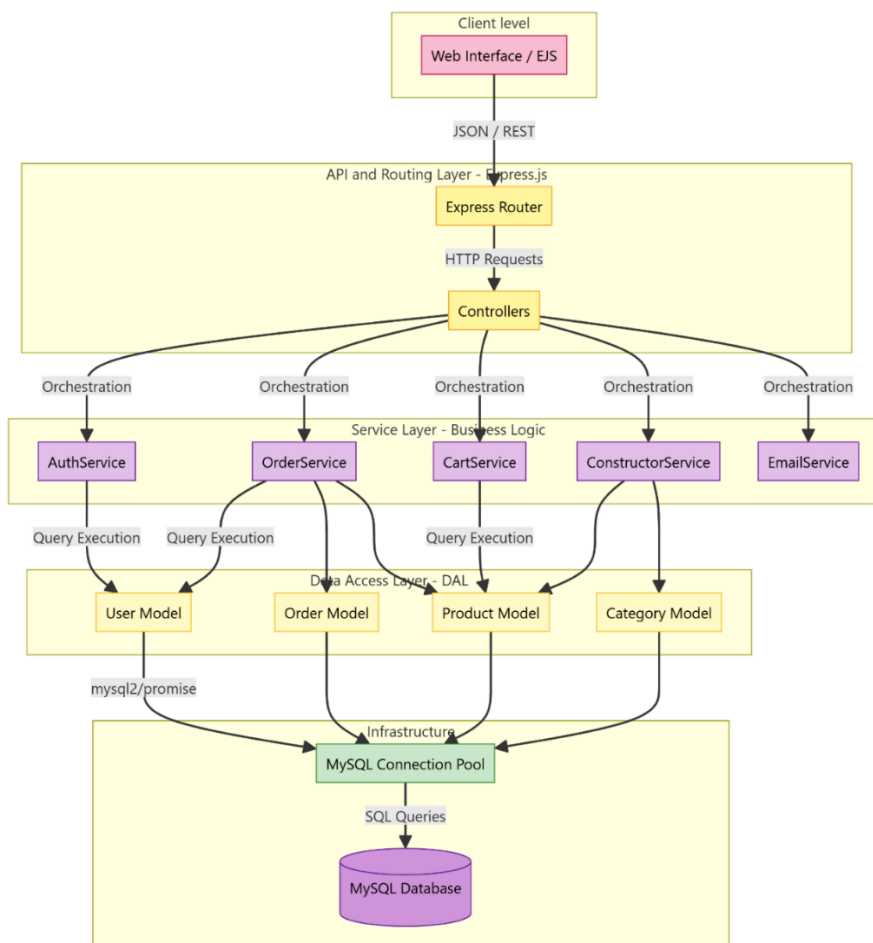


Рис. 1.4 – Діаграма монолітної архітектури

На клієнтському рівні реалізовано інтерфейс, що формується за допомогою EJS-шаблонів [10] і доповнюється JavaScript-кодом у браузері. Запити надходять на сервер на базі Express.js [10], де Express Router передає їх відповідним контролерам. Контролери координують обробку звернень і звертаються до сервісного шару, де зосереджено основну бізнес-логіку: автентифікацію, роботу з замовленнями, кошиком, конструктором букетів і надсилання електронних листів.

Сервіси використовують моделі User, Order, Product та Category для доступу до даних; SQL-запити виконуються через бібліотеку mysql2/promise з використанням пулу з'єднань до реляційної бази даних MySQL [19]. Такий поділ на шари дозволяє чітко розмежувати відповідальність між компонентами системи та спрощує подальший супровід і доопрацювання застосунку [10].

У робочому середовищі база MySQL 8.0 розміщена на хмарній платформі Aiven [4], серверна частина застосунку працює на Render.com [26], а доступ користувачів до системи здійснюється через протокол HTTPS [17].

Архітектуру розгортання та інфраструктуру наведено на рис. 1.5

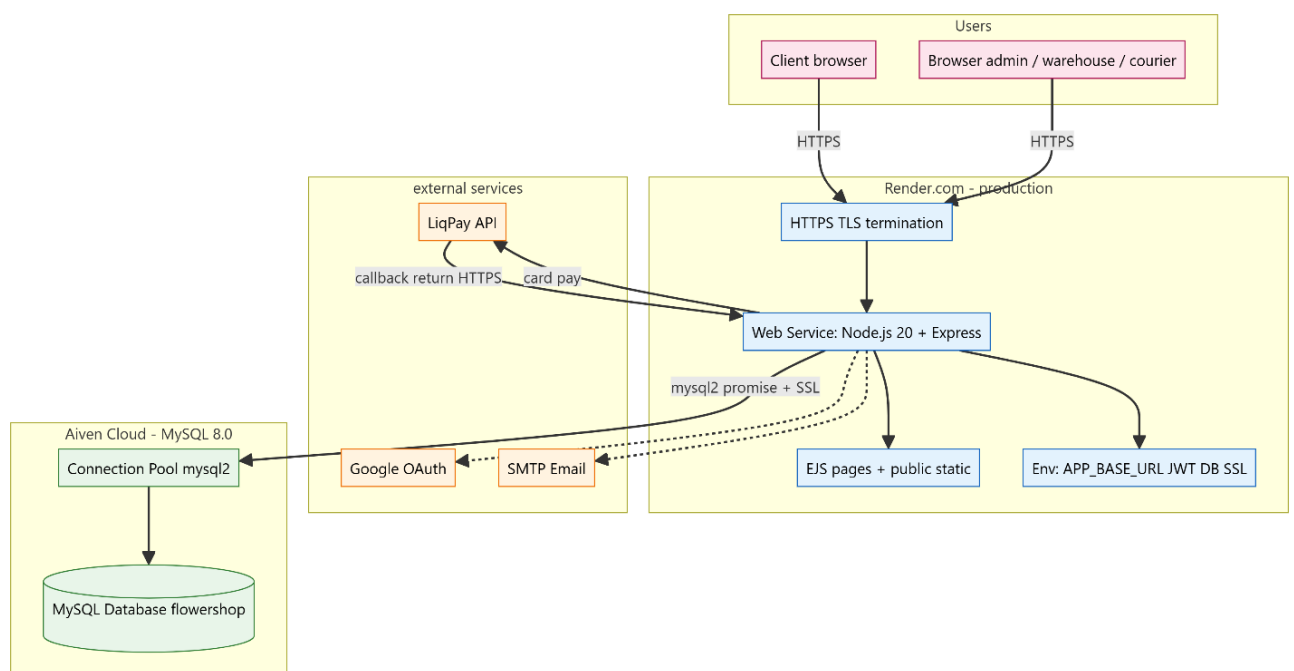


Рис .1.5 – Діаграма розгортання та інфраструктура

Користувачі (покупці, адміністратор, працівники складу та кур'єри) працюють із застосунком через веббраузер; усі звернення до сервера

виконуються за протоколом HTTPS [17], що забезпечує захищену передачу даних між клієнтом і сервером.

На платформі Render.com розгорнуто вебсервіс на Node.js [20], з використанням Express, який обробляє запити, формує HTML-сторінки на EJS [10], віддає статичні ресурси та JSON-відповіді API. Налаштування роботи застосунку, зокрема адреса сайту, параметри JWT і дані підключення до бази, задаються через змінні оточення Render, що дозволяє змінювати конфігурацію без внесення змін у вихідний код.

Окремо від сервера на Render розміщено базу даних MySQL 8.0 на платформі Aiven [3]. Зв'язок між застосунком і базою здійснюється через mysql2/promise з використанням SSL-з'єднання, що забезпечує безпечний обмін інформацією в мережі Інтернет. Крім того, до системи підключено зовнішні сервіси: LiqPay для прийому онлайн-оплати, Google OAuth для входу через обліковий запис Google та SMTP для надсилання електронних листів. Після оплати через LiqPay результат повертається на адресу застосунку на Render, після чого у базі даних оновлюється статус платежу. Запропонована діаграма показує, що FlowersGo працює як цілісна система, яка поєднує хмарний сервер, керовану базу даних і сторонні API в одному робочому середовищі.

Рольову модель користувачів FlowersGo та основні функціональні модулі, до яких вони мають доступ у системі, наведено на рис. 1.6.

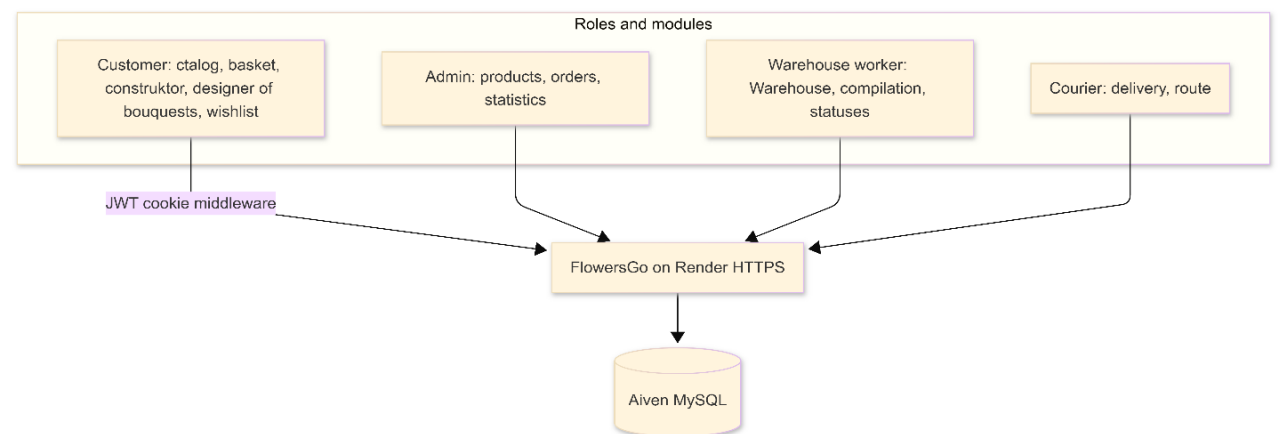


Рис. 1.6 – Діаграма рольової моделі користувачів

Система передбачає чотири основні ролі авторизованих користувачів, кожна з яких має відповідний набір функціональних можливостей. Клієнт

працює з каталогом товарів, кошиком, конструктором букетів та списком обраного. Адміністратор керує товарами, замовленнями та переглядає статистичні дані. Працівник складу обробляє замовлення, займається комплектацією та зміною відповідних статусів. Кур'єр працює з доставкою та маршрутами.

Усі звернення від різних ролей надходять до серверної частини FlowersGo, розміщеної на Render і доступної через HTTPS [17].

Доступ до системи контролюється на рівні middleware: для клієнта використовується JWT-cookie [29], що забезпечує розмежування доступу авторизованих і неавторизованих користувачів та відкривати лише дозволені для них сторінки й операції.

Дані, з якими працюють усі ролі, зберігаються в базі Aiven MySQL [3], до якої звертається сервер застосунку. Таким чином, діаграма відображає розподіл функцій між учасниками системи в межах одного хмарного застосунку.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Проект FlowersGo є вебзастосунком для електронної комерції у сфері флористики, який поєднує функціональність інтернет-магазину та інтерактивного конструктора букетів. Система охоплює повний цикл роботи із замовленням: від перегляду каталогу та формування композиції до оплати, обробки на складі та доставки. З огляду на складність бізнес-логіки та необхідність подальшого розвитку платформи, було обрано архітектуру модульного багат шарового моноліту, що забезпечує баланс між простотою реалізації та масштабованістю [10, 24].

Вибір Node.js [20] зумовлений необхідністю ефективної обробки великої кількості асинхронних запитів, що виникають під час роботи інтерактивного конструктора букетів, системи сповіщень та одночасного оформлення замовлень кількома користувачами. Подієво-орієнтована модель Node.js забезпечує високу пропускну здатність за відносно низького споживання ресурсів, що критично для

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						19

сервісів із реальним часом взаємодії. MySQL [19] обрано як реляційну СУБД через потребу в надійних транзакціях для забезпечення цілісності даних під час оформлення замовлень, списання товарів зі складу та фіксації платежів. Відмова від складних клієнтських фреймворків (React/Vue) на користь EJS [10] є навмисним архітектурним рішенням, спрямованим на спрощення структури застосунку, зменшення часу завантаження початкової сторінки завдяки серверному рендерингу та покращення пошукової оптимізації (SEO), що особливо важливо для комерційного вебресурсу. Така комбінація технологій забезпечує оптимальну продуктивність на поточному етапі розвитку проєкту та залишає можливість для масштабування окремих модулів у майбутньому.

Серверна частина реалізована на базі Node.js [20] із використанням фреймворку Express.js. Такий вибір дозволяє ефективно обробляти велику кількість асинхронних запитів, що особливо важливо під час роботи з конструктором букетів, кошиком і оформленням замовлення. Express.js використовується для організації маршрутизації, побудови REST API, а також для серверного рендерингу сторінок і обробки запитів адміністративної частини системи.

Для зберігання даних використовується реляційна система керування базами даних MySQL [19]. Такий підхід є доцільним, оскільки предметна область містить велику кількість взаємопов'язаних сутностей, зокрема користувачів, товари, замовлення, категорії та платежі. Реляційна модель забезпечує цілісність даних і зручність реалізації транзакцій, що є критично важливим для e-commerce систем. База даних розгорнута у хмарному середовищі Aiven, а доступ до неї здійснюється через бібліотеку mysql2/promise із використанням пулу з'єднань та захищеного SSL-підключення [3]. Архітектура проєкту передбачає можливість легкої інтеграції Prisma ORM [24] для подальшої оптимізації рівня роботи з даними, що забезпечить додаткову типізацію та спростить керування міграціями схеми бази даних у майбутніх версіях системи.

Інтерфейс користувача реалізовано за допомогою засобу шаблонізації EJS [10]. Такий підхід дозволяє здійснювати серверний рендеринг сторінок і

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						20

передавати дані безпосередньо у HTML-шаблони без складної процедури збирання клієнтських скриптів. Компонентна структура шаблонів дає змогу повторно використовувати загальні елементи інтерфейсу, такі як шапка сайту, нижня частина сторінок та картки товарів. Хоча сучасні вебзастосунки часто будуються на React [26], у межах цього проєкту EJS є більш раціональним вибором через простішу архітектуру та відсутність додаткової складності клієнтської частини.

Для реалізації функціоналу системи використано набір допоміжних бібліотек. Зокрема, bcryptjs і jsonwebtoken відповідають за автентифікацію та захист паролів, multer використовується для завантаження зображень, nodemailer забезпечує надсилання електронних повідомлень, а dotenv дозволяє керувати конфігурацією через змінні середовища. Платіжна інтеграція реалізована через API LiqPay [2]. Додатково підтримується OAuth авторизація через Google [5]. Усі залежності описані у файлі package.json та встановлюються через npm [20].

Розгортання серверної частини здійснено на платформі Render.com [26], що дозволяє працювати без власної фізичної інфраструктури. Доступ до системи здійснюється через захищене HTTPS з'єднання, що відповідає сучасним вимогам безпеки вебзастосунків [17].

Для клієнтської частини достатньо сучасного веббраузера та стабільного інтернет-з'єднання зі швидкістю від 10 Мбіт/с. Рекомендовані мінімальні параметри пристрою становлять 4 ГБ оперативної пам'яті. Серверна частина працює у середовищі Node.js [20], а її продуктивність залежить від обраного тарифу хмарного провайдера.

База даних розгортається як керований екземпляр MySQL 8.0 на платформі Aiven [3] з підтримкою резервного копіювання та масштабування. За потреби система може бути розширена за рахунок реплікації або збільшення ресурсів цього екземпляра.

З точки зору безпеки використовується TLS для захисту клієнт-серверної взаємодії та SSL для з'єднання з базою даних [17]. Конфіденційні дані

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						21

зберігаються у змінних середовища, а залежності періодично оновлюються для усунення вразливостей.

Для забезпечення стабільності передбачено резервне копіювання бази даних із можливістю відновлення у разі збоїв. Також реалізується базовий моніторинг системи, який включає технічні показники (навантаження CPU, використання пам'яті, час відповіді сервера) та бізнес-показники (кількість замовлень, помилки оплати).

Процес розгортання організовано за принципами CI/CD, що включає автоматичне виконання тестів, перевірку коду та розгортання у тестове і продакшен середовище.

Для підвищення продуктивності використовується кешування HTTP-запитів та серверних даних. Статичні ресурси можуть доставлятися через CDN, а за потреби можливе використання Redis для кешування часто запитуваних даних.

У тестовому середовищі достатньо конфігурації 1 vCPU та 1–2 ГБ оперативної пам'яті. Для продукційного середовища рекомендовано від 2 vCPU і 4 ГБ RAM з можливістю подальшого масштабування залежно від навантаження.

Висновки до першого розділу

У першому розділі проведено комплексний аналіз предметної області онлайн-замовлення квітів та доставки букетів, визначено основні проблеми існуючих рішень і сформульовано вимоги до майбутньої системи FlowersGo. Встановлено, що ключовими недоліками сучасних сервісів є відсутність інтегрованого середовища для керування процесом замовлення, недостатній рівень персоналізації та обмежені можливості взаємодії між учасниками бізнес-процесу.

Проведено аналіз конкурентних рішень (LoraShen, Dicentra, Flowers.ua), який показав, що на ринку переважають або вузькоспеціалізовані преміальні сервіси, або масові платформи з обмеженою гнучкістю функціоналу. Це

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						22

підтвердило доцільність розробки власного вебзастосунку, який поєднує зручність використання, широкий спектр функцій та підтримку інтерактивного конструктора букетів.

Обґрунтовано вибір архітектури модульного багат шарового моноліту, що забезпечує баланс між простотою реалізації, продуктивністю та можливістю подальшого масштабування системи. Детально аргументовано вибір технологічного стеку: Node.js забезпечує ефективну обробку асинхронних запитів, MySQL гарантує транзакційну цілісність даних, а EJS обрано замість сучасних фронтенд-фреймворків для спрощення архітектури та покращення SEO завдяки серверному рендерингу. Визначено хмарні сервіси для розгортання (Render, Aiven) та вимоги до апаратного забезпечення.

Таким чином, результати проведеного аналізу створюють необхідну теоретичну та технологічну основу для подальшого проєктування і реалізації вебзастосунку FlowersGo. Визначені функціональні вимоги та обрана архітектура безпосередньо впливають на структуру програмних модулів, а обґрунтований технологічний стек задає чіткий напрямок для написання серверного коду, роботи з базою даних і побудови клієнтського інтерфейсу. Такий підхід забезпечує цілісність проєкту на всіх етапах розроблення — від початкового моделювання до впровадження в реальне середовище електронної комерції. Сформована теоретична база дозволяє перейти до практичного проєктування окремих компонентів системи з урахуванням виявлених вимог до продуктивності, безпеки та масштабованості.

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						23

Подана діаграма є основою для подальшого формування вимог до системи, оскільки дозволяє визначити ключові сценарії взаємодії користувачів із платформою та логіку виконання бізнес-процесів.

У системі FlowersGo визначено п'ять основних ролей користувачів, які беруть участь у бізнес-процесі обробки замовлень. Кожна роль має власний рівень доступу та набір дозволених операцій. Розподіл функціональних повноважень між ролями наведено в табл. 2.1.

Таблиця 2.1

Таблиця ролей інтернет-магазину

№	Роль користувача	Опис повноважень
1	Адміністратор	Управління товарами, користувачами, замовленнями та системними налаштуваннями
2	Працівник складу	Комплектація замовлень, контроль залишків, оновлення статусів
3	Кур'єр	Доставка замовлень та оновлення статусів доставки
4	Гість	Перегляд каталогу та інформації про товари
5	Авторизований користувач	Використання повного клієнтського функціоналу

Ролі користувачів визначають механізм авторизації та розмежування доступу до функціоналу системи FlowersGo. Вони використовуються як базова основа для реалізації контролю доступу на рівні застосунку та бізнес-логіки. Для забезпечення надійного керування ідентичністю користувачів, їхніми сесіями та метаданими в сучасних вебзастосунках доцільно використовувати спеціалізовані сервіси автентифікації, такі як Clerk [7], що надає розширені можливості керування обліковими записами та їх безпекою.

Далі наведено функціональні можливості системи для неавторизованих користувачів. Вони охоплюють базовий набір операцій, доступних без створення облікового запису. Структуру та опис відповідних функціональних модулів наведено в таблиці 2.2.

Таблиця 2.2

Функціонал сайту для неавторизованих користувачів

№	Функціональний модуль	Опис можливостей
1	Каталог	Перегляд товарів, категорій, пошук і фільтрація
2	Картка товару	Перегляд опису, ціни, наявності та відгуків
3	Конструктор букетів	Формування композицій, вибір елементів, розрахунок вартості
4	Додаткові сервіси	Кошик (cookie), Wishlist, інформація про доставку, чат підтримки

Далі наведено функціональні можливості системи для неавторизованих користувачів, які забезпечують ознайомлення з асортиментом та базовими сервісами платформи без необхідності створення облікового запису.

Розширений функціонал для зареєстрованих користувачів наведено в таблиці 2.3.

Таблиця 2.3

Функціонал сайту для авторизованих користувачів

№	Функціональний модуль	Опис можливостей
1	Особистий кабінет	Редагування профілю, зміна пароля, аватар
2	Замовлення	Оформлення, перегляд історії, скасування
3	Оплата	LiqPay, оплата при отриманні
4	Дані користувача	Збережений кошик та Wishlist у базі даних
5	Відгуки	Додавання та перегляд відгуків
6	Сповіщення	Інформування про статуси замовлень
7	Авторизація	Вхід через Google OAuth
8	Підтримка	Звернення до чату підтримки

Наявність особистого кабінету забезпечує персоналізацію роботи користувача та збереження його даних у базі даних із прив'язкою до облікового запису. Аутентифікація користувача реалізується з використанням JWT, що зберігається у cookie браузера.

Для адміністрування системи передбачено окрему адміністративну панель, яка забезпечує повний контроль над каталогом, користувачами, замовленнями та

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						26

налаштуваннями системи. Основні функції адміністратора наведено в таблиці 2.4.

Таблиця 2.4

Функціонал адміністратора

№	Модуль	Опис функцій
1	Товари	Додавання, редагування, видалення товарів
2	Категорії	Управління категоріями товарів
3	Конструктор	Налаштування елементів букетів
4	Замовлення	Модерація та зміна статусів
5	Користувачі	Блокування та управління акаунтами
6	Доставка	Призначення кур'єрів і правила доставки
7	Відгуки	Модерація відгуків
8	Аналітика	Статистика продажів
9	Система	Юридичні сторінки та налаштування

Адміністратор виконує функції повного керування системою, включаючи контроль контенту, користувачів та бізнес процесів замовлення, що забезпечує стабільну роботу платформи FlowersGo.

Працівник складу відповідає за фізичну комплектацію замовлень та оновлення їх статусів у системі. Структуру його функціональних можливостей наведено в таблиці 2.5.

Таблиця 2.5

Функціонал працівника складу

№	Функція	Опис
1	Перегляд замовлень	Список замовлень для обробки
2	Деталі замовлення	Склад, адреса, оплата
3	Комплектація	Збирання букетів та зміна статусів
4	Склад	Оновлення залишків товарів

Працівник складу виконує проміжну роль між оформленням замовлення клієнтом та передачею його кур'єру, забезпечуючи коректну підготовку товару до доставки.

Кур'єр забезпечує доставку готових замовлень кінцевому отримувачу. Основні функції кур'єра наведено в таблиці 2.6.

Таблиця 2.6

Функціонал кур'єра

№	Функція	Опис
1	Замовлення	Перегляд призначених доставок
2	Дані доставки	Адреса, контакти, коментарі
3	Статуси	Оновлення статусу доставки
4	Оплата	Підтвердження оплати при отриманні

Кур'єр завершує повний цикл обробки замовлення, забезпечуючи доставку товару клієнту та фінальне оновлення статусу в системі.

Наступним етапом є побудова об'єктно-орієнтованої структури системи FlowersGo, яка відображає логічну модель предметної області та взаємозв'язки між її основними сутностями.

У межах системи виділено ключові доменні об'єкти: користувач, роль, товар, категорія, замовлення, позиція замовлення, відгук, список бажаного, сповіщення, а також допоміжні сутності, що забезпечують реалізацію бізнес-логіки. Вони формують основу для взаємодії між функціональними модулями системи та забезпечують цілісність предметної області.

Система умовно поділяється на чотири логічні блоки: користувачі та доступ, каталог, замовлення та оплата, відгуки і підтримка. Така декомпозиція дозволяє чітко розмежувати відповідальність між підсистемами та спростити подальше проектування і реалізацію програмної архітектури.

У межах об'єктної моделі користувач є центральною сутністю, яка взаємодіє з усіма основними компонентами системи. Він може формувати замовлення, залишати відгуки, додавати товари до списку бажаного та отримувати сповіщення. Замовлення виступає узагальнювальним об'єктом, що об'єднує позиції товарів, спосіб оплати, адресу доставки та статус виконання.

Товари групуються у категорії, що забезпечує структурування каталогу та зручну навігацію. Відгуки формують зв'язок між користувачем і товаром,

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						28

реалізуючи механізм зворотного зв'язку. Список бажаного є допоміжною сутністю, призначеною для збереження обраних користувачем товарів.

Зв'язки між сутностями реалізуються за допомогою асоціації, агрегації та композиції. Композиція застосовується для залежних об'єктів, зокрема позицій замовлення, які не можуть існувати без самого замовлення. Агрегація використовується для зв'язку категорій і товарів, де категорія виконує функцію групування без жорсткої залежності життєвого циклу об'єктів. Асоціація відображає взаємодію незалежних сутностей, зокрема взаємодію користувача із замовленнями, відгуками та списком бажаного.

На рис. 2.2 представлено UML-діаграму класів FlowersGo. Вона містить основні класи системи та зв'язки між ними. Композиція використовується для залежних сутностей, зокрема позицій замовлення, повідомлень чату та кольорових варіантів. Агрегація застосовується для зв'язку товарів із категоріями, а асоціація — для незалежних об'єктів, зокрема користувачів, замовлень та відгуків.

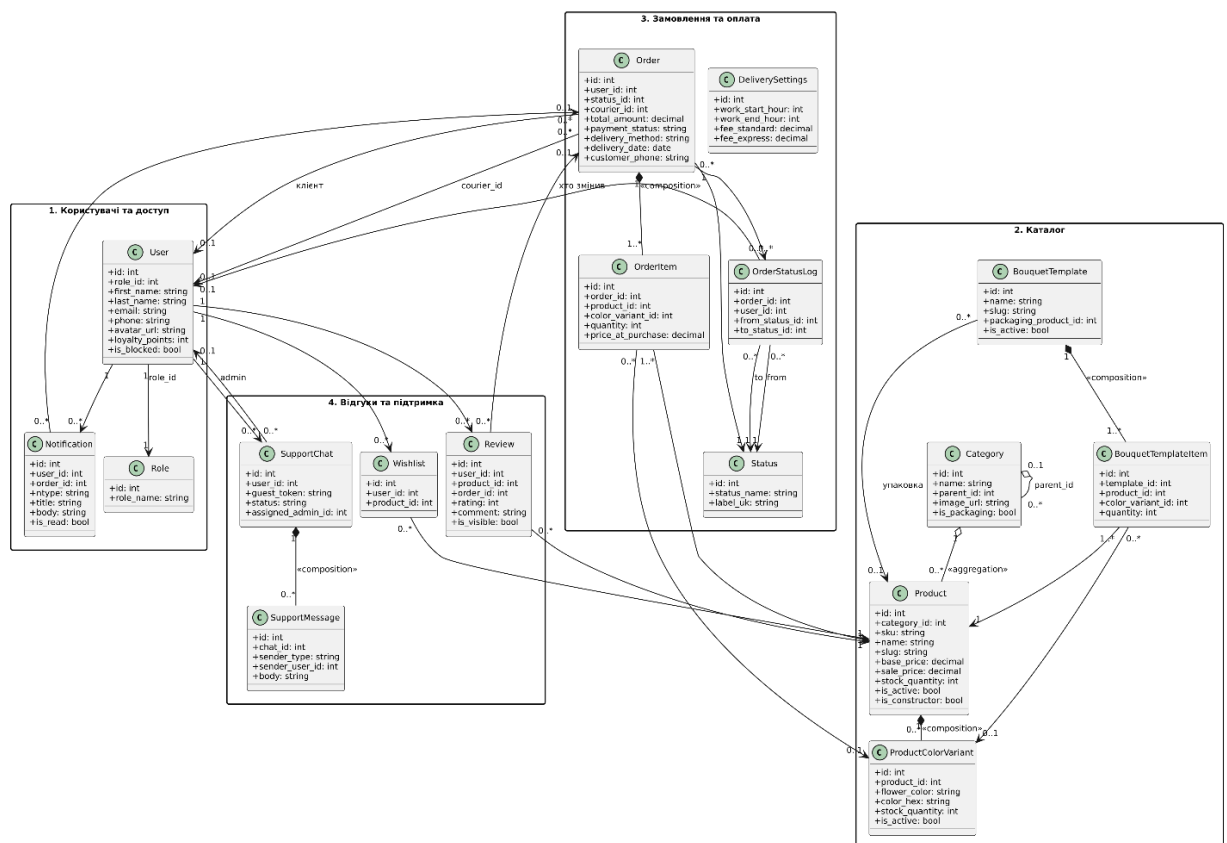


Рис. 2.2 – Діаграма класів проекту

Запропонована структура об'єктно-орієнтованої моделі дозволяє поступово розвивати платформу, додаючи нові категорії товарів, способи доставки або аналітичні модулі без порушення базової бізнес-логіки.

2.2. Розробка бази даних системи

З метою надійного збереження та ефективної обробки даних у вебзастосунку FlowersGo розроблено реляційну базу даних на основі MySQL. Її структура відображає ключові бізнес-процеси системи, зокрема роботу каталогу товарів, конструктора букетів, оформлення замовлень, оплати, доставку та взаємодію з користувачами.

Під час проєктування бази даних застосовано принципи нормалізації, що дозволило мінімізувати дублювання даних і забезпечити їх цілісність. Для організації зв'язків між сутностями використано первинні та зовнішні ключі, що гарантує узгоджену взаємодію таблиць і запобігає появі неузгоджених записів. Для залежних сутностей використовується каскадне видалення, що спрощує підтримку цілісності даних.

Основними сутностями бази даних є користувачі (users), ролі (roles), товари (products), категорії (categories), варіанти товарів, шаблони букетів, замовлення (orders) та позиції замовлення (order_items). Каталог товарів підтримує ієрархічну структуру категорій, а також можливість зберігання варіантів продукції, зокрема кольорових.

Життєвий цикл замовлення реалізовано через систему статусів, яка включає таблиці статусів та журнал змін, що дозволяє відстежувати всі етапи обробки замовлення. Історичні дані зберігаються незалежно від змін пов'язаних сутностей, що забезпечує цілісність аналітичної інформації.

Кошик користувача реалізовано на стороні клієнта та зберігається у cookies до моменту оформлення замовлення, що зменшує навантаження на сервер і спрощує роботу системи.

Для автентифікації користувачів реалізовано комбінований підхід: стандартну реєстрацію через e-mail з паролем та вхід за допомогою сервісу

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						30

Google OAuth. Розмежування доступу реалізовано на основі ролікової моделі через таблицю roles. Усі грошові значення зберігаються у копійках, що виключає похибки округлення під час фінансових розрахунків.

На рис. 2.3 наведено спрощену діаграму структури бази даних FlowersGo, яка відображає основні таблиці та зв'язки між ними. Повна фізична модель бази даних із переліком полів та ключів наведена в додатку А.

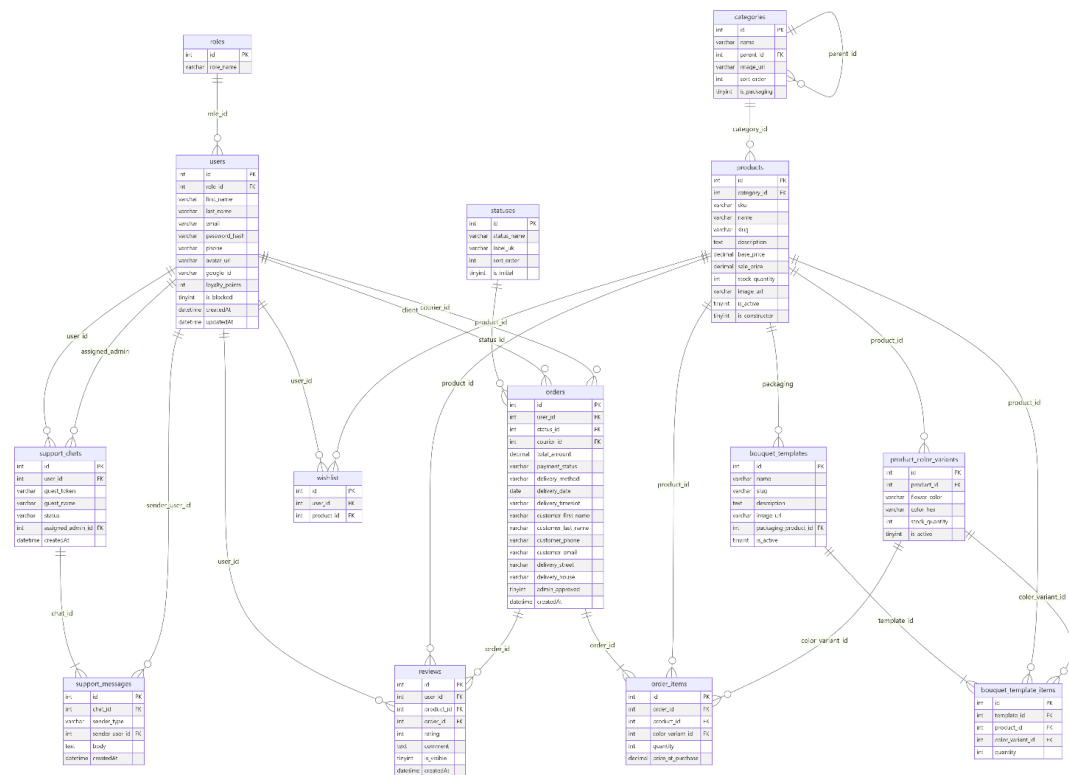


Рис. 2.3 Спрощена діаграма структури БД

Облік користувачів реалізовано через таблицю users. Ролі зберігаються у довіднику roles, який визначає рівні доступу для різних типів користувачів. Зв'язки з іншими частинами системи, такими як замовлення, відгуки, список бажаного, сповіщення та підтримка, реалізовані через зовнішні ключі. Окремо ведуться журнали змін статусів та обліку складських залишків.

Каталог побудовано на таблицях categories та products із підтримкою ієрархії категорій. Варіанти кольорів товарів зберігаються в таблиці product_color_variants. Шаблони букетів для конструктора реалізовані через таблиці bouquet_templates та bouquet_template_items. Кошик реалізується на

стороні клієнта (cookie) і не зберігається у базі даних до моменту оформлення замовлення з метою зменшення навантаження на сервер.

Основною таблицею продажів є orders, яка містить загальну інформацію про замовлення. Склад замовлення деталізується у таблиці order_items. Життєвий цикл замовлення підтримується через систему статусів, яка включає таблиці statuses, status_transitions та order_status_log.

Окремі модулі системи відповідають за оплату, доставку, відгуки, сповіщення та підтримку користувачів. Налаштування доставки, юридичні сторінки, журнали складських змін та коди відновлення пароля винесені в окремі довідкові таблиці. Аналітика адміністративної панелі реалізується на основі запитів до існуючих даних без створення окремого сховища.

Для реалізації рольової моделі доступу в системі використовується окрема таблиця roles, яка містить перелік ролей користувачів та використовується під час авторизації й розмежування прав доступу. Структуру таблиці наведено в табл. 2.9

Таблиця 2.9

Опис таблиці roles

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Унікальний ідентифікатор ролі
role_name	varchar(50)	-	-	Назва ролі (admin, courier, staff, client)

Таблиця roles є довідником ролей користувачів і використовується для реалізації рольової моделі доступу в системі.

Основні відомості про користувачів, їх персональні дані та належність до певної ролі зберігаються в таблиці users, структуру якої наведено в табл. 2.10.

Таблиця 2.10

Опис таблиці users

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Унікальний ідентифікатор
role_id	int	-	+	Посилання на roles

продовж. табл. 2.10

first_name	varchar(100)	-	-	Ім'я
last_name	varchar(100)	-	-	Прізвище
email	varchar(255)	-	-	Унікальна електронна пошта
password_hash	varchar(255)	-	-	Хеш пароля
phone	varchar(30)	-	-	Телефон
avatar_url	varchar(1024)	-	-	Аватар
google_id	varchar(255)	-	-	Google OAuth ID
loyalty_points	int	-	-	Бали лояльності
is_blocked	int	-	-	Прапорець блокування (0/1)
created_at	datetime	-	-	Створено
updated_at	datetime	-	-	Оновлено

Таблиця users містить персональні дані користувачів системи. Зв'язок із ролями реалізовано через зовнішній ключ role_id.

Для структуризації каталогу товарів використовується таблиця categories, структуру якої наведено в табл. 2.11.

Таблиця 2.11

Опис таблиці categories

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор категорії
name	varchar(255)	-	-	Назва категорії
parent_id	int	-	+	Батьківська категорія (опціонально)
image_url	varchar(1024)	-	-	Посилання на зображення
sort_order	int	-	-	Порядок сортування
is_packaging	int	-	-	Позначка упаковки (0/1)

Таблиця users містить персональні дані користувачів системи. Зв'язок із таблицею roles реалізовано через зовнішній ключ role_id, який визначає рівень доступу користувача.

Відомості про товари зберігаються в таблиці products, структуру якої наведено в табл. 2.12.

Таблиця 2.12

Опис таблиці products

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор товару
category_id	int	-	+	Категорія товару
sku	varchar(100)	-	-	Артикул
name	varchar(255)	-	-	Назва товару
description	varchar(1000)	-	-	Опис товару
sale_price	int	-	-	Ціна продажу (ціла, у копійках або в одиницях)
base_price	int	-	-	Базова ціна (ціла)
stock_quantity	int	-	-	Кількість на складі
image_url	varchar(1024)	-	-	Посилання на зображення
is_active	int	-	-	Активність товару (0/1)
is_constructor	int	-	-	Доступний у конструкторі (0/1)

На цій таблиці зберігаються товари каталогу: назва, опис, артикул, ціни, кількість на складі та інші атрибути. Поле category_id є зовнішнім ключем, що посилається на categories.id.

Для підтримки різних кольорових варіантів товарів використовується таблиця product_color_variants, опис якої наведено в табл. 2.13.

Таблиця 2.13

Опис таблиці product_color_variants

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор варіанту кольору
product_id	int	-	+	Посилання на товар
flower_color	varchar(100)	-	-	Назва кольору
color_hex	varchar(7)	-	-	HEX-код кольору (наприклад #ff0000)
stock_quantity	int	-	-	Запас для варіанту
is_active	int	-	-	Активність варіанту (0/1)

Таблиця products містить інформацію про товари каталогу, їх характеристики, ціни та залишки на складі. Зв'язок із категоріями реалізовано через поле category_id.

Для роботи конструктора букетів у системі використовуються шаблони композицій, що зберігаються в таблиці bouquet_templates, структуру якої наведено в табл. 2.14.

Таблиця 2.14

Опис таблиці bouquet_templates

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор шаблону букета
name	varchar(255)	-	-	Назва шаблону
flag	varchar(50)	-	-	Позначка/тип шаблону
description	varchar(1000)	-	-	Опис шаблону
image_url	varchar(1024)	-	-	Зображення шаблону
packaging_product_id	int	-	+	Посилання на упаковку (опц.)
is_active	int	-	-	Активність шаблону (0/1)

Таблиця bouquet_templates використовується для збереження шаблонів букетів, що застосовуються у конструкторі композицій.

Склад кожного шаблону букета описується окремими записами в таблиці bouquet_template_items, структуру якої наведено в табл. 2.15.

Таблиця 2.15

Опис таблиці bouquet_template_items

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор елемента шаблону
template_id	int	-	+	Посилання на шаблон
product_id	int	-	+	Товар у шаблоні
color_variant_id	int	-	+	Варіант кольору (опціонально)
quantity	int	-	-	Кількість одиниць у шаблоні

Поле template_id посиляється на bouquet_templates.id, product_id на products.id, а color_variant_id на product_color_variants.id.

Для керування процесом обробки замовлень використовується довідник статусів, структура якого описана в табл. 2.16.

Таблиця 2.16

Опис таблиці statuses

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор статусу
status_name	varchar(50)	-	-	Код статусу (англійською)
label_uk	varchar(100)	-	-	Назва статусу українською
sort_order	int	-	-	Порядок сортування
is_initial	int	-	-	Чи початковий статус (0/1)

Таблиця statuses містить перелік можливих статусів замовлення та використовується під час його обробки.

Допустимі переходи між статусами замовлень визначаються таблицею status_transitions, яку наведено в табл. 2.17.

Таблиця 2.17

Опис таблиці status_transitions

Назва поля	Тип даних	РК	ФК	Опис
id	int	+	-	Ідентифікатор переходу
from_status_id	int	-	+	Початковий статус

Таблиця status_transitions визначає допустимі переходи між статусами замовлення. Її використання під час обробки замовлень реалізовано через таблицю orders, опис якої наведено в табл. 2.18.

Таблиця 2.18

Опис таблиці orders

Назва поля	Тип даних	РК	ФК	Опис
id	int	+	-	Ідентифікатор замовлення
user_id	int	-	+	Клієнт (NULL для гостя)
status_id	int	-	+	Поточний статус замовлення
courier_id	int	-	+	Призначений кур'єр
total_amount	int	-	-	Загальна сума (ціла, у копійках)
payment_status	varchar(50)	-	-	Статус оплати
delivery_method	varchar(50)	-	-	Метод доставки
delivery_timeslot	varchar(100)	-	-	Слот доставки
delivery_date	datetime	-	-	Дата/час доставки
customer_first_name	varchar(100)	-	-	Ім'я отримувача
customer_last_name	varchar(100)	-	-	Прізвище отримувача
customer_phone	varchar(30)	-	-	Телефон отримувача
customer_email	varchar(255)	-	-	Email отримувача
delivery_street	varchar(255)	-	-	Вулиця доставки
delivery_house	varchar(50)	-	-	Номер будинку

продовж. табл. 2.18

delivery_flat	varchar(50)	-	-	Квартира/офіс
admin_approved	int	-	-	Підтвердження адміністратором (0/1)
created_at	datetime	-	-	Дата створення замовлення

Поля user_id, status_id та courier_id є зовнішніми ключами, що посиляються відповідно на таблиці users та statuses.

Перелік товарів, що входять до кожного замовлення, зберігається в таблиці order_items, опис якої наведено в табл. 2.19.

Таблиця 2.19

Опис таблиці order_items

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор позиції замовлення
order_id	int	-	+	Посилання на замовлення
product_id	int	-	+	Посилання на товар
color_variant_id	int	-	+	Варіант кольору (опц.)
quantity	int	-	-	Кількість одиниць
price_at_purchase	int	-	-	Ціна на момент покупки (ціла, у копіях)

Таблиця order_items містить перелік товарів, що входять до кожного замовлення, із зазначенням кількості та вартості на момент оформлення покупки. Поля order_id, product_id та color_variant_id є зовнішніми ключами, що посиляються на таблиці замовлень, товарів та варіантів кольорів.

Історію зміни статусів для цих замовлень наведено в табл. 2.20.

Таблиця 2.20

Опис таблиці order_status_log

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор запису логу

продовж. табл. 2.20

order_id	int	-	+	Посилання на замовлення
previous_status_id	int	-	+	Попередній статус
new_status_id	int	-	+	Новий статус
changed_by_user_id	int	-	+	Хто змінив статус
changed_at	datetime	-	-	Час зміни
comment	varchar(1000)	-	-	Коментар до зміни

Таблиця order_status_log забезпечує ведення історії зміни статусів замовлень, що дозволяє відстежувати всі етапи їх обробки. Як клієнти оцінюють отримані замовлення, описано в табл. 2.21.

Таблиця 2.21

Опис таблиці reviews

Назва поля	тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор відгуку
user_id	int	-	+	Автор відгуку
order_id	int	-	+	Пов'язане замовлення (опц.)
product_id	int	-	+	Товар, що оцінюється
rating	int	-	-	Оцінка 1–5
comment	varchar(1000)	-	-	Текст відгуку
is_visible	int	-	-	Публікація відгуку (0/1)
created_at	datetime	-	-	Дата створення відгуку

Структура таблиці reviews, що зберігає відгуки користувачів. Поля user_id, order_id та product_id є зовнішніми ключами, що посилаються на таблиці користувачів, замовлень та товарів. Додаткова інформація про запити на відгуки наведена в табл. 2.22.

Таблиця 2.22

Опис таблиці review_requests

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Унікальний ідентифікатор запиту на відгук
user_id	int	-	+	Користувач, який ініціював запит
order_id	int	-	+	Пов'язане замовлення (опціонально)
product_id	int	-	+	Товар, щодо якого запитують відгук (опц.)
status	varchar(50)	-	-	Статус запиту (pending/sent/reminded)
requested_at	datetime	-	-	Дата/час створення запиту
last_sent_at	datetime	-	-	Дата/час останньої відправки нагадування
admin_note	varchar(1000)	-	-	Примітка адміністратора (опц.)

Таблиця review_requests призначена для керування запитами на залишення відгуків після виконання замовлення або взаємодії з товаром. Поля user_id, order_id та product_id є зовнішніми ключами до відповідних таблиць.

Про товари, додані до списку бажань, йдеться в табл. 2.23.

Таблиця 2.23

Опис таблиці wishlist

Назва поля	тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор запису wishlist
user_id	int	-	+	Власник списку бажань
product_id	int	-	+	Товар у списку бажань

Таблиця review_requests призначена для керування запитами на залишення відгуків після виконання замовлення або взаємодії з товаром.

Про механізм сповіщень користувачів описано в табл. 2.24.

Таблиця 2.24

Опис таблиці user_notifications

Назва поля	тип даних	PK	FK	Опис
id	int	+	-	Унікальний ідентифікатор повідомлення
user_id	int	-	+	Одержувач повідомлення
message	varchar(1000)	-	-	Текст повідомлення
link	varchar(1024)	-	-	Посилання/ціль (опц.)
is_read	int	-	-	Позначка прочитано (0/1)
type	varchar(100)	-	-	Тип повідомлення (order, promo, system)
created_at	datetime	-	-	Дата/час створення повідомлення
expires_at	datetime	-	-	Дата/час автоматичного видалення/архівації (опц.)

Опис таблиці user_notifications, що містить сповіщення. Поле user_id є зовнішнім ключем до таблиці users. Деталі чатів підтримки наведено в табл. 2.25.

Таблиця 2.25

Опис таблиці support_chats

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор чату підтримки
user_id	int	-	+	Користувач (NULL для гостя)
guest_token	varchar(255)	-	-	Токен гостя (опц.)
guest_name	varchar(255)	-	-	Ім'я гостя (опц.)
assigned_admin_id	int	-	+	Призначений адміністратор
status	varchar(50)	-	-	Статус чату (open/closed)
created_at	datetime	-	-	Дата створення чату

Сковлюк М.С.

Власенко О.В.

ПІЗ.КР.Б-121-26-ПЗ

Арк.

41

продовж. табл. 2.25

guest_name	varchar(255)	-	-	Ім'я гостя (опц.)
assigned_admin_id	int	-	+	Призначений адміністратор
status	varchar(50)	-	-	Статус чату (open/closed)
created_at	datetime	-	-	Дата створення чату

Структура таблиці support_chats, що пов'язує чат із користувачем user_id та призначеним адміністратором assigned_admin_id. Про зміст повідомлень у цих чатах у табл. 2.26.

Таблиця 2.26

Опис таблиці support_messages

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор повідомлення
chat_id	int	-	+	Посилання на чат
sender_type	varchar(50)	-	-	Тип відправника (user/admin/guest)
sender_user_id	int	-	+	Посилання на користувача-відправника (опц.)
text	varchar(1000)	-	-	Текст повідомлення
created_at	datetime	-	-	Час відправлення

Опис таблиці support_messages, де поле chat_id є зовнішнім ключем до support_chats.id, а sender_user_id до users.id. Про налаштування доставки йдеться в табл. 2.27.

Таблиця 2.27

Опис таблиці delivery_settings

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор налаштування доставки
work_start_hour	varchar(5)	-	-	Початок робочого дня (HH:MM)
work_end_hour	varchar(5)	-	-	Кінець робочого дня (HH:MM)

продовж. табл. 2.27

standard_buffer_min	int	-	-	Мінімальний буфер у хвилинах
pickup_buffer_min	int	-	-	Буфер для самовивозу у хвилинах
flex_start	varchar(5)	-	-	Початок гнучкого інтервалу (HH:MM)
flex_expense	int	-	-	Додаткова вартість за flex (ціла, у копіях)
flex_pickup	int	-	-	Прапорець дозволу flex pickup (0/1)
created_at	datetime	-	-	Дата створення запису
updated_at	datetime	-	-	Дата останнього оновлення

Таблиця `delivery_settings` використовується для налаштування логіки доставки та не містить зовнішніх ключів, оскільки є довідковою системною конфігурацією.

Наступною є таблиця `legal_pages`, яка використовується для зберігання юридичної інформації та статичних сторінок сервісу.

Таблиця 2.28

Опис таблиці `legal_pages`

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор юридичної сторінки
slug	varchar(255)	-	-	Унікальний короткий ідентифікатор сторінки
title	varchar(255)	-	-	Заголовок сторінки
body_text	varchar(4000)	-	-	Текст сторінки (основна мова)
body_text_en	varchar(4000)	-	-	Текст англійською (опц.)
is_published	int	-	-	Прапорець публікації (0/1)
updated_at	datetime	-	-	Дата/час останнього оновлення

Структура таблиці `legal_pages` для публікації правил сервісу. Про коригування складських залишків описано в табл. 2.29.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						43

Таблиця 2.29

Опис таблиці stock_adjustments

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор коригування запасу
product_id	int	-	+	Товар, для якого робиться коригування
color_variant_id	int	-	+	Варіант кольору (опц.)
delta	int	-	-	Зміна запасу (позитивне/негативне)
previous_stock	int	-	-	Запас до коригування
new_stock	int	-	-	Запас після коригування
changed_by_user_id	int	-	+	Хто виконав коригування
reason	varchar(1000)	-	-	Причина коригування
created_at	datetime	-	-	Дата/час коригування

Структура таблиці stock_adjustments. Поля product_id, color_variant_id та changed_by_user_id є зовнішніми ключами до таблиць товарів, варіантів кольорів та користувачів. Про коди безпеки наведено в табл. 2.30.

Таблиця 2.30

Опис таблиці password_email_codes

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Ідентифікатор коду відновлення пароля
user_id	int	-	+	Користувач, для якого видано код
code	varchar(100)	-	-	Код підтвердження (одноразовий)
purpose	varchar(50)	-	-	Призначення коду (password_reset, email_confirm)
expires_at	datetime	-	-	Дата/час закінчення дії коду
used	int	-	-	Прапорець використання (0/1)

продовж. табл. 2.30

created_at	datetime	-	-	Дата/час створення коду
------------	----------	---	---	-------------------------

Опис таблиці password_email_codes, яка використовується для відновлення доступу (зовнішній ключ user_id). Опис активних вкладок браузера (сесій) наведено в табл. 2.31.

Таблиця 2.31

Опис таблиці password_email_codes

Назва поля	Тип даних	PK	FK	Опис
id	int	+	-	Унікальний ідентифікатор запису вкладки
user_id	int	-	+	Користувач, якому належить сесія (опц.)
page_title	varchar(1000)	-	-	Заголовок сторінки (pageTitle)
page_url	varchar(2000)	-	-	URL сторінки (pageUrl)
tab_id	int	-	-	Ідентифікатор вкладки у браузері (tabId)
is_current	int	-	-	Прапорець активної вкладки (1 = поточна, 0 = фонова)
last_active_at	datetime	-	-	Час останньої активності вкладки
created_at	datetime	-	-	Дата/час створення запису
updated_at	datetime	-	-	Дата/час останнього оновлення запису

Поле user_id є зовнішнім ключем, що посилається на users (опціонально). Таблиця зберігає метадані відкритих вкладок браузера: заголовок сторінки, URL, ідентифікатор вкладки, позначку активності та час останньої активності.

2.3. Проектування та реалізація алгоритмів роботи системи

Для опису основних процесів роботи системи FlowersGo розроблено діаграми активності для ключових сценаріїв клієнта та адміністратора, а також

діаграми послідовності та кооперації для відображення взаємодії компонентів застосунку, контролерів, сервісів і бази даних. Оскільки предметом розробки є інтернет-магазин з доставкою квітів, основний акцент зроблено на оформленні замовлення, роботі конструктора букетів, наповненні каталогу та ланцюжку обробки замовлення від оплати до доставки.

Процес оформлення замовлення починається з формування кошика, після чого користувач переходить до оформлення замовлення. На етапі checkout він обирає спосіб доставки, дату, часовий інтервал та вводить необхідні контактні дані. Далі система виконує перевірку наявності товарів на складі, коректності введених даних і можливості здійснення доставки за вказаною адресою.

Процес оформлення може виконуватися як гостем, так і авторизованим користувачем. Для гостя ідентифікатор користувача у замовленні не вказується, а контактні та дані доставки зберігаються безпосередньо в замовленні. Для авторизованого користувача замовлення пов'язується з відповідним обліковим записом.

Після успішного проходження перевірок система створює записи у таблицях orders та order_items. Якщо обрано оплату картою, користувач перенаправляється на сторінку платіжного сервісу LiqPay. Після успішного підтвердження транзакції статус оплати змінюється на paid. У разі вибору оплати при отриманні замовлення фіксується відповідний спосіб оплати, після чого замовлення може бути передане на подальшу обробку після підтвердження адміністратором.

Якщо під час перевірки виявлено недостатню кількість товару на складі або некоректні дані, процес оформлення завершується повідомленням про помилку, а замовлення не створюється. Операції, пов'язані зі створенням замовлення та його позицій, виконуються в межах транзакції бази даних, що забезпечує їх атомарність і запобігає частковому збереженню даних у разі виникнення помилки.

Після успішного створення замовлення система переходить до наступного етапу його обробки. Для працівника складу формується відповідне сповіщення,

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						46

що ініціює процес комплектації замовлення. Клієнту відображається підтвердження успішного оформлення із зазначенням унікального номера замовлення. Після цього замовлення переходить до подальших етапів обробки, комплектації та доставки. Загальну послідовність виконання процесу оформлення замовлення наведено на рис. 2.4.

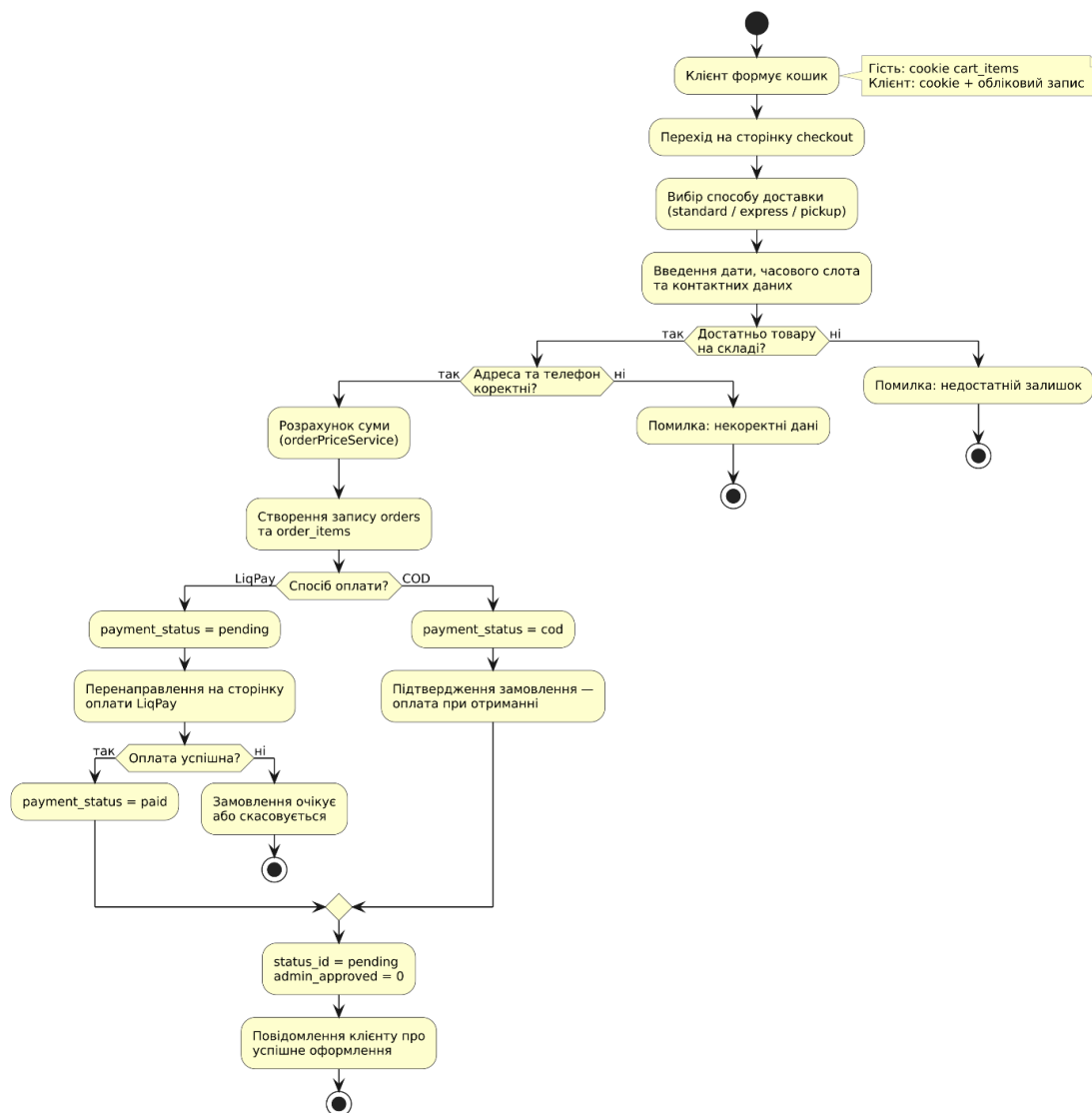


Рис. 2.4 – Діаграма Activity оформлення замовлення

На рис. 2.5 показано діаграму активності збирання букета в конструкторі. Алгоритм починається з відкриття модуля конструктора: користувач обирає квіти з ознакою `is_constructor`, їхні кольорові варіанти з таблиці `product_color_variants`, а також упаковку з відповідної категорії. Окремою гілкою передбачено завантаження готового шаблону з `bouquet_templates`, після чого склад можна змінити вручну.

На кожному кроці система перераховує ціну і перевіряє stock_quantity. Якщо обраної кількості немає, відображається помилка; якщо все коректно - генерується прев'ю букета і позиції додаються в кошик (cookie). Додатково авторизований клієнт може паралельно додати товар у wishlist, але це не є обов'язковим кроком для покупки. Конструктор доступний без авторизації в систему; для оплати далі використовується сценарій оформлення замовлення на рис. 2.4.

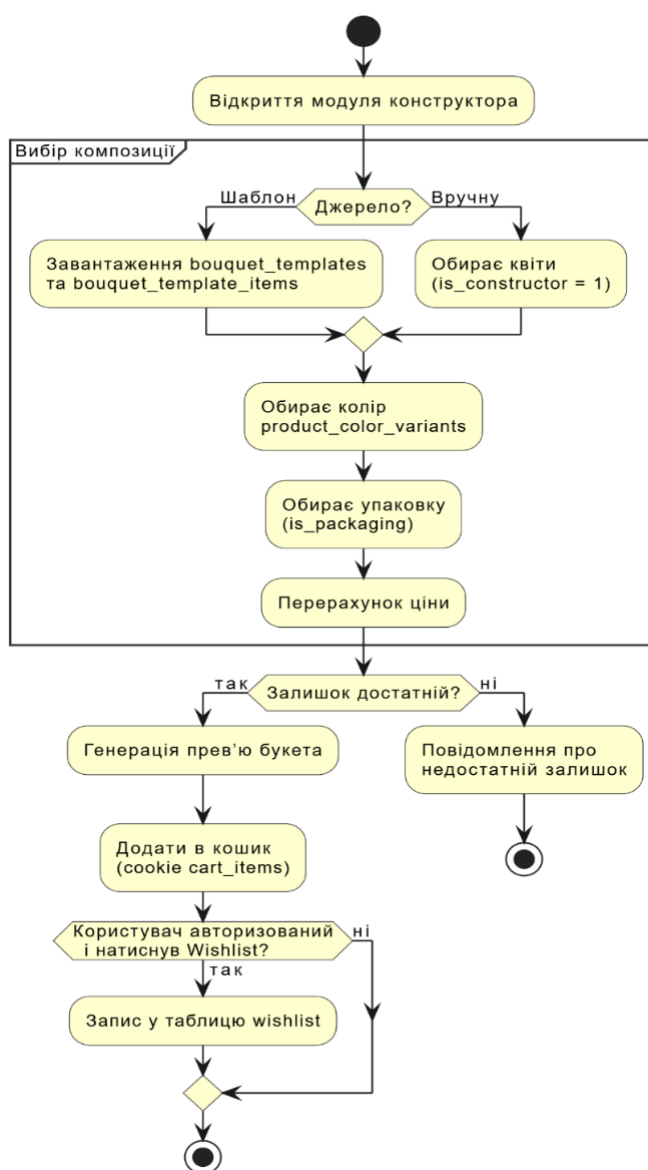


Рис. 2.5 – Діаграма Activity Конструктор букетів

Процес наповнення каталогу з двома ролями показано на рис. 2.6. Звичайний користувач не може безпосередньо створювати записи в products - він переглядає каталог, додає товари в кошик або wishlist. За потреби він звертається

в чат підтримки, але офіційне додавання асортименту виконує лише адміністратор через захищену панель.

Адміністратор створює або редагує категорії, додає товар з ціною, slug, залишком і зображенням. Система перевіряє унікальність slug і SKU; при дублікаті повертається помилка. Для квітів із кількома відтінками створюються записи в product_color_variants. Окремо можна керувати шаблонами букетів та глобальними правилами delivery_settings. Після збереження товар з'являється в каталозі та стає доступним у конструкторі.

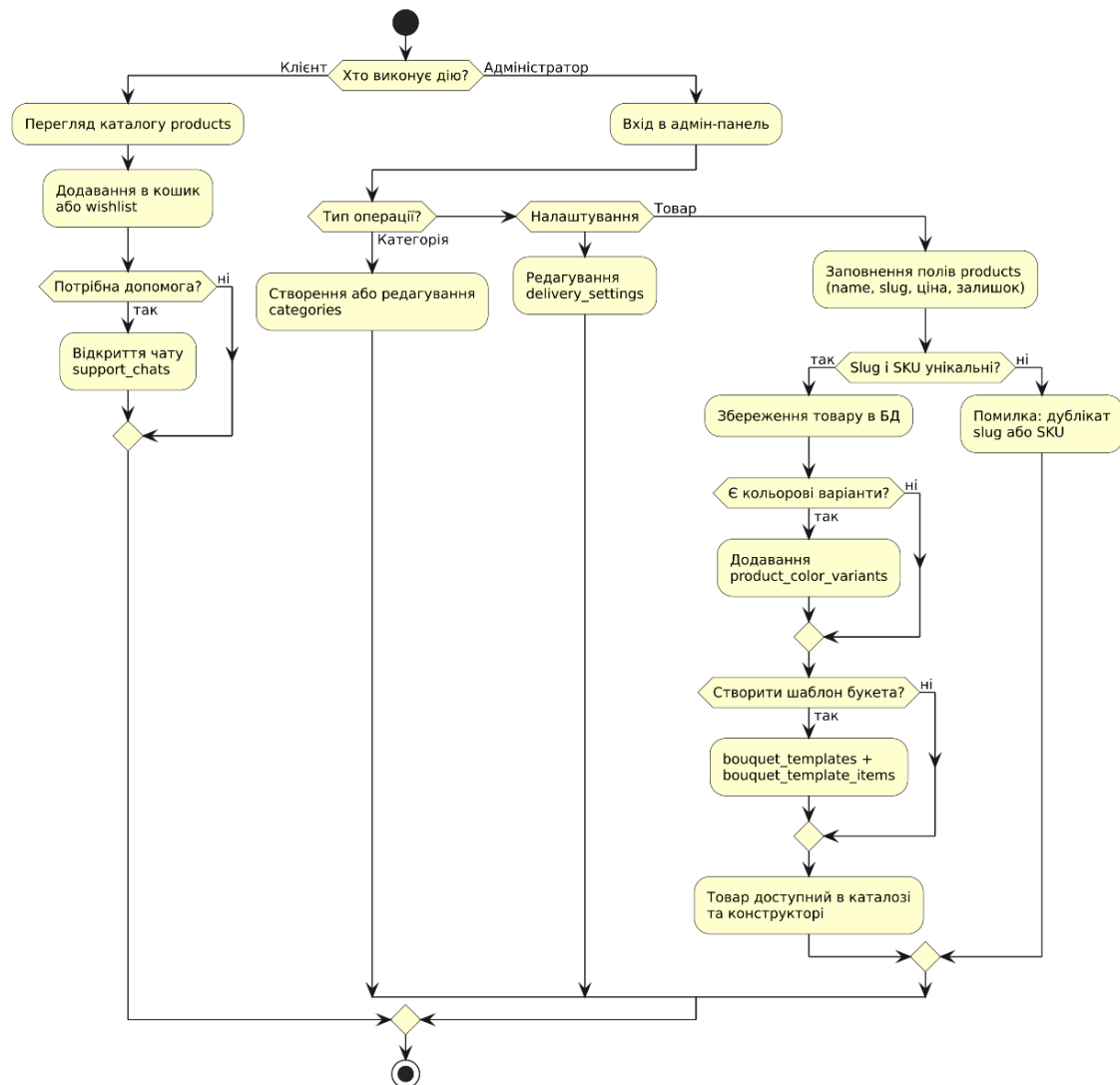


Рис. 2.6 – Діаграма Activity додавання товару в каталог

На рис. 2.7 наведено діаграму послідовності створення замовлення. Клієнт надсилає форму checkout з браузера; запит обробляє orderController, який звертається до cartService (позиції кошика), orderPriceService (сума з доставкою)

та orderStockService (перевірка залишків). Модель Order транзакційним методом записує дані в orders і order_items.

Далі відбувається розгалуження за способом оплати. При LiqPay клієнт переходить на сторінку оплати, а після callback сервер оновлює payment_status. При COD статус одразу встановлюється як оплата при отриманні. Якщо користувач авторизований, Notification model створює запис у user_notifications. Діаграма показує, що створення замовлення - центральна взаємодія між інтерфейсом, контролером, сервісами та базою даних MySQL.

Діаграму Sequence наведено детально описано на рис.2.7 .

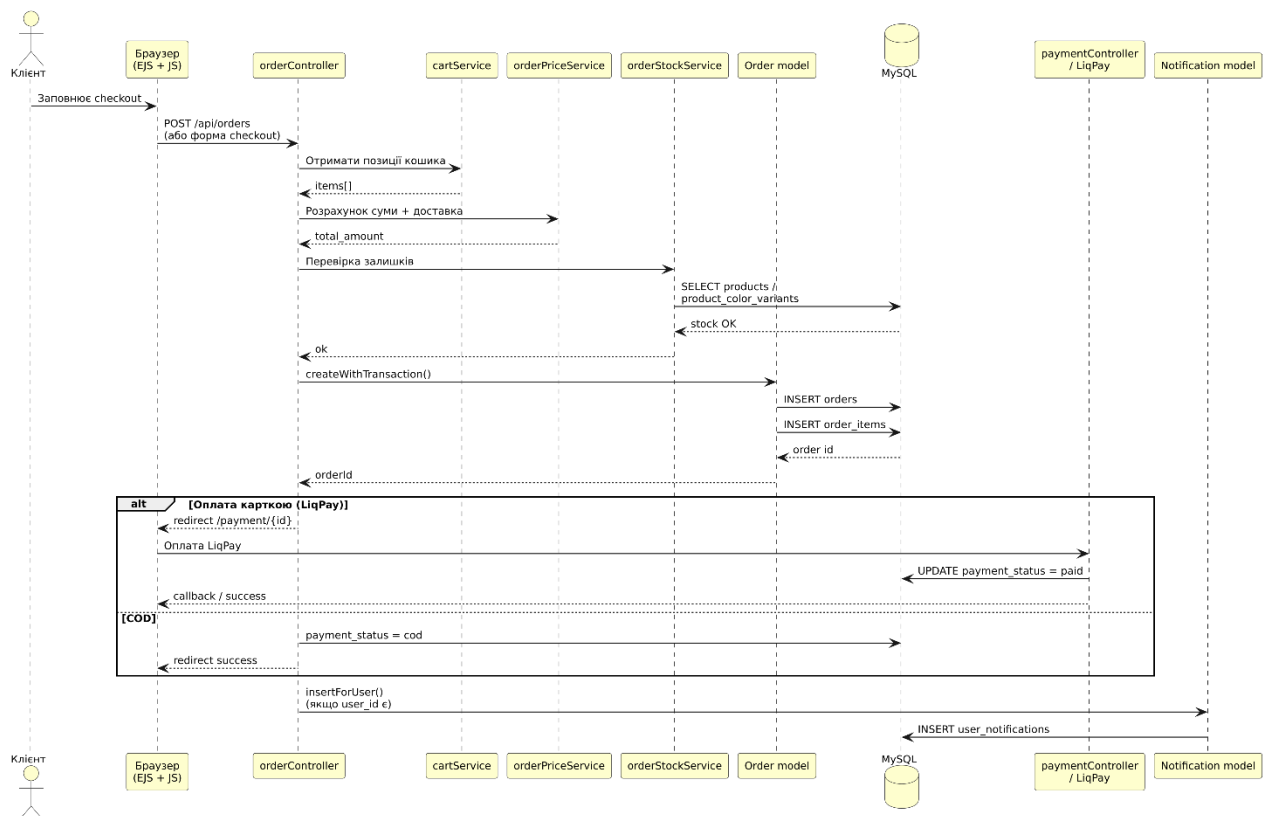


Рис. 2.7 – Діаграма Sequence замовлення товару – створення замовлення

Діаграма послідовності зображена на рис. 2.8. Вона описує оновлення статусу замовлення після оплати. Адміністратор підтверджує замовлення (admin_approved = 1). Працівник складу через warehouseController змінює статус згідно з довідником statuses і правилами status_transitions; кожна зміна фіксується в order_status_log через orderStatusService.

Кур'єр отримує призначене замовлення (courier_id), оновлює статуси доставки (shipped, delivered) і за потреби підтверджує COD. Після кожної важливої зміни Notification model записує сповіщення в user_notifications, які клієнт бачить у кабінеті. На відміну від платформ з real-time хабом, FlowersGo зберігає сповіщення в БД і завантажує їх при відкритті сторінки - це також відображено на діаграмі.

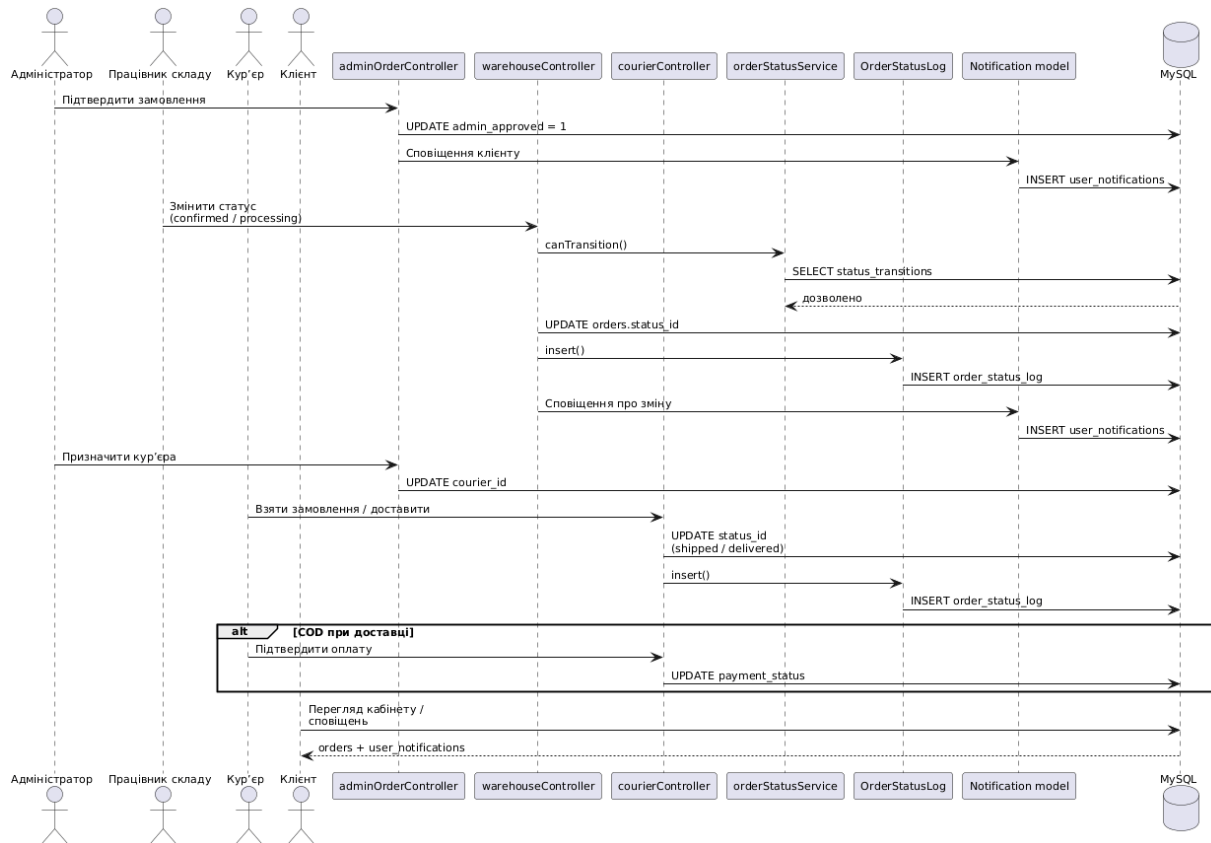


Рис. 2.8 – Діаграма Sequence зміна статусу замовлення

Діаграма кооперації на рис. 2.9 показує структурні зв'язки між учасниками процесу обробки замовлення. На відміну від діаграми послідовності, тут акцент зроблено не на часі, а на взаємодії об'єктів: клієнт, адміністратор, працівник складу, кур'єр, відповідні контролери, моделі та база даних MySQL.

Клієнт через orderController ініціює створення замовлення; Order model зберігає дані в orders і order_items. Адміністратор підтверджує замовлення і може модерувати reviews (поле is_visible). Склад і кур'єр змінюють статуси з записом у order_status_log; Notification model інформує клієнта через user_notifications.

Нумерація повідомлень на діаграмі показує повний ланцюжок від оформлення покупки до завершення доставки без окремого сервісу модерації контенту.

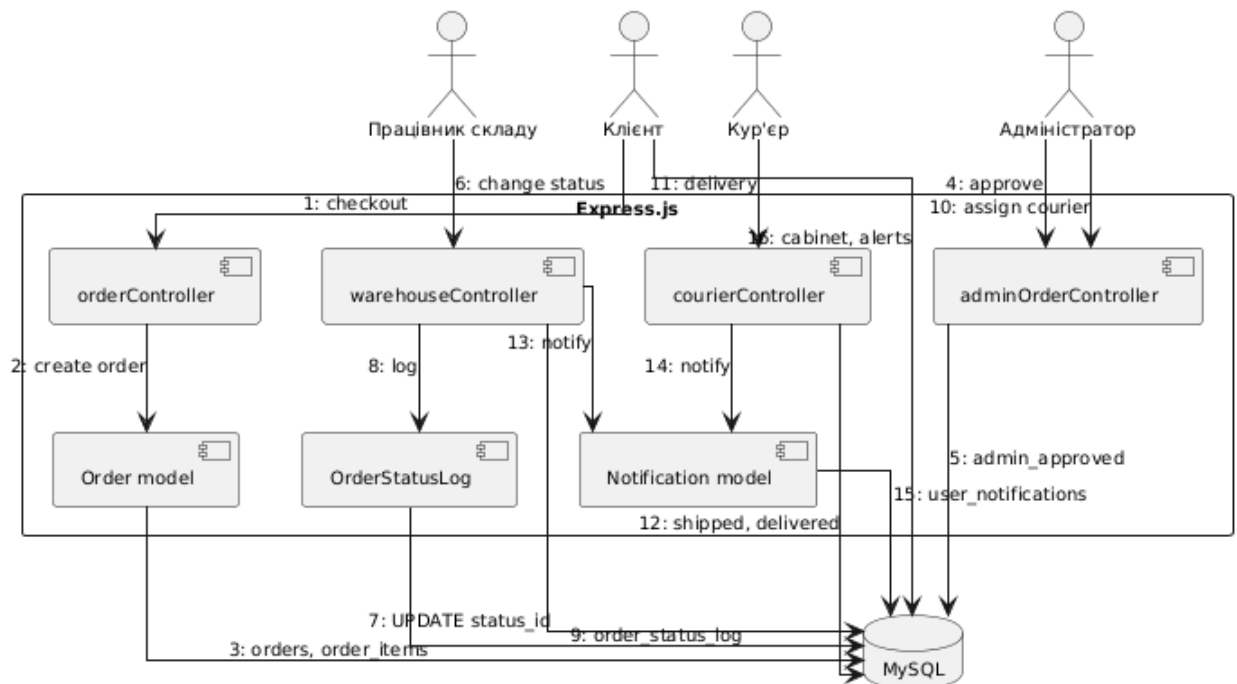


Рис. 2.9 – Діаграма Communication кооперація обробки замовлення

Окрім діаграм активності та послідовності, у FlowersGo реалізовано низку прикладних алгоритмів, які безпосередньо впливають на коректність продажів. Нижче коротко описано логіку, яка вже відображена у сервісному шарі Node.js і моделі Order.

Алгоритм розрахунку вартості замовлення виконується у orderPriceService. Після оформлення кошика сервер не довіряє ціні з браузера: для кожної позиції заново зчитується актуальна ціна товару або кольорового варіанту, додається вартість доставки згідно з delivery_settings (стандарт, точний час, експрес або самовивіз) і лише після цього формується підсумкова сума для запису в orders.

Перед створенням замовлення orderStockService перевіряє наявність кожної позиції на складі. Для звичайних товарів аналізується stock_quantity, для квіток конструктора - залишок у product_color_variants. Якщо хоча б однієї одиниці не вистачає, транзакція не виконується і клієнт отримує повідомлення про помилку без зайвих записів у БД.

Створення замовлення реалізовано у межах бази даних у методі Order.createWithTransaction. У межах одного з'єднання з MySQL спочатку повторно перевіряються залишки, потім додається рядок у orders зі статусом pending, позиції - у order_items, а для онлайн-оплати встановлюється payment_deadline_at (10 хвилин на оплату через LiqPay). При виборі оплати при отриманні payment_status одразу отримує значення cod, після чого замовлення може потрапити на модерацію.

Після схвалення адміністратором (admin_approved = 1) спрацьовує списання складу: кількість товарів і кольорових варіантів зменшується відповідно до складу замовлення, а зміна фіксується в order_status_log. Далі замовлення переходить у чергу складу, а після статусу ready_for_pickup може бути призначене кур'єру вручну або через автоматичне призначення серед кур'єрів із courier_on_shift = 1.

Окремий алгоритм реалізовано в constructorService.calcBouquet: при кожній зміні складу букета перераховується кількість елементів композиції та їх доступність у відповідних варіантах, перевіряється доступність кольорів і формується підсумкова ціна з урахуванням упаковки. Результат однаково використовується на сторінці конструктора, у кошику та на checkout, тому клієнт не може оформити букет за ціною, яка не відповідає поточним залишкам.

Для сповіщень застосовано подієву модель: після зміни статусу замовлення, відповіді в чаті підтримки або рішення по відгуку у user_notifications створюється запис, який користувач бачить у кабінеті та у випадючій панелі в шапці сайту. Такий підхід не вимагає окремого real-time сервера, але забезпечує прозорий супровід замовлення на всіх етапах.

Висновки до другого розділу

Для вебсистеми FlowersGo визначено функціональні та нефункціональні вимоги з урахуванням особливостей інтернет-магазину з доставкою квітів. Виділено основних учасників системи: гостя, клієнта, працівника складу, кур'єра

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						53

та адміністратора, а також визначено основні сценарії їхньої взаємодії із системою.

Для опису структури системи побудовано діаграму варіантів використання, діаграму класів, узагальнену та повну ER-схеми бази даних, а також визначено поля таблиць. Це дало змогу розподілити основні сутності системи, зокрема користувачів, каталог, замовлення, відгуки та підтримку, і визначити зв'язки між ними.

Розроблено реляційну базу даних MySQL з 22 таблицями, яка зберігає облікові записи, каталог, конструктор букетів, замовлення, статуси, сповіщення та допоміжні довідники. Використання зовнішніх ключів і правил цілісності зменшує ризик “висячих” записів і спрощує підтримку даних при розвитку функціоналу.

Змодельовано динаміку роботи системи за допомогою діаграм активності, послідовності та кооперації. Деталізовано алгоритми оформлення замовлення, роботи конструктора букета, наповнення каталогу адміністратором, створення замовлення на сервері, зміни статусів та взаємодії ролей під час обробки замовлення. Це дозволило перевірити логічні ланцюжки ще на етапі проєктування.

Розпочато програмну реалізацію на стеку Node.js + Express + EJS + MySQL: розроблено контролери та моделі для управління замовленнями, каталогом, оплатами, складом, кур'єрами та адмін-панеллю; реалізовано валідацію вхідних даних, інтеграцію з LiqPay, модуль конструктора букетів і механізм сповіщень користувачів. Отримана модель і код узгоджені між собою і створюють основу для подальшого опису інтерфейсу та тестування системи в наступних розділах роботи.

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						54

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНТЕРНЕТ-МАГАЗИНОМ

3.1. Проєктування варіантів використання та архітектури системи

Вебзастосунок FlowersGo, розроблений у межах випускної кваліфікаційної роботи бакалавра, побудований як модульний моноліт на основі Node.js та Express. Серверна частина одночасно обробляє HTTP-запити, надає REST API та формує HTML-сторінки за допомогою засобу шаблонізації EJS. Такий підхід спрощує розгортання порівняно з розподіленою архітектурою, оскільки один процес застосунку обслуговує весь функціонал системи: каталог товарів, кошик, замовлення, адмін-панель та кабінет користувача. Логіка системи структурована за шарами (routes – controllers – services – models), що забезпечує чіткий поділ відповідальності між компонентами та спрощує і подальше масштабування без переходу до архітектури мікросервісів.

Користувацька взаємодія із системою реалізується за клієнт-серверною моделлю. Клієнтська частина відповідає за відображення інтерфейсу та взаємодію з користувачем, тоді як серверна частина забезпечує виконання бізнес-логіки, обробку запитів і доступ до даних. Такий підхід дозволяє чітко розділити відповідальність між компонентами системи, підвищує її масштабованість і спрощує тестування та подальший супровід.

Для публічного доступу система розгорнута у хмарному середовищі: вебзастосунок працює на платформі Render.com, а реляційна база даних розміщена у керованому сервісі Aiven MySQL 8.0. З'єднання між сервером і базою даних здійснюється через змінні середовища з використанням SSL-шифрування. Додатково інтегровані зовнішні сервіси: платіжна система LiqPay, авторизація Google OAuth та SMTP-сервер для надсилання електронних повідомлень. Це забезпечує відокремлення обчислювальної частини від сховища даних і дозволяє оновлювати застосунок без зупинки бази даних.

Для наочного відображення складових розгортання наведено діаграму розгортання на рис. 3.1.

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						55

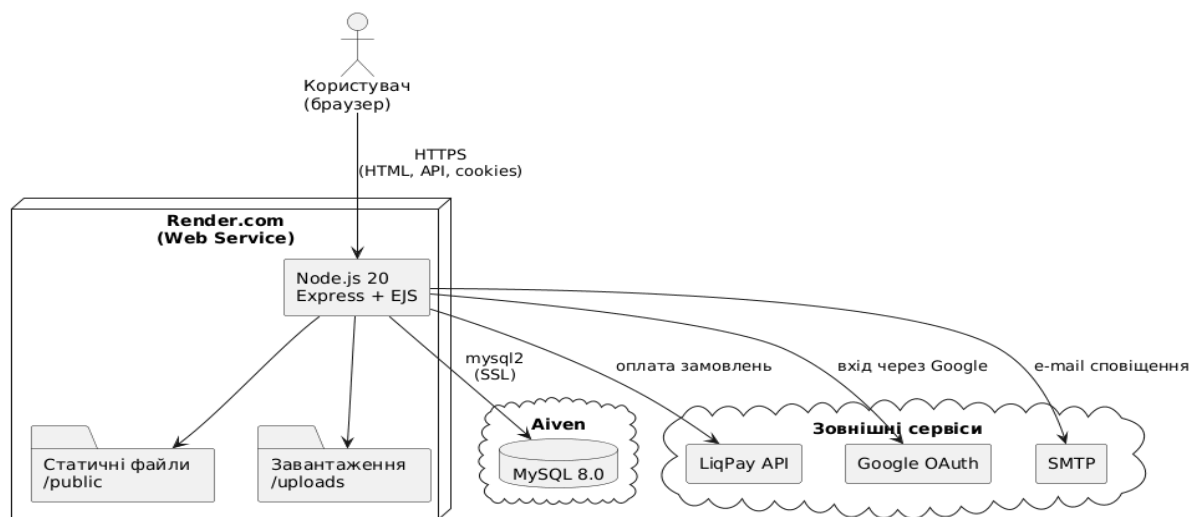


Рис. 3.1 – Діаграма розгортання системи

Вебплатформа FlowersGo має три логічні компоненти, які в продакшені можуть бути розміщені на різних вузлах інфраструктури. Для стандартизації середовища виконання, незалежно від типу розгортання (хмарна платформа чи локальний сервер), використовується технологія контейнеризації Docker [9]. Використання Docker Compose дозволяє автоматизувати керування компонентами застосунку, забезпечуючи ідентичність оточення під час розробки, тестування та експлуатації. Основні підходи до розгортання системи наведено у табл. 3.1.



Рис. 3.2 Логічна структура та взаємодія компонентів вебплатформи

Послідовність взаємодії компонентів вебплатформи продемонстровано на рис. 3.2.

Рекомендовані системні вимоги для запуску на сервері наведено нижче.

Процесор	Оперативна пам'ять	Пам'ять на диску
• 4 ядра • тактова частота від 3 ГГц	• мінімум 8 ГБ	• мінімум 16 ГБ вільного місця

Рис. 3.3 – Системні вимоги до сервера

Таблиця 3.1

Хмарне розгортання (Render.com)	Розгортання через Docker Compose	Інфраструктурні компоненти
Створюється Web Service на Render.com із підключенням GitHub-репозиторію (синхронізація коду).	На сервері або локальному середовищі встановлюються Docker і Docker Compose, після чого клонується репозиторій проєкту.	Створюється керований екземпляр MySQL 8.0 на платформі Aiven із отриманням параметрів підключення (host, port, user, password, database).
У налаштуваннях сервісу задається Root Directory: backend, команда запуску npm start та порт 5000.	Створюється файл .env у корені проєкту та задаються змінні DB_*, JWT_SECRET, APP_BASE_URL, ключі LiqPay, Google OAuth та SMTP.	Увімкнено SSL-з'єднання з базою даних; у застосунку встановлюється DB_SSL=1 (за потреби — шлях до СА-сертифіката).
Додаються змінні середовища в панелі Render: DB_HOST, DB_PORT, DB_USER, DB_PASS, DB_NAME, JWT_SECRET, APP_BASE_URL, LIQPAY_*, GOOGLE_CLIENT_ID, SMTP.	Виконується збірка та запуск контейнерів: docker compose up --build -d (контейнери API та DB).	Імпортується структура та дані бази даних (SQL-скрипт) у MySQL.
Після push у GitHub виконується автоматичний deploy. Перевірка доступу здійснюється через публічну URL-адресу (наприклад, https://flowersgo-app.onrender.com).	Перевірка стану виконується через docker compose ps або відкриття застосунку на порту 5000.	Налаштовується доступ до БД: дозволяються підключення з IP-адреси сервера застосунку (Render або VPS) у консолі Aiven.

Рекомендації з безпеки та експлуатації наведено на рис. 3.4.

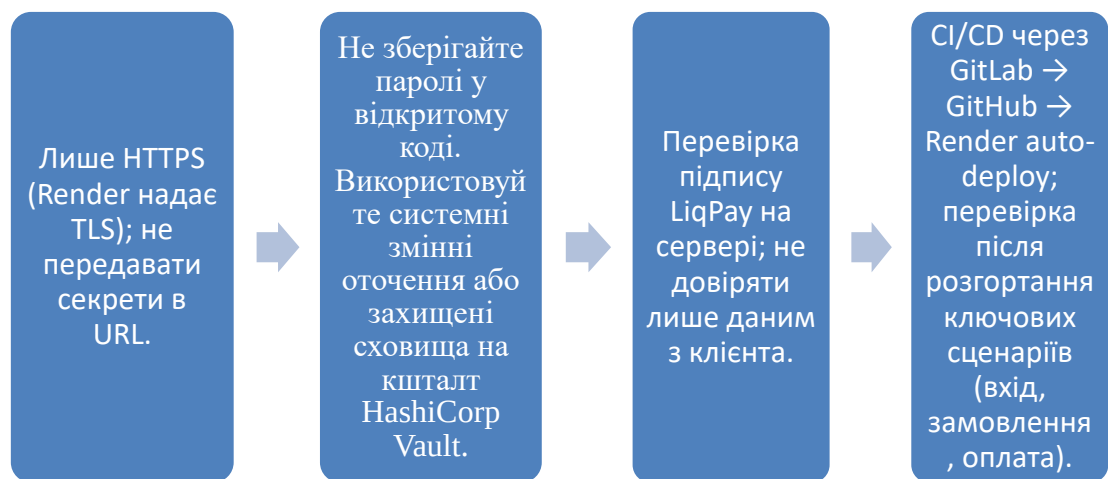


Рис. 3.4 – Рекомендації з безпеки та експлуатації

Клієнтська система адаптована для роботи у сучасних браузерах: Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Стабільність забезпечується використанням стандартів HTML5/CSS3 та мінімального набору JavaScript-модулів у директорії /public/js без потреби у складних інструментах збирання коду.

3.2. Структура інтерфейсу вебплатформи

Вебплатформа FlowersGo реалізована як модульний вебзастосунок з використанням серверного рендерингу (EJS) та клієнтської взаємодії через REST API. Проєкт розміщено на платформі Render.com із використанням керованої бази даних Aiven MySQL (<https://flowersgo-app.onrender.com>).

Головна сторінка FlowersGo призначена для швидкого переходу до основних розділів магазину та перегляду доступних товарів. Її інтерфейс наведено на рис. 3.5. Верхня частина сторінки містить основні елементи навігації, за допомогою яких користувач може перейти до каталогу, конструктора букетів, кошика та особистого кабінету. Центральну частину займає інформаційний блок із рекламними пропозиціями та добірками товарів.

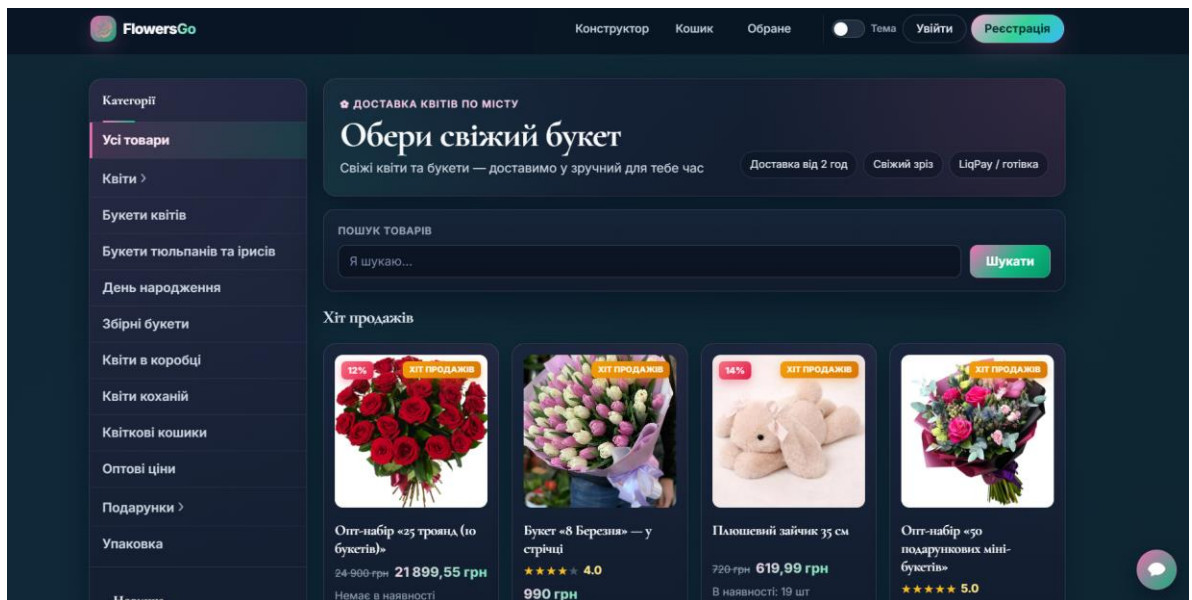


Рис. 3.5 – Головна сторінка сайту

Для зміни вигляду інтерфейсу передбачено перемикач колірної теми. При його використанні елементи сторінки змінюють оформлення відповідно до обраного режиму. Приклад переходу з темної теми на світлу наведено на рис. 3.6.

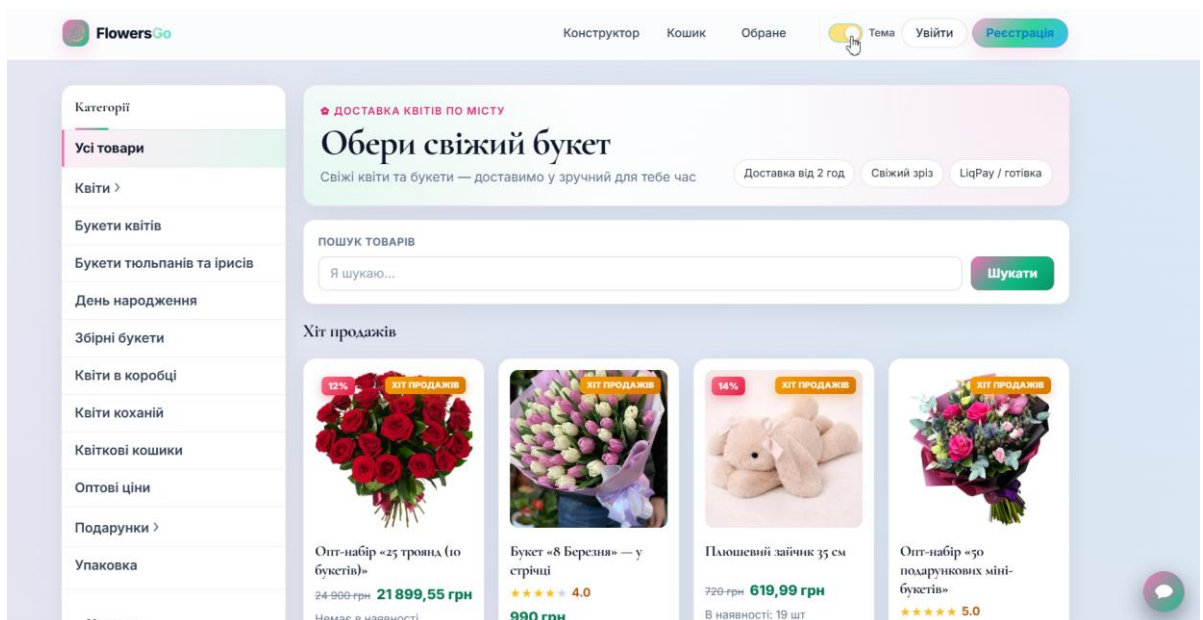


Рис. 3.6 – Зміна теми із темної на світлу за допомогою перемикача

На головній сторінці також представлено добірку популярних товарів. Каталог вибору товарів і блок «Хіт продажів» наведено на рис. 3.7. Картки містять основну інформацію про товар, зображення, ціну та елемент для переходу до його оформлення.

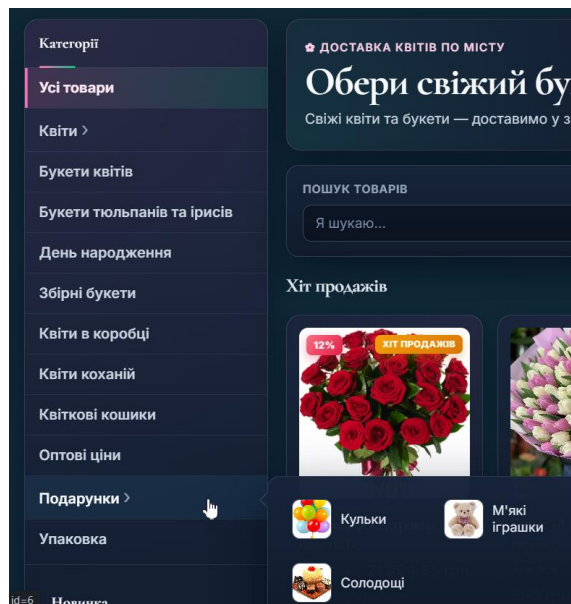


Рис. 3.7 – Каталог вибору товару та стрічка “Хіт продажів”

Окремий блок головної сторінки призначений для переходу до конструктора букетів. Відповідний банер із посиланням на модуль конструктора наведено на рис. 3.8. Такий спосіб розміщення дозволяє перейти до створення власної композиції без необхідності додаткового пошуку потрібного розділу в меню.

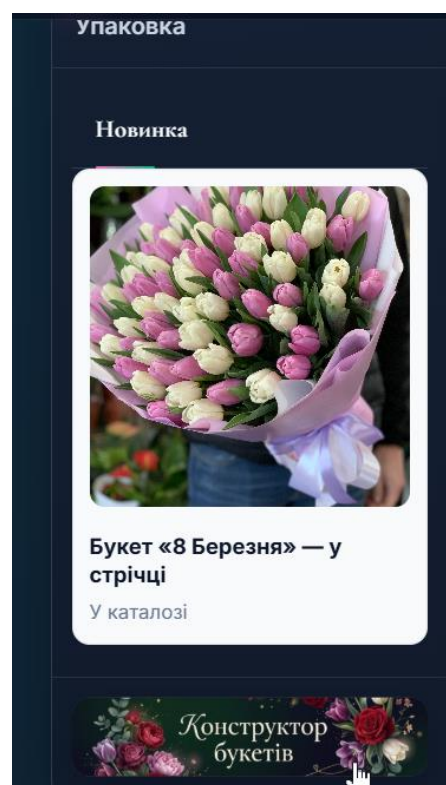


Рис. 3.8 – Банер для переходу в конструктор букетів

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				60

Під основними товарними пропозиціями розміщено додаткові категорії та рекомендації. Під час прокручування сторінки користувач може переглядати різні добірки товарів, що відповідають окремим категоріям. Такий спосіб представлення дозволяє ознайомитися з асортиментом без переходу до окремих сторінок каталогу та швидше знайти потрібний товар. Розміщення кількох тематичних добірок послідовно також спрощує перегляд великої кількості товарів. Приклад такого представлення наведено на рис. 3.9.

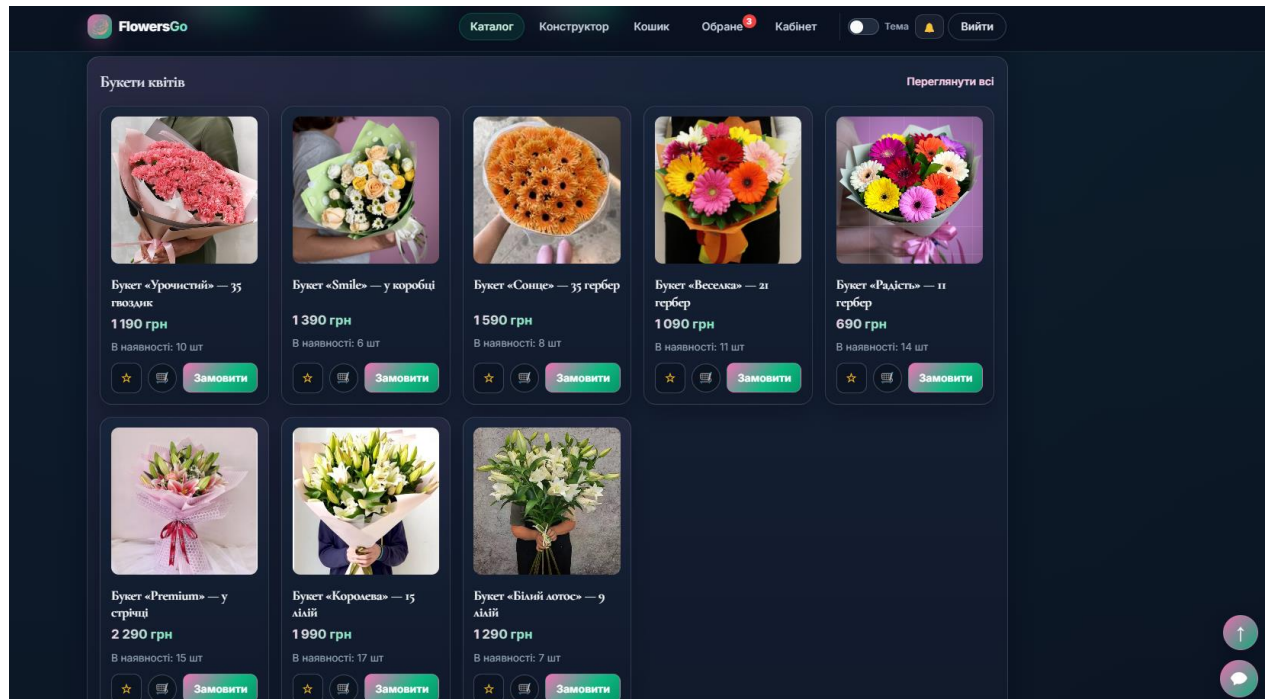


Рис. 3.9 – Відображення різних рекомендованих категорій товарів користувачу при гортанні вниз головної сторінки

Використання модульних шаблонів EJS забезпечує однакову структуру основних елементів інтерфейсу та спрощує їх повторне використання на різних сторінках вебплатформи. Подальше вдосконалення стилізації та адаптивності інтерфейсу може передбачати використання utility-first підходу, зокрема фреймворку Tailwind CSS [28].

Реєстрація користувача виконується через окрему форму, у якій необхідно вказати основні облікові дані. Інтерфейс форми реєстрації наведено на рис. 3.10.

Рис. 3.10 – Форма реєстрації користувача

Для спрощення процесу реєстрації також передбачено можливість використання облікового запису Google. Відповідна форма наведена на рис. 3.11.

Рис. 3.11 – Форма реєстрації користувача через Google

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						62

Після успішного завершення реєстрації система відображає користувачу відповідне повідомлення, що підтверджує створення облікового запису. Приклад такого повідомлення наведено на рис. 3.12.

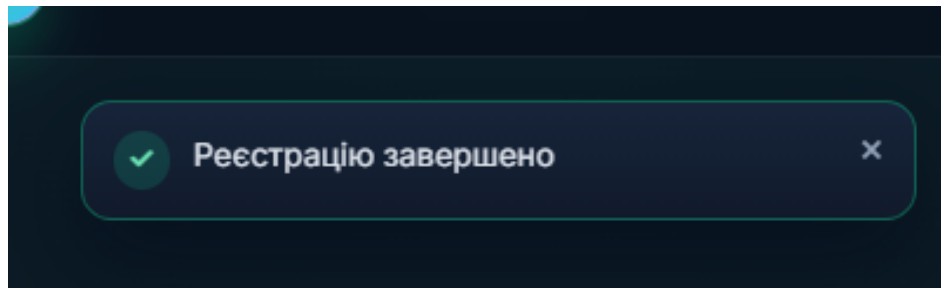


Рис. 3.12 – Сповіщення про реєстрацію користувачу

Для входу до облікового запису користувач вводить адресу електронної пошти та пароль. Форма авторизації наведена на рис. 3.13.

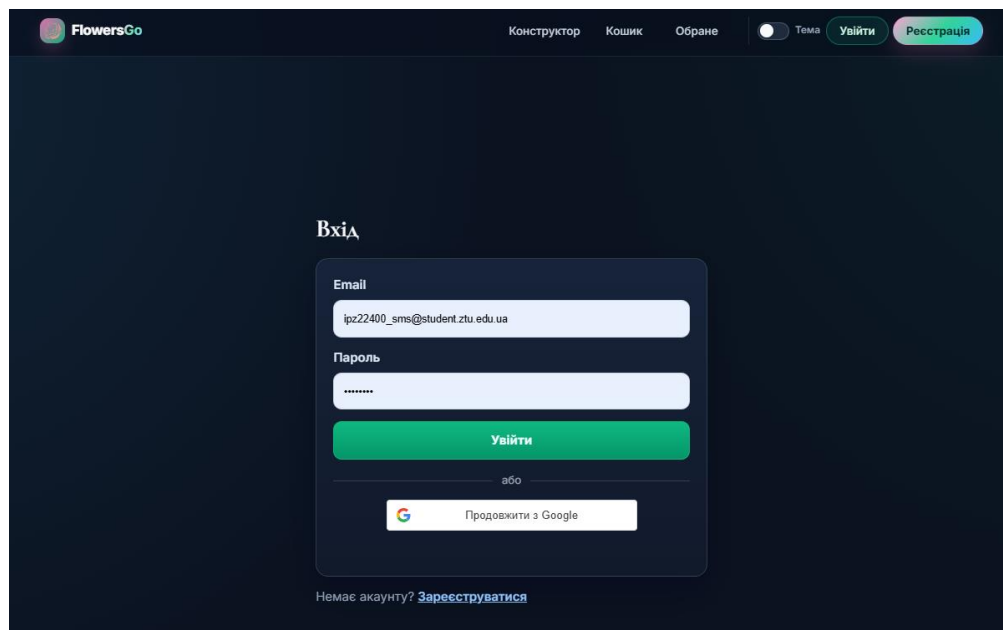


Рис. 3.13 – Форма авторизації користувача

Після успішної авторизації сервер створює JWT-токен, який зберігається в cookie та використовується для подальшої ідентифікації користувача під час роботи із системою. Паролі перед збереженням у базі даних хешуються за допомогою бібліотеки bcrypt.

На сторінці «Акаунт заблоковано» користувач отримує повідомлення про неможливість входу до облікового запису. Приклад спроби входу до заблокованого адміністратором акаунта наведено на рис. 3.14.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						63

Рис. 3.14 – Спроба входу в заблокований Адміном акаунт

Після цього користувач може перейти до сторінки оскарження блокування. На ній відображається інформація про причину блокування та передбачена можливість надіслати повідомлення адміністратору. Відповідний інтерфейс наведено на рис. 3.15.

Рис. 3.15 – Сторінка оскарження блокування акаунта для користувача

Користувач може описати проблему та надіслати звернення до служби підтримки. Після успішного надсилання система відображає відповідне повідомлення про виконання операції, що наведено на рис. 3.16.

Рис. 3.16 – Сповіщення про надсилання повідомлення адміну про причини блокування.

Надіслане звернення надходить до адміністратора через чат підтримки. Адміністратор може переглянути повідомлення користувача та опрацювати його звернення. Відповідний інтерфейс наведено на рис. 3.17.

№	Клієнт	Останнє	Статус	
5	Матвій Сковлюк ipz22400_sms@student.ztu.edu.ua	[Звернення щодо блокування] Скажіть точну причину блокування	В черзі 1	Взяти

Рис. 3.17 – Сповіщення та повідомлення адміну про оскарженн від користувача про блокування

Кошик і список обраного доступні як авторизованим користувачам, так і гостям. Для неавторизованого користувача вибрані товари зберігаються в cookie, а після входу до облікового запису дані синхронізуються з профілем. Приклад порожнього списку обраного для неавторизованого користувача наведено на рис. 3.18.

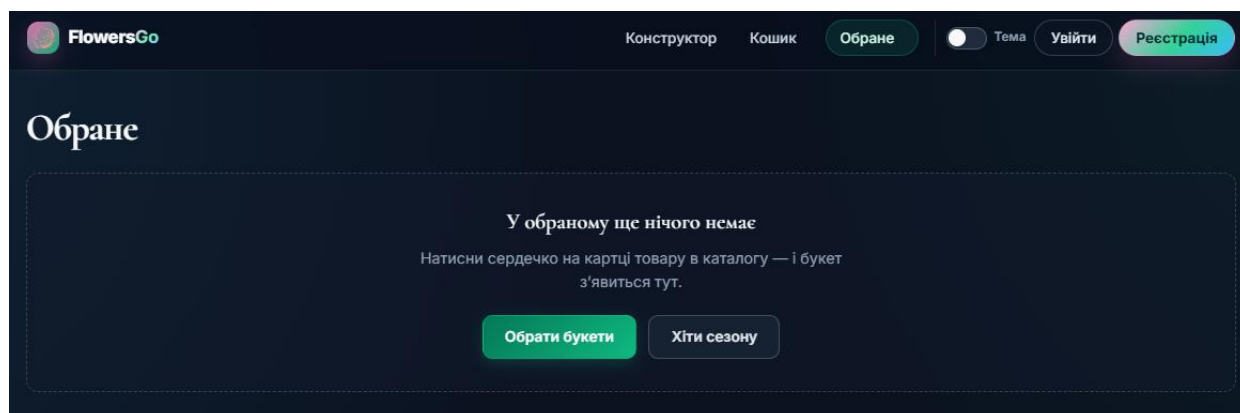


Рис. 3.18 – Порожній кошик обраного, без авторизації

Після додавання товарів список обраного містить відповідні позиції, які користувач може переглядати та використовувати для подальшого переходу до оформлення покупки. Приклад списку обраного з товарами наведено на рис. 3.19.

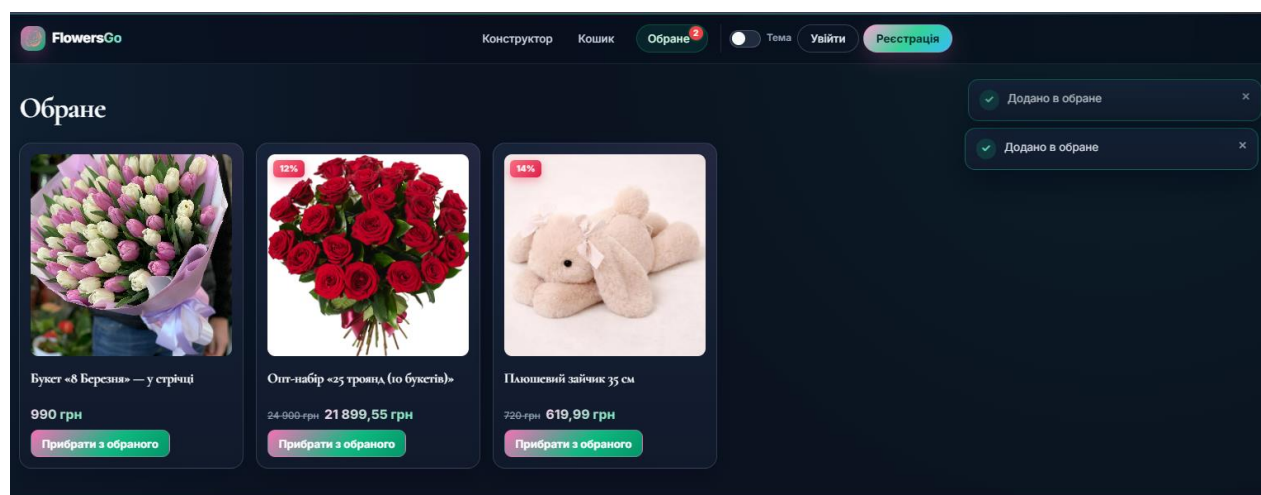


Рис. 3.19 – Кошик обраного, з товарами

Кошик призначений для перегляду вибраних для покупки товарів та зміни їх кількості. Якщо до кошика не додано жодної позиції, система відображає відповідне повідомлення, як показано на рис. 3.20.

		Сковлюк М.С.			ПТЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						66

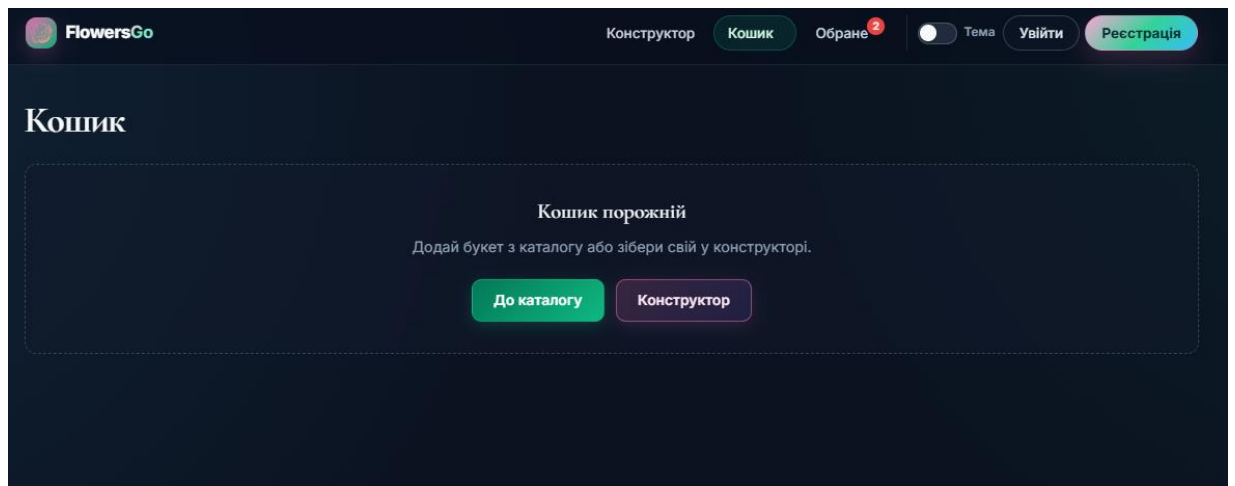


Рис. 3.20 – Порожній кошик, без авторизації

Після додавання товарів у кошик користувач може змінити їх кількість, видалити окремі позиції та перейти до оформлення замовлення. Перед переходом до оформлення замовлення користувач може перевірити склад кошика та скоригувати кількість вибраних товарів. Приклад заповненого кошика наведено на рис. 3.21.

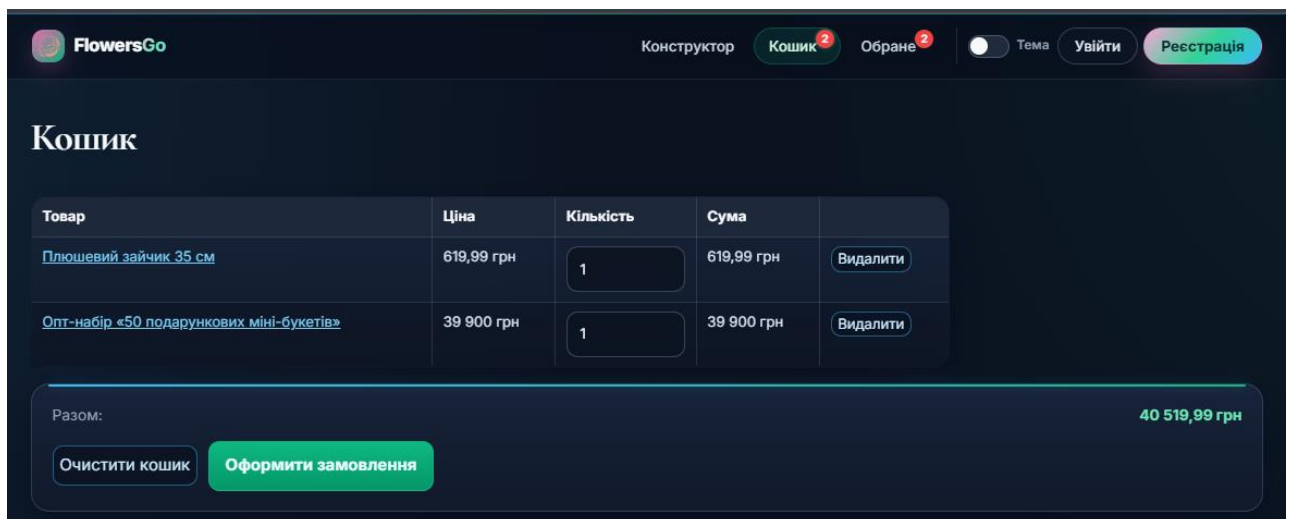


Рис. 3.21 – Кошик з доданими товарами

Сторінка овару містить основну інформацію про вибрану позицію: зображення, ціну, доступну кількість та опис. Приклад сторінки перегляду товару наведено на рис. 3.22.

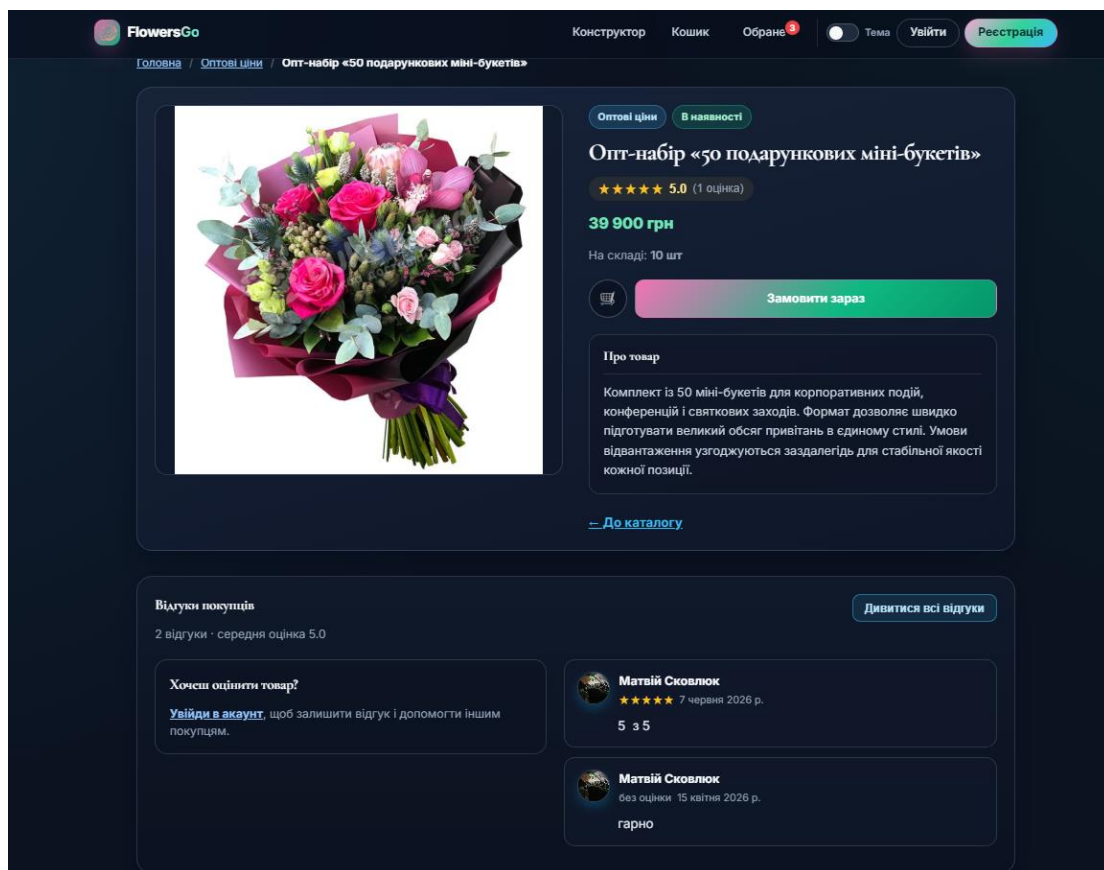


Рис. 3.22 – Сторінка перегляду товару з описом

Для неавторизованого користувача на сторінці товару доступний блок із загальним рейтингом та відгуками інших покупців. Відповідний елемент інтерфейсу наведено на рис. 3.23.

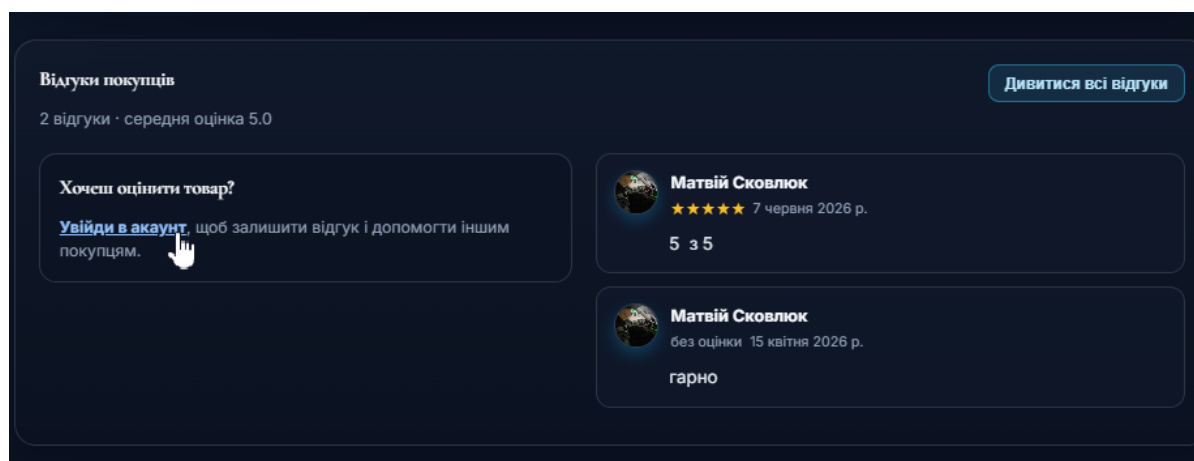


Рис. 3.23 – Блок рейтингу та відгуків для неавторизованого користувача

Після перегляду відгуків користувач може ознайомитися з оцінками та текстовими коментарями інших клієнтів. Загальний рейтинг товару та перелік відгуків наведено на рис. 3.24.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						68

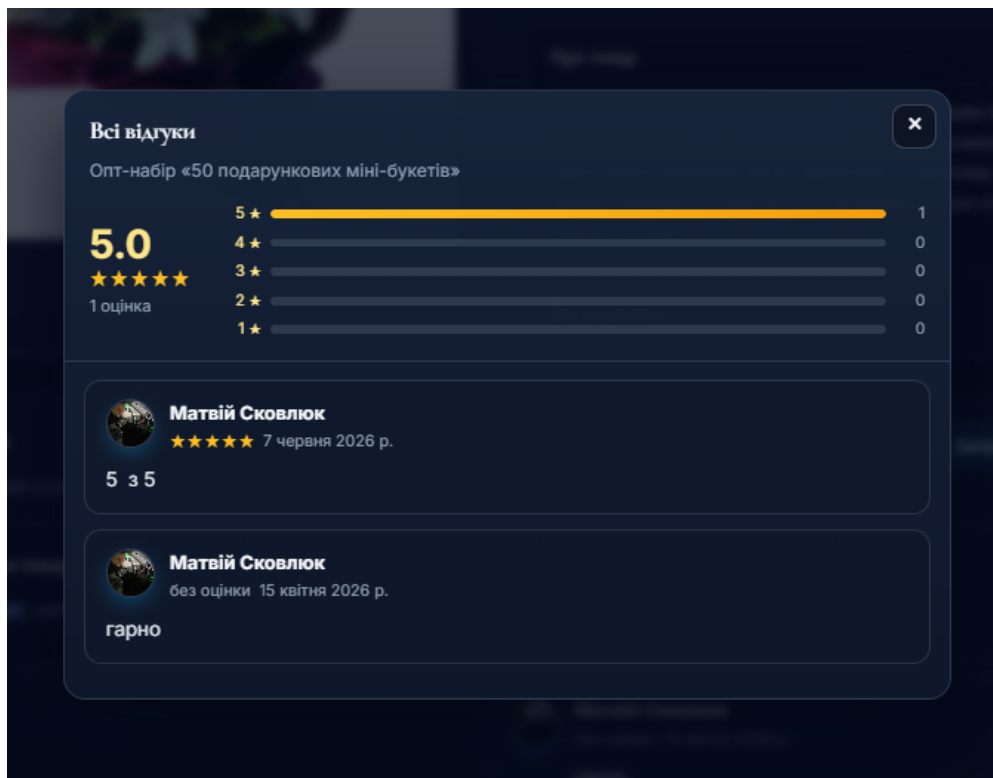


Рис. 3.24 – Блок усіх відгуків та загальний рейтинг від користувачів

Авторизований клієнт додатково отримує можливість залишити власну оцінку товару від 1 до 5 балів та додати текстовий відгук. Форма для створення відгуку наведена на рис. 3.25.

Рис. 3.25 – Блок відгуків для авторизованого користувача

Додані користувачами відгуки перед публікацією проходять перевірку адміністратором. В адміністративній частині системи відображаються відгуки, які очікують схвалення, що показано на рис. 3.26.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						69

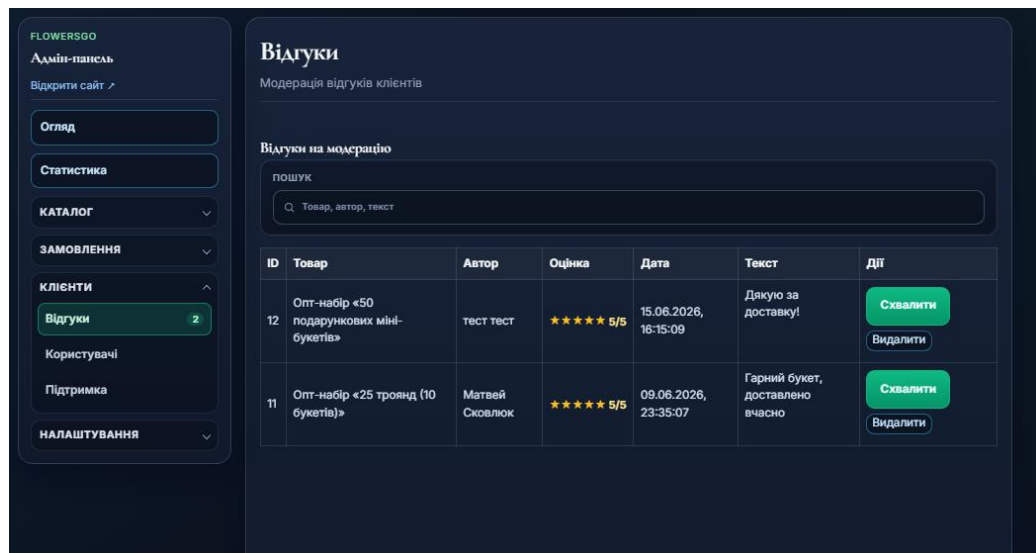


Рис. 3.26 – Перегляд зі сторони адміністратора відгуків які очікують схвалення

Оформлення замовлення доступне як гостю, так і авторизованому користувачу. На сторінці оформлення користувач вказує контактні дані та адресу доставки, обирає спосіб і час доставки, спосіб оплати та за потреби додає коментар до замовлення. Форма оформлення замовлення для неавторизованого користувача наведена на рис. 3.27.

Рис. 3.27 – Форма замовлення товару не авторизованим користувачем

На наступному етапі користувач обирає дату доставки, доступний спосіб оплати та локацію доставки. Відповідні елементи форми наведено на рис. 3.28.

Рис. 3.28 – Форма вибору дати доставки, способу оплати, та вибір локації доставки

При виборі оплати картою користувач перенаправляється на платіжну сторінку LiqPay. Перехід до зовнішнього платіжного сервісу після оформлення замовлення наведено на рис. 3.29.

Рис. 3.29 – Форма перенаправлення на LiqPay для оплати

Після успішного завершення платіжної операції система отримує відповідь від платіжного сервісу та фіксує результат оплати. Приклад успішного завершення оплати наведено на рис. 3.30.

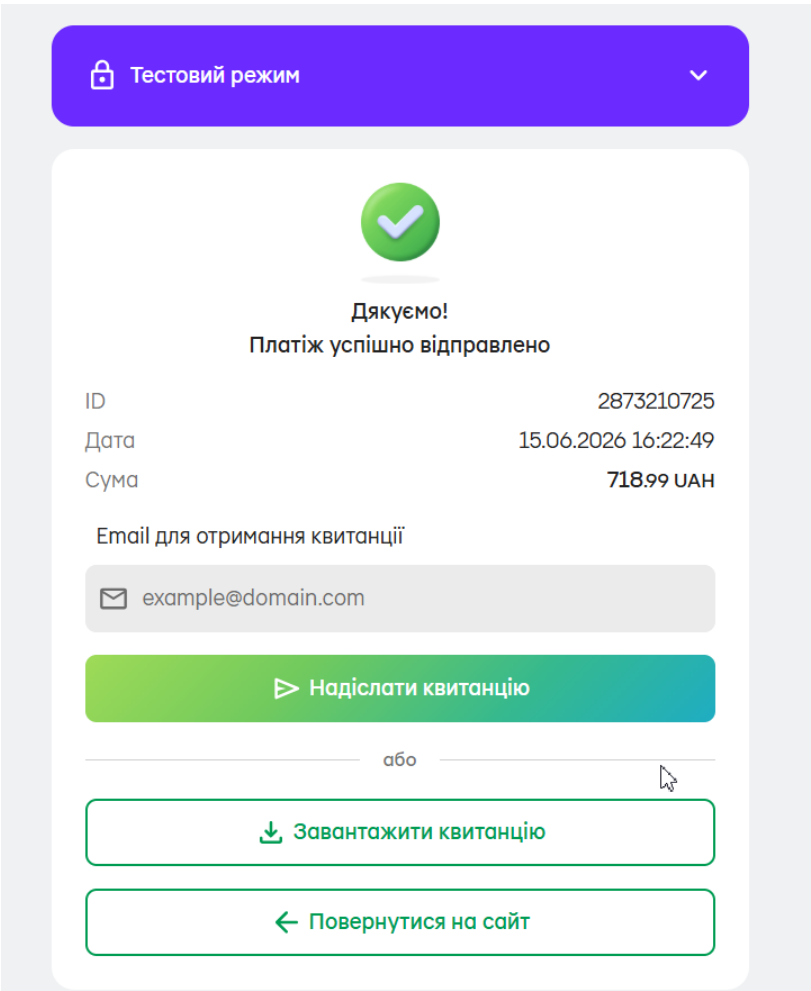


Рис. 3.30 – Успішна оплата товару

Якщо користувач не завершує оплату одразу, для замовлення з онлайн-оплатою встановлюється обмежений час її виконання. На сторінці відображається таймер із залишком часу для здійснення платежу, що наведено на рис. 3.31.

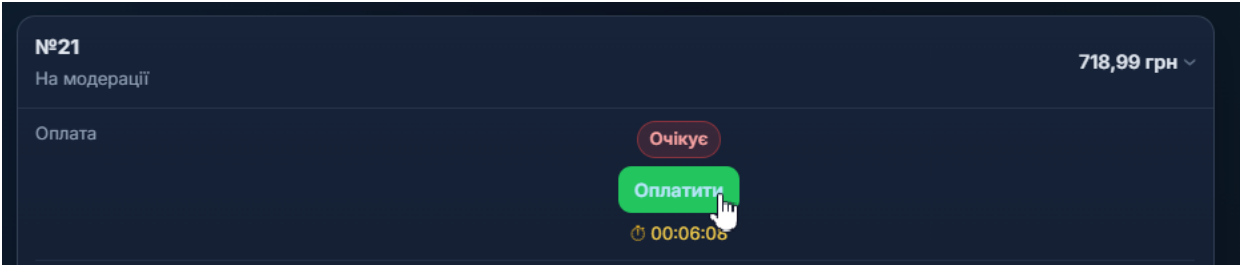


Рис. 3.31 – У разі несплати одразу, діє таймер на оплату 10 хв

Особистий кабінет користувача містить основні розділи для роботи з профілем, відгуками та замовленнями. У вкладці «Профіль» відображається основна інформація про користувача та доступні налаштування облікового запису. Відповідний інтерфейс наведено на рис. 3.32.

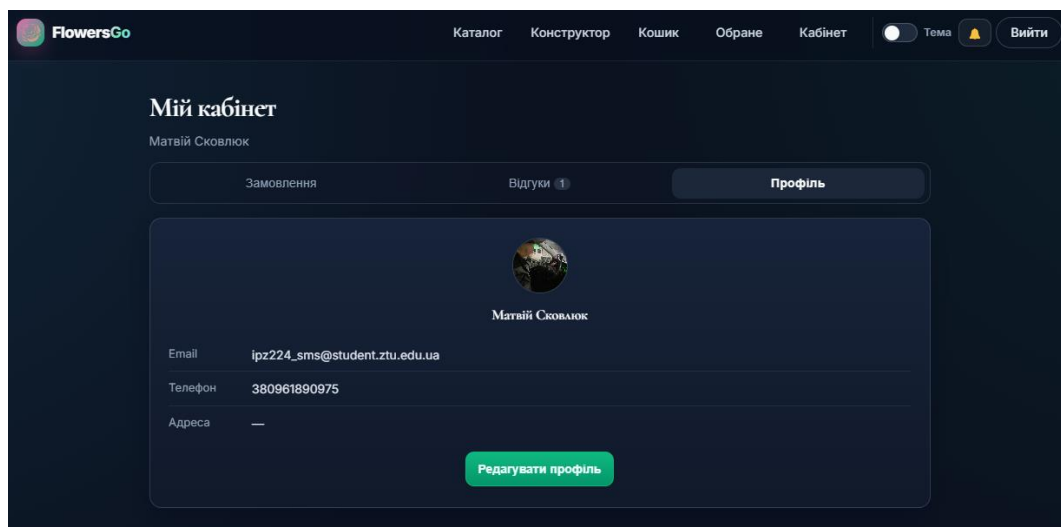


Рис. 3.32 – Вкладка “Профіль” у кабінеті користувача

Користувач може змінювати особисту інформацію, зокрема контактні дані та аватар. Форма редагування профілю наведена на рис. 3.33.

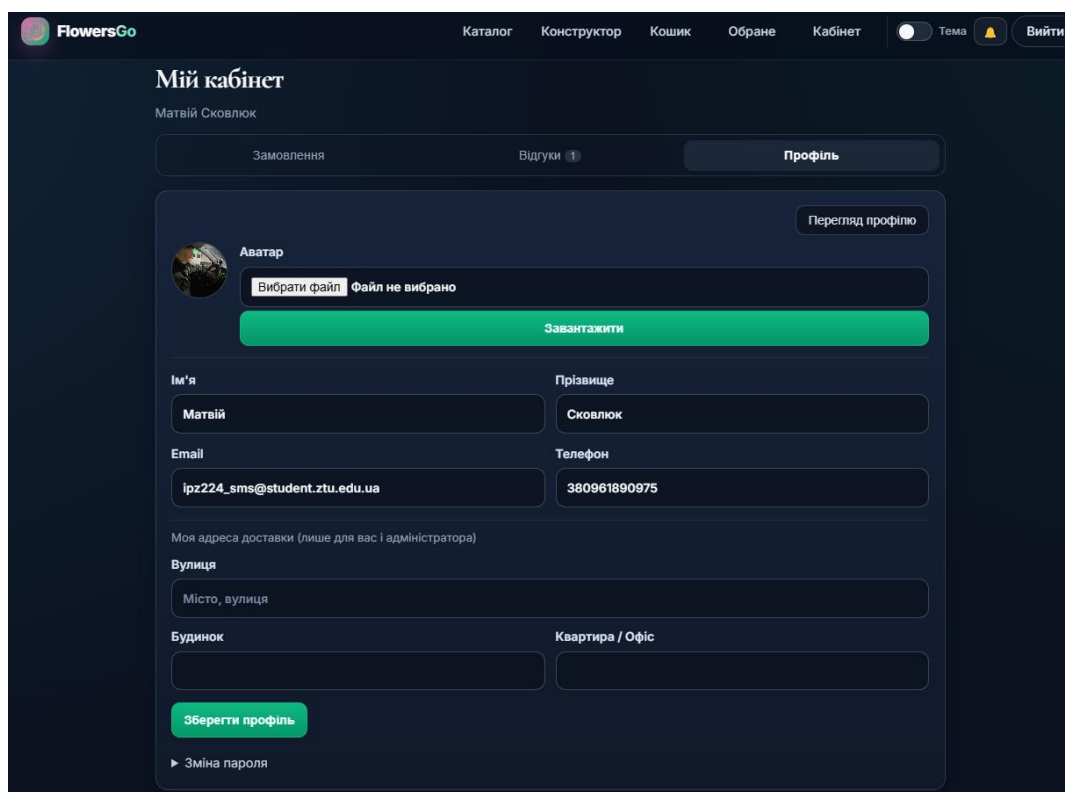


Рис. 3.33 – Редагування особистої інформації користувача

Окремо передбачено можливість зміни пароля облікового запису. Для цього у вкладці «Профіль» відкривається відповідна форма редагування, показана на рис. 3.34.

The screenshot shows a dark-themed user profile page. At the top left, there is a green button labeled 'Зберегти профіль'. Below it, a dropdown menu is open, showing 'Зміна пароля'. Under this menu, there are two input fields: 'Новий пароль' (New password) and 'Підтвердження' (Confirmation). At the bottom of this section is a large green button labeled 'Змінити пароль' (Change password).

Рис. 3.34 – Форма зміни пароля в кабінеті користувача у вкладці “Профіль”, режим редагування

У вкладці «Відгуки» користувач може переглядати залишені ним відгуки та за потреби подати заявку на їх редагування або видалення. Приклад такого інтерфейсу наведено на рис. 3.35.

The screenshot shows a user's cabinet page titled 'Мій кабінет' for user 'Матвій Сковлюк'. There are three tabs: 'Замовлення', 'Відгуки 1' (selected), and 'Профіль'. The main content area shows a review for 'Опт-набір «50 подарункових мі...»' with a rating of 5 stars and a status of 'Опубліковано'. The review text is 'гарно' and the date is '15.04.26, 03:19'. There is a 'Видалити' (Delete) button. Below the review, there is a section for 'Оцінка' (Rating) with a dropdown menu showing '5', and a 'Текст' (Text) field with the text 'гарно'. At the bottom of this section are two buttons: 'Надіслати' (Send) and 'Скасувати' (Cancel).

Рис. 3.35 – Перегляд вкладки відгуки користувача, з можливістю подати заявку на видалення чи редагування власного відгуку

Вкладка «Замовлення» містить історію оформлених користувачем замовлень. У ній можна переглянути основну інформацію про замовлення та його поточний статус. Приклад відображення історії замовлень наведено на рис. 3.36.

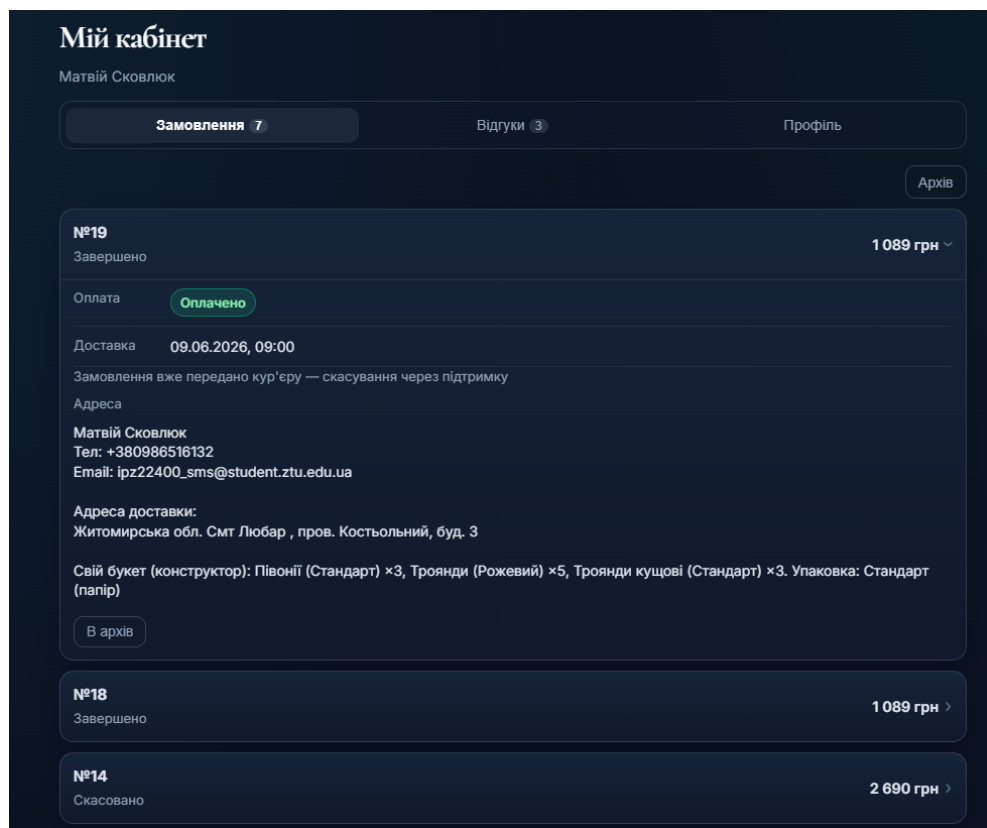


Рис. 3.36 – Перегляд вкладки замовлення, історія особистих замовлень

Для авторизованих користувачів у шапці вебплатформи передбачено іконку сповіщень. Через неї клієнт отримує інформацію про зміну статусу замовлення, відповідь служби підтримки та результати розгляду відгуків. Відображення іконки сповіщень у верхній частині інтерфейсу наведено на рис. 3.37.

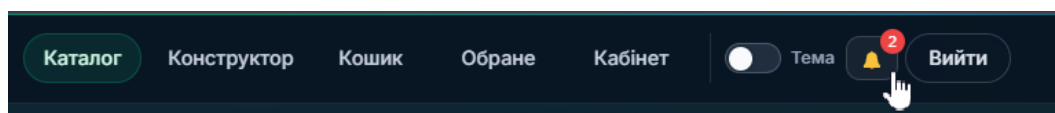


Рис. 3.37 – Іконка дзвоника в хедері для авторизованих користувачів

Після натискання на іконку відкривається список отриманих сповіщень. Користувач також може переглянути доступні налаштування сповіщень, що показано на рис. 3.38.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						75

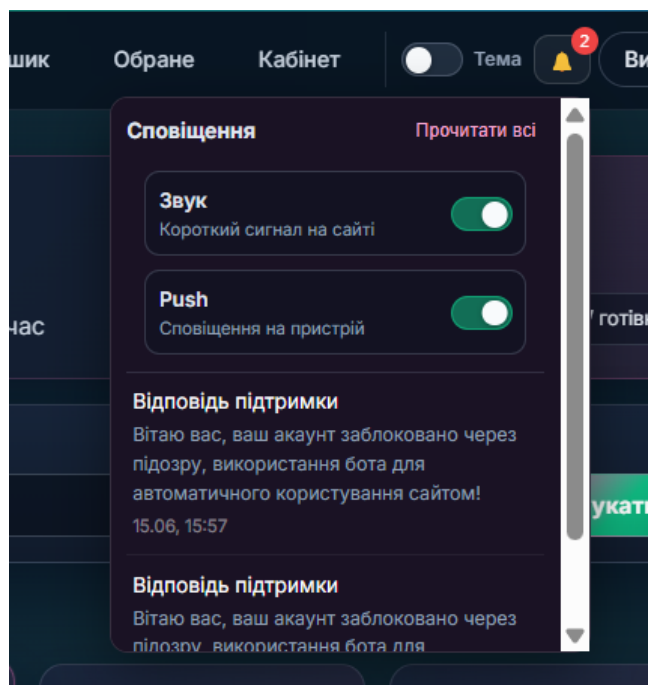


Рис. 3.38 – Вікно сповіщень з налаштуваннями

Для комунікації зі службою підтримки на вебплатформі передбачено окремий чат. Віджет підтримки розміщений у нижній частині сторінки та доступний гостям і авторизованим користувачам. Загальний вигляд вікна підтримки наведено на рис. 3.39.

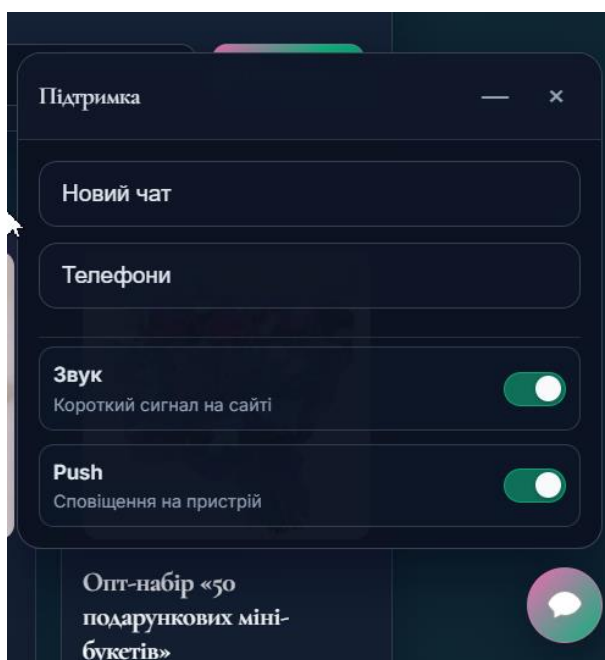


Рис. 3.39 – Вікно підтримки

Неавторизований користувач може відкрити чат та надіслати звернення без входу до облікового запису. Приклад вікна чату для гостя наведено на рис. 3.40.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				76

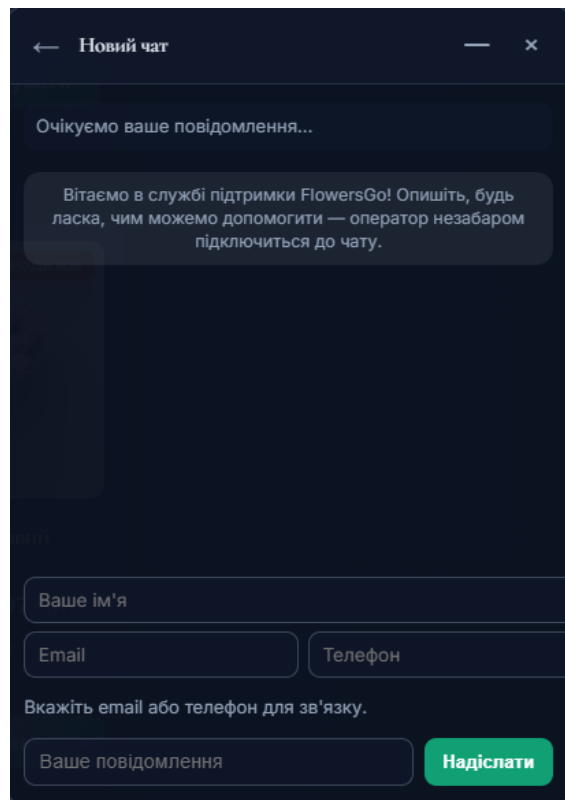


Рис. 3.40 – Вікно чату неавторизованого користувача

Авторизованому користувачу доступний аналогічний механізм звернення до служби підтримки. Відмінність полягає у тому, що звернення пов'язується з його обліковим записом. Відповідний інтерфейс наведено на рис. 3.41.

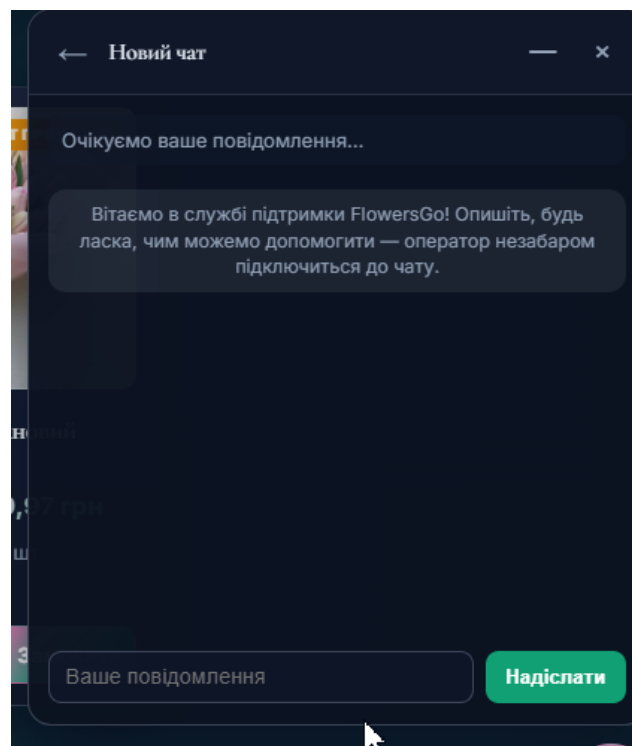


Рис. 3.41 – Вікно чату авторизованого користувача

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				77

Після введення тексту повідомлення користувач може надіслати його до служби підтримки. Приклад відправлення повідомлення наведено на рис. 3.42.

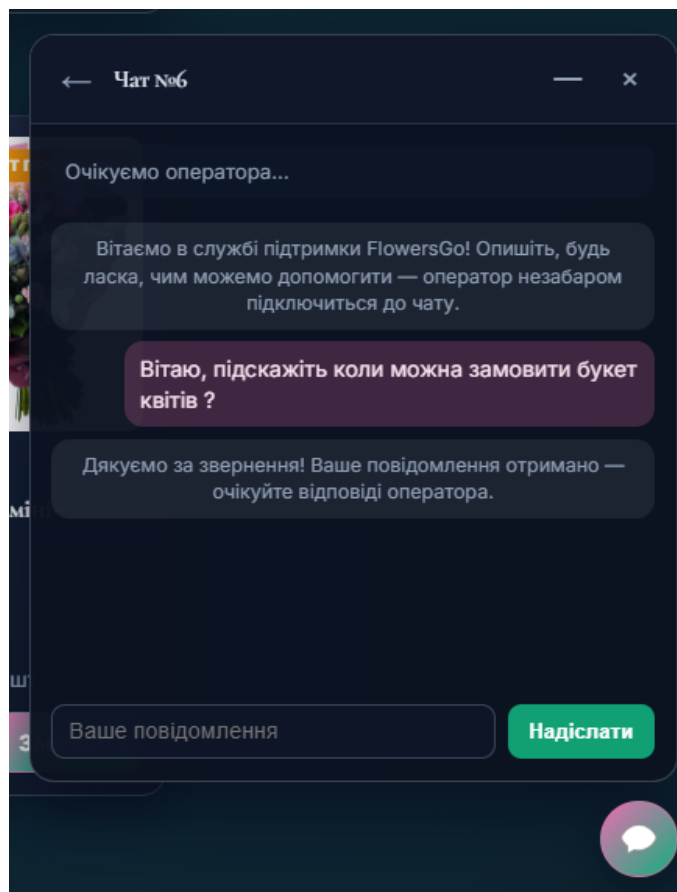


Рис. 3.42 – Вікно чату надіслане користувачем повідомлення

Надіслане звернення відображається в адміністративній панелі, де адміністратор може переглянути його та розпочати опрацювання. Вигляд повідомлення зі сторони адміністратора наведено на рис. 3.43.

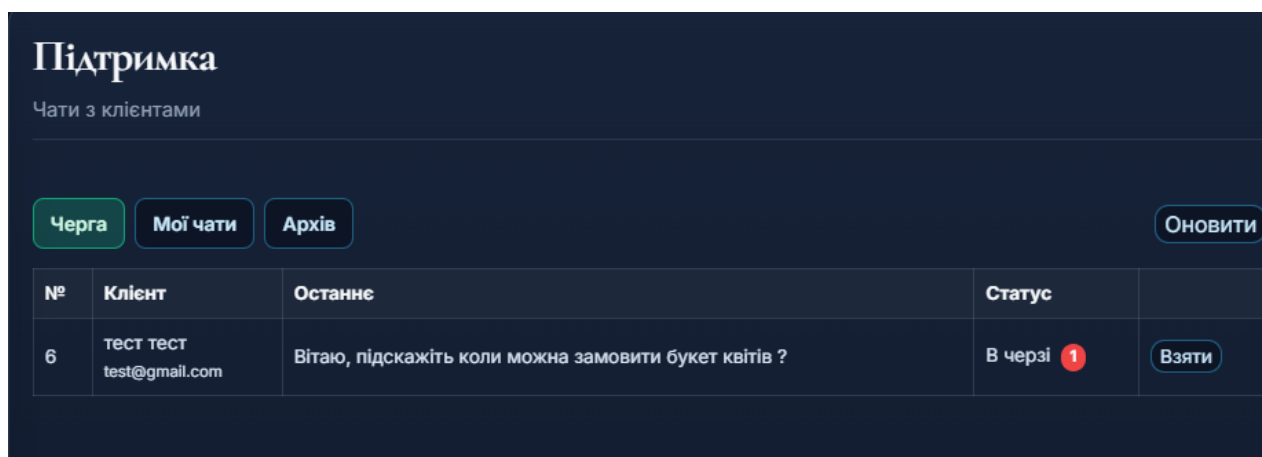


Рис. 3.43 – Повідомлення надіслане адміністратору, вигляд зі сторони панелі адміністратора

Після відкриття звернення адміністратор може продовжити листування з користувачем безпосередньо через панель керування. Приклад такого листування наведено на рис. 3.44.

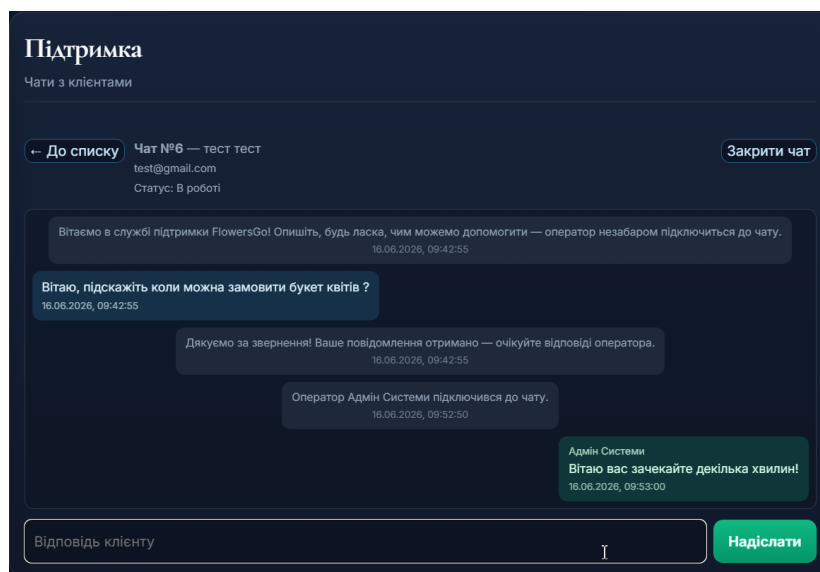


Рис. 3.44 – Листування з користувачем зі сторони адміністратора

Відповідь адміністратора відображається у чаті користувача. Після отримання відповіді клієнт може продовжити спілкування зі службою підтримки, що показано на рис. 3.45.

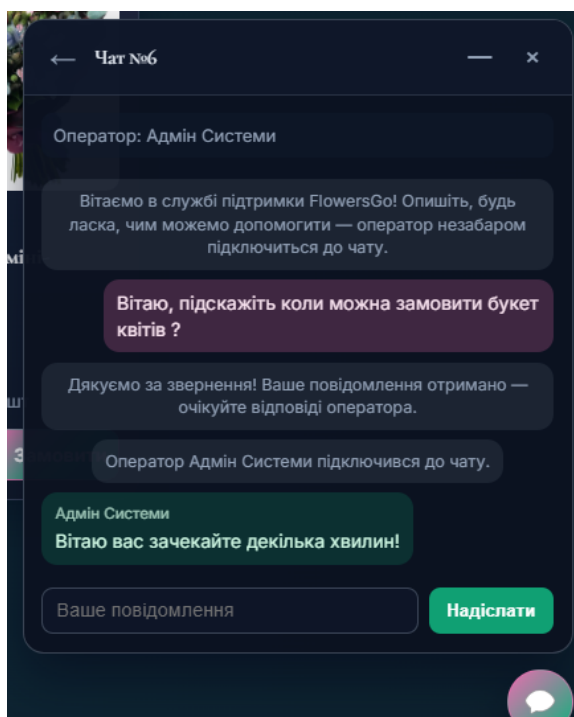


Рис. 3.45 – Стан чату зі сторони користувача після отримання відповіді підтримки

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				79

В адміністративній панелі також передбачено окремий перегляд чатів, які адміністратор взяв на опрацювання. Це дозволяє розподіляти звернення та контролювати їх поточний стан. Відповідний розділ наведено на рис. 3.46.

№	Клієнт	Останнє	Статус	
6	тест тест test@gmail.com	Вітаю вас зачекайте декілька хвилин!	В роботі	Відкрити
5	Матвій Сковлюк ipz22400_sms@student.ztu.edu.ua	Вітаю вас, ваш акаунт заблоковано через підозру, використанн	В роботі	Відкрити

Рис. 3.46 – Перегляд чатів адміністратором, які він взяв на опрацювання

Після завершення спілкування адміністратор може закрити звернення. Закриті чати переміщуються до архіву, де їх можна переглянути за потреби. Приклад архіву звернень наведено на рис. 3.47.

№	Клієнт	Останнє	Закрито	
4	Матвей Сковлюк skovlyukv@gmail.com	Діалог завершено. Дякуємо за звернення! Щоб написати знову —	11.06.2026, 19:59:07	Переглянути
2	Mot Google sasha.jesik13@gmail.com	Діалог завершено. Дякуємо за звернення! Щоб написати знову —	09.06.2026, 23:39:53	Переглянути
3	Матвей Сковлюк skovlyukv@gmail.com	Діалог завершено. Дякуємо за звернення! Щоб написати знову —	09.06.2026, 23:37:57	Переглянути
1	Mot Google sasha.jesik13@gmail.com	Діалог завершено. Дякуємо за звернення! Щоб написати знову —	09.06.2026, 21:01:31	Переглянути

Рис. 3.47 – Перегляд архіву чатів, які адміністратор закрив після спілкування

Користувач із роллю адміністратора працює в окремій панелі керування за адресою /admin. Навігаційне меню панелі згруповано за основними напрямками роботи: каталог, замовлення, клієнти та налаштування. Окремо передбачено розділи «Огляд» і «Статистика», які використовуються для перегляду поточного стану магазину та аналізу його роботи. Структуру навігаційного меню у згорнутому вигляді наведено на рис. 3.48.

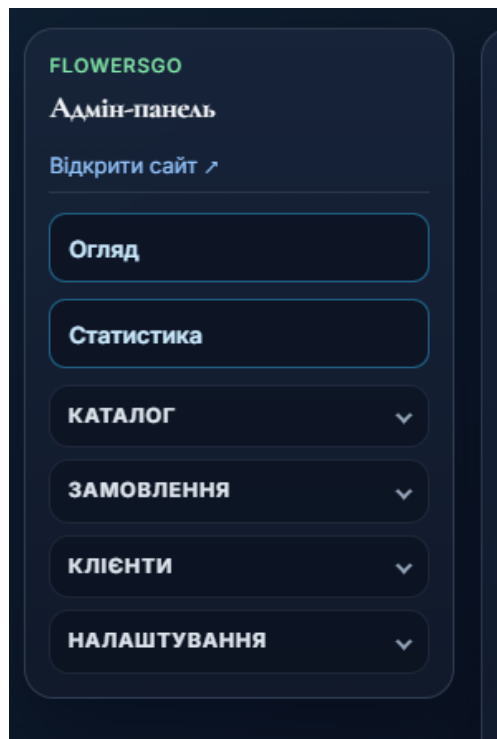


Рис. 3.48 – Згорнутий блок панелі керування адміністратора за категоріями

Після переходу до розділу «Огляд» адміністратор отримує коротке зведення поточного стану системи. На сторінці відображається інформація про товари, замовлення, модерацію відгуків, підтримку, конструктор букетів і доставку, а також блок із повідомленнями, які потребують уваги. Головну сторінку адміністративної панелі наведено на рис. 3.49.

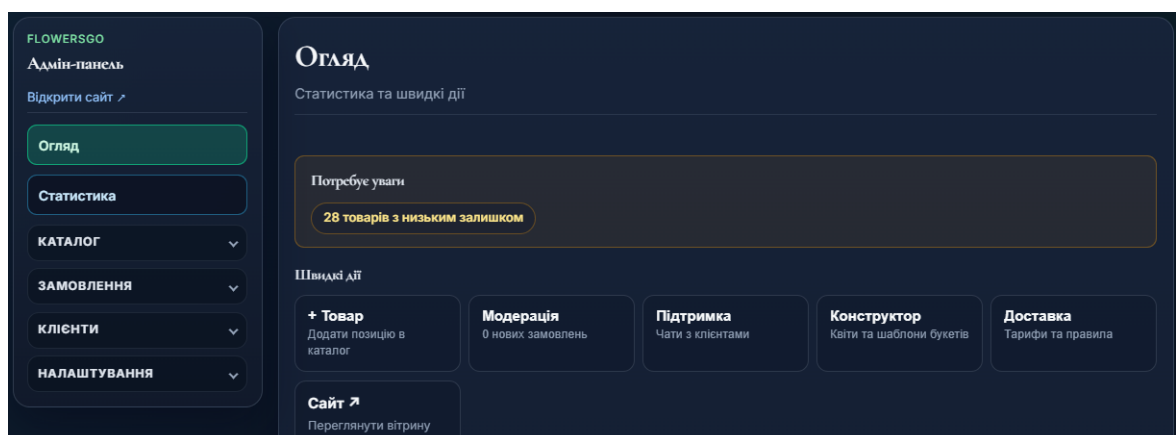


Рис. 3.49 – Головна сторінка панелі адміністратора

Вкладка «Статистика» призначена для аналізу роботи магазину за вибраний період. Адміністратор може застосовувати фільтри за способом оплати, типом доставки, категорією товару та статусом замовлення. Основну частину вкладки «Статистика» наведено на рис. 3.50.

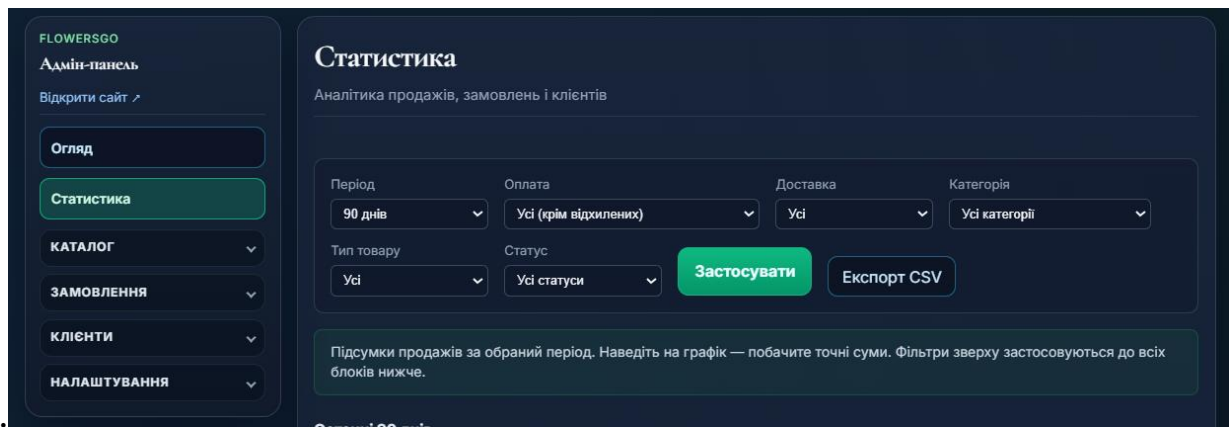


Рис. 3.50 – Вкладка статистика для адміністратора

На продовженні сторінки відображаються додаткові графіки та показники, що дають змогу детальніше оцінити результати продажів за вибраними параметрами. Відповідний фрагмент інтерфейсу наведено на рис. 3.51.

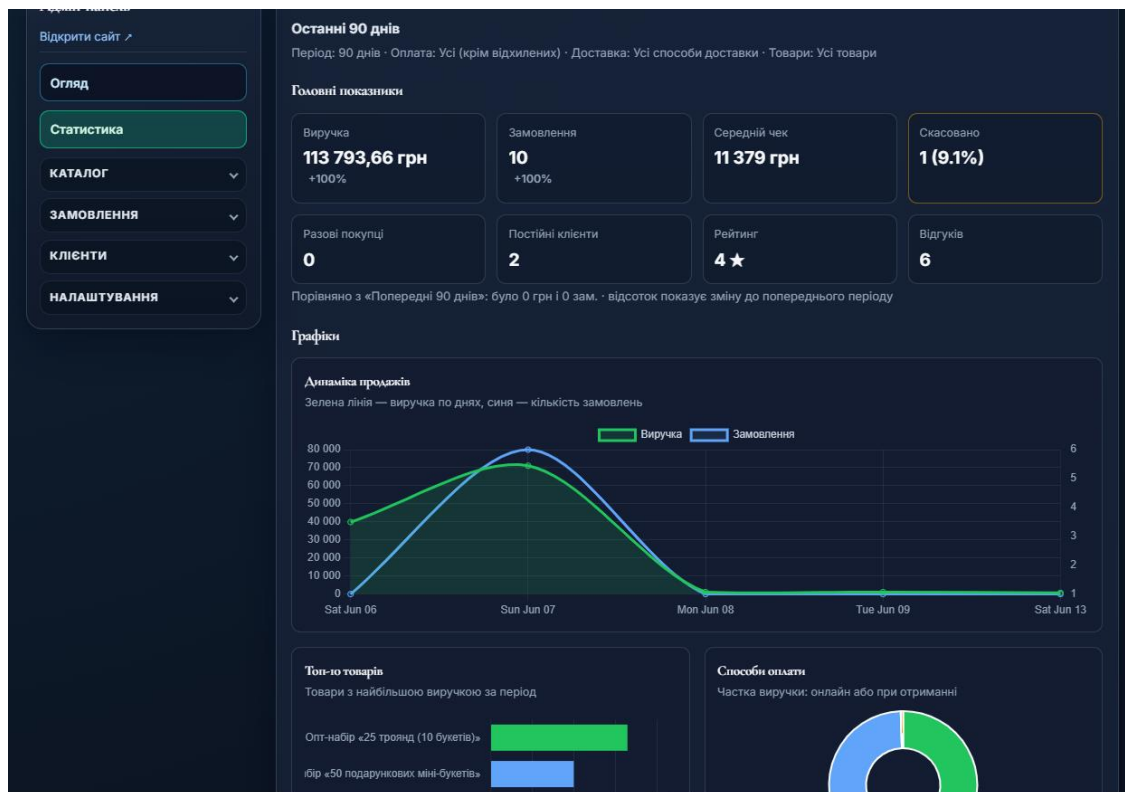


Рис. 3.51 – Продовження вкладки статистики для адміністратора

На продовженні сторінки відображаються додаткові графіки та показники, що дають змогу детальніше оцінити результати продажів за вибраними параметрами. Відповідний фрагмент вкладки «Статистика» наведено на рис. 3.51.

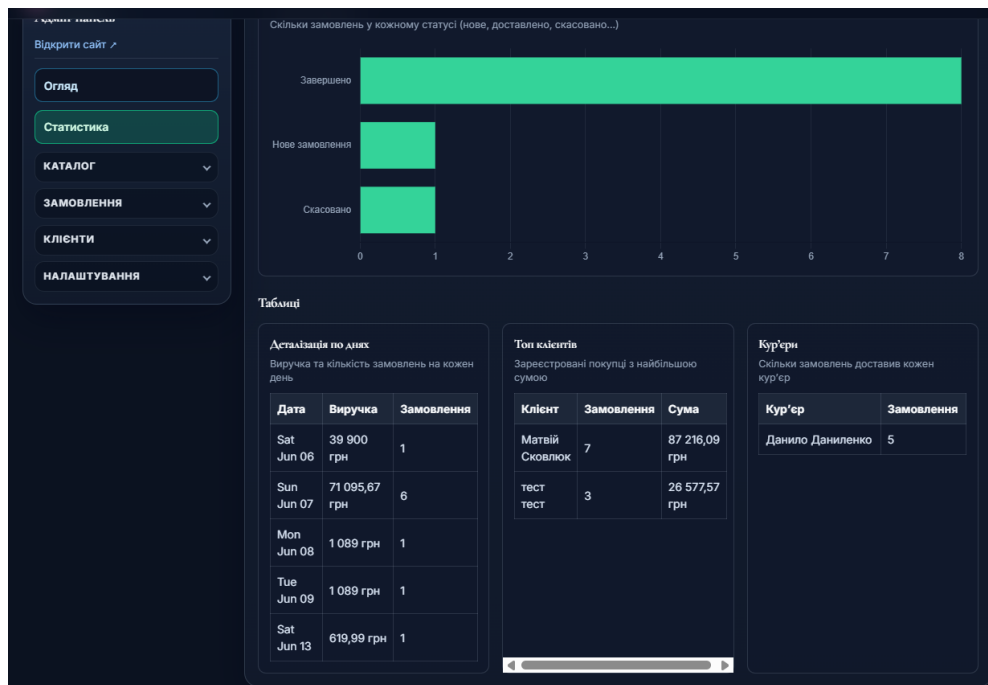


Рис. 3.52 – Продовження вкладки статистики для адміністратора

Розділ «Товари» призначений для керування асортиментом магазину. Адміністратор може додавати нові позиції до каталогу, задавати їх основні характеристики, ціну, кількість доступного товару та зображення. Форма створення нового товару наведена на рис. 3.53

Новий товар
Додавання позиції в каталог

Додавання товару

Категорія:

Назва:

Опис:

Базова ціна:

Знижка: ☐ Підсумкова ціна: 0.00 грн (без знижки)

Кількість на складі:

Одиниця:

Активний:

Для конструктора:

Фото: Файл не вибрано

Створити товар

Рис. 3.53 – Сторінка додавання нового товару

Для структурування асортименту в адміністративній панелі передбачено окремий розділ «Категорії». У ньому адміністратор може створювати та переглядати категорії товарів. Відповідний інтерфейс наведено на рис. 3.54.

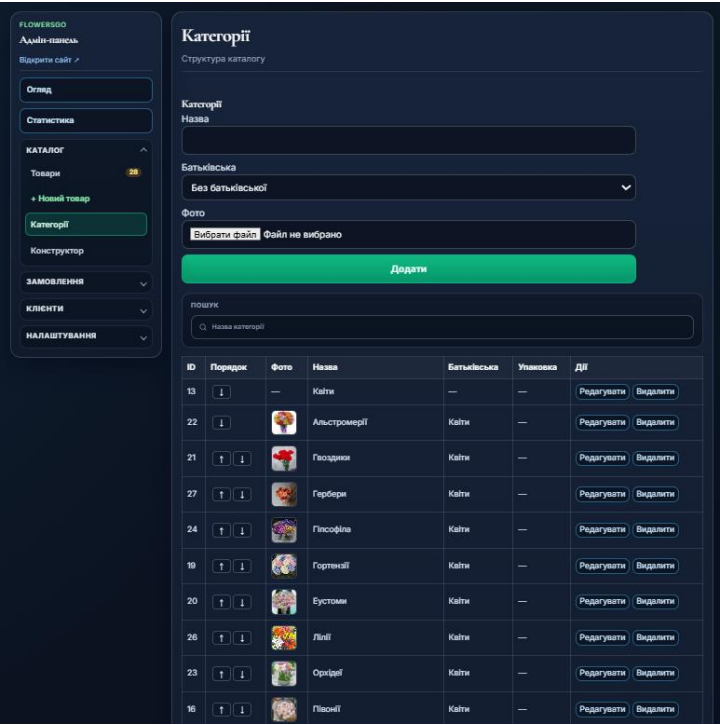


Рис. 3.54 – Сторінка додавання та керування категоріями

Створені категорії можна редагувати, змінюючи їх назву та інші доступні параметри. Приклад редагування категорії наведено на рис. 3.55.

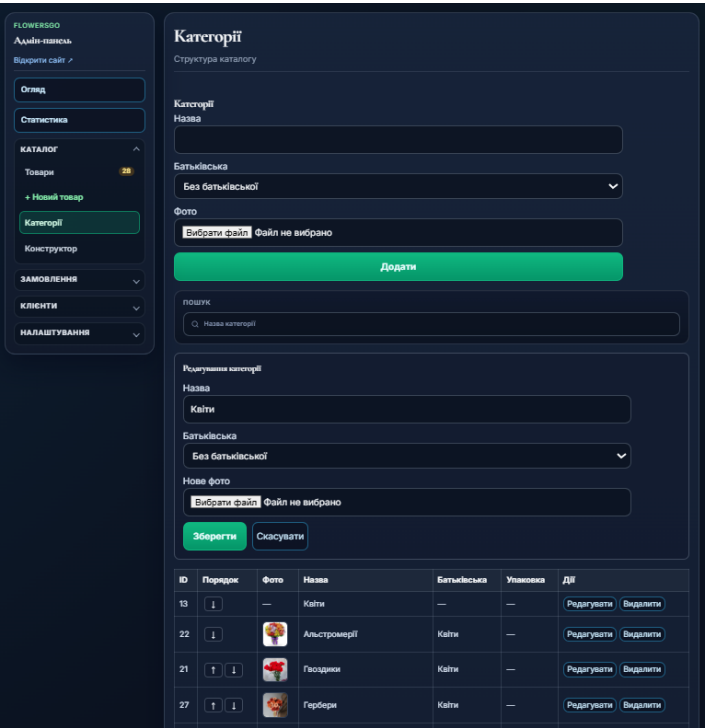


Рис. 3.55 – Редагування категорії

Окремий функціонал адміністративної панелі призначений для налаштування конструктора букетів. Адміністратор може визначати товари, які будуть доступні користувачеві під час формування власної композиції. Приклад додавання товару до конструктора наведено на рис. 3.56.

Рис. 3.56 – Додавання нового товару в конструктор букетів

Крім окремих товарів, адміністратор може створювати готові шаблони букетів, які користувач може використати як основу для власної композиції. Форма додавання такого шаблонного рішення наведена на рис. 3.57.

Бібліотека готових букетів

Додати шаблон

ID	Назва	Позицій	Квіток	Ціна	Активний	Дія
1	Класика — 11 червоних троянд	1	11	994 грн	Так	РедагуватиВимкнутиВидалити
2	Весняний — 15 тюльпанів	1	0	—	Так	РедагуватиВимкнутиВидалити

Рис. 3.57 – Додавання готових шаблонних рішень в конструктор букетів

Робота із замовленнями в адміністративній панелі поділена на окремі етапи. Нові заявки спочатку надходять на модерацію, де адміністратор може переглянути основну інформацію та прийняти рішення щодо подальшої обробки замовлення. Сторінку модерації нових замовлень наведено на рис. 3.58.

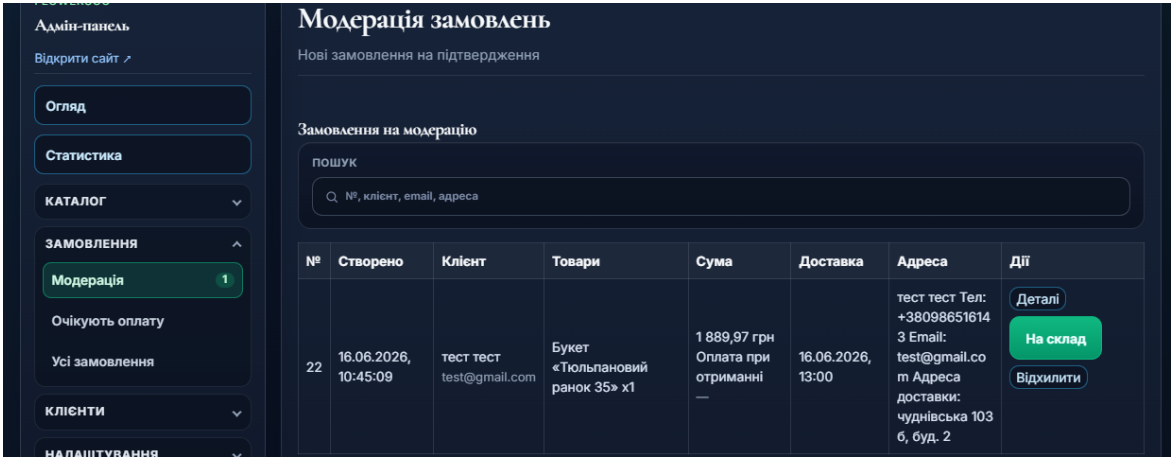


Рис. 3.58 – Модерація нових замовлень

Для перевірки окремого замовлення адміністратор може відкрити його детальну інформацію, переглянути склад, дані клієнта, спосіб оплати та доставки. Приклад детального перегляду замовлення наведено на рис. 3.59.

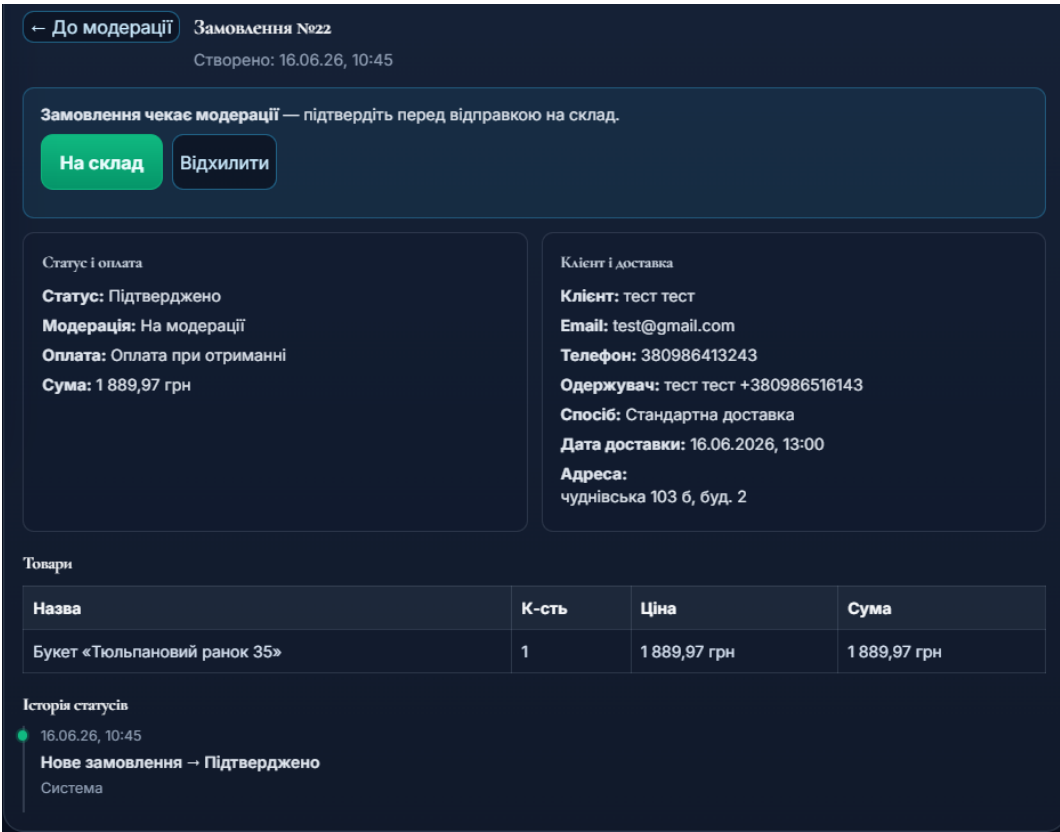


Рис. 3.59 – Детальний огляд замовлення

Окремий розділ призначений для контролю замовлень, за якими очікується оплата через LiqPay. Для таких замовлень відображається поточний стан платежу та встановлений час для його завершення. Сторінку замовлень, що очікують оплати, наведено на рис. 3.60.

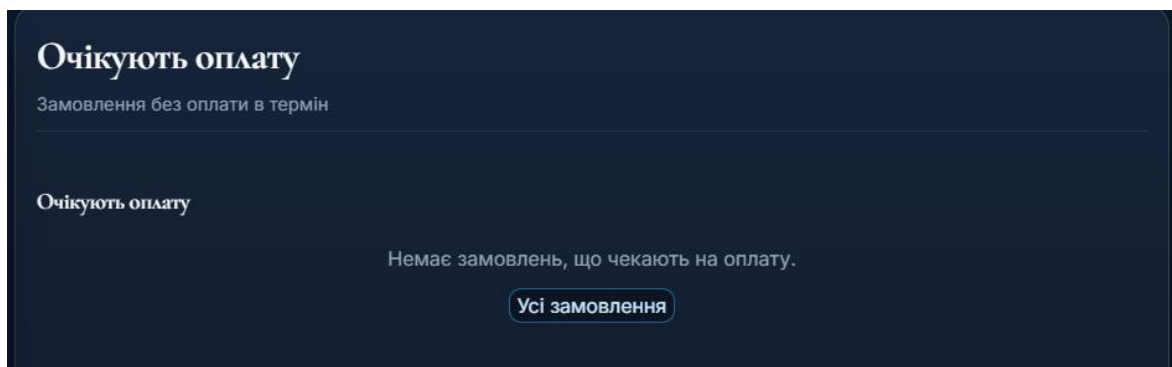


Рис. 3.60 – Огляд замовлень на очікування оплати

У загальному реєстрі адміністратор може переглядати всі створені замовлення, виконувати пошук і сортування, а також контролювати їх поточні статуси. Відповідний інтерфейс реєстру замовлень наведено на рис. 3.61.

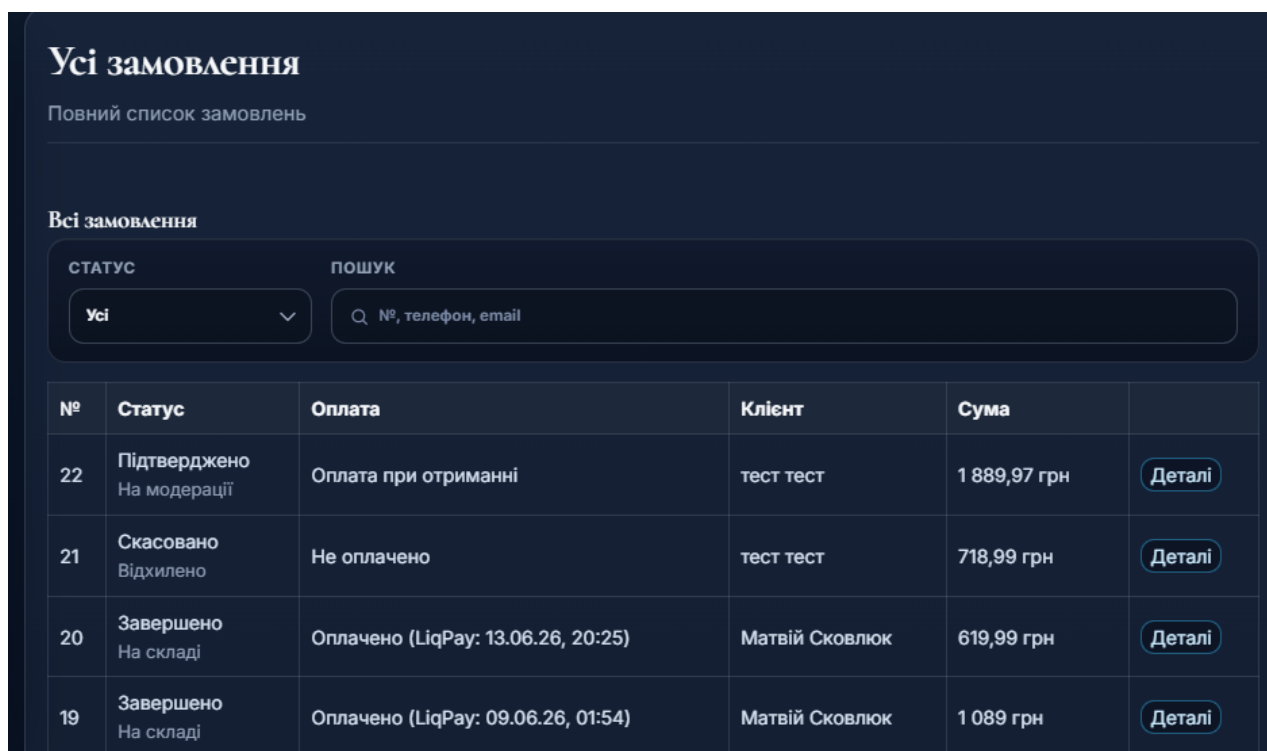


Рис. 3.61 – Перегляд усіх замовлень з пошуком та сортуванням

Окремі розділи адміністративної панелі призначені для модерації відгуків і опрацювання запитів на їх редагування або видалення. Інтерфейс модерації відгуків наведено на рис. 3.62.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						87

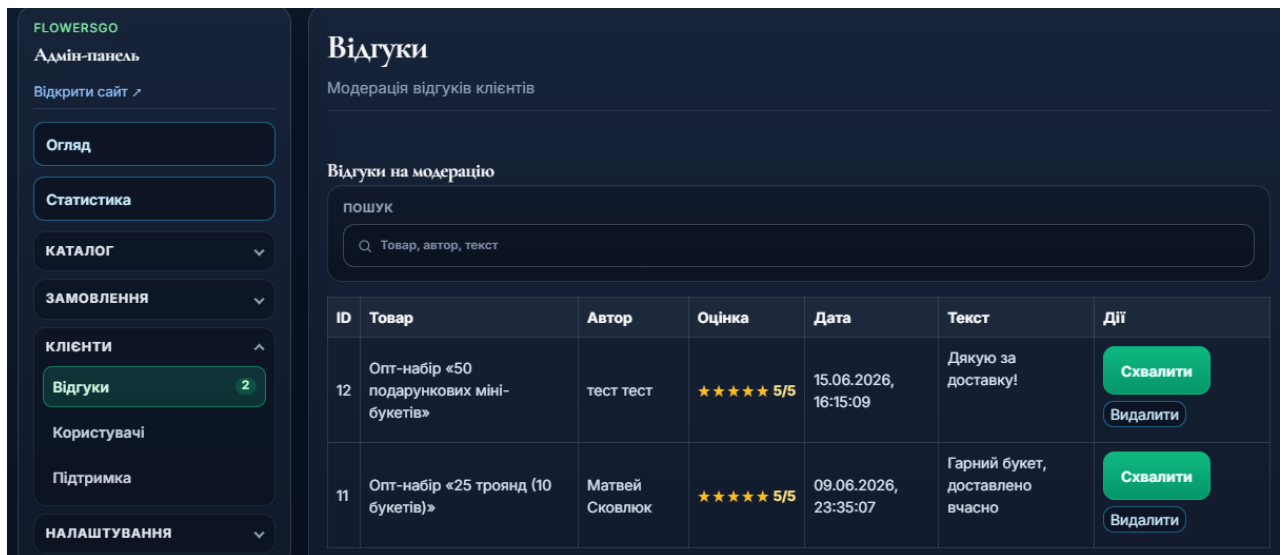


Рис. 3.62 – Сторінка відгуків на модерації

Для керування обліковими записами використовується розділ «Користувачі». Він містить перелік зареєстрованих користувачів та основну інформацію про їхні облікові записи. Вигляд відповідної сторінки наведено на рис. 3.63.

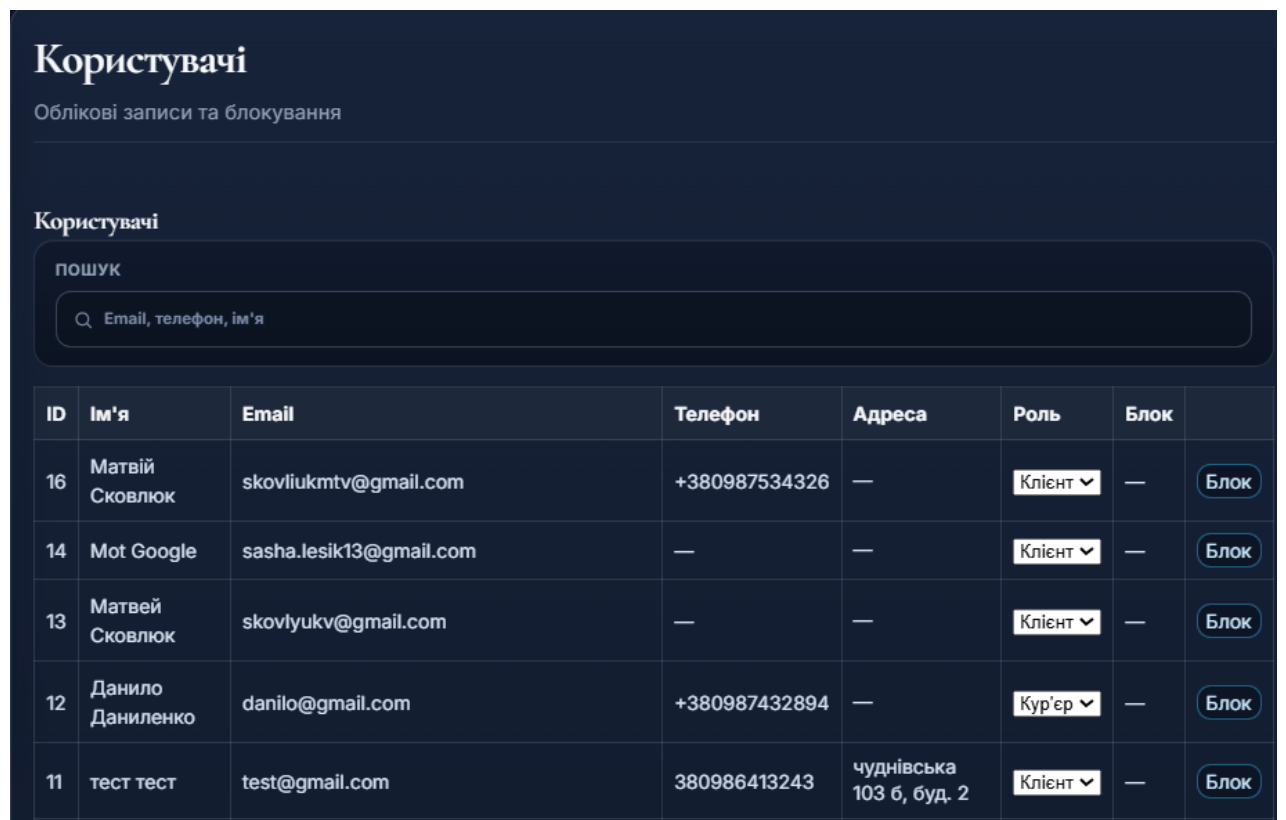


Рис. 3.63 – Вивід користувачів сайту та керування ними

У розділі налаштувань адміністратор може задавати параметри, які використовуються під час оформлення та обробки замовлень. Зокрема, налаштовуються робочі години, часові слоти доставки, буфери для стандартної та експрес-доставки, а також параметри самовивозу. Інтерфейс налаштування параметрів доставки наведено на рис. 3.64.

FLOWERSGO
Адмін-панель
Відкрити сайт

Доставка
Тарифи та правила доставки

Налаштування доставки

Робота з: 9

Робота до: 23

Слот (год): 2

Буфер стандарт (хв): 120

Буфер експрес (хв): 60

Самовивіз з: 9

Самовивіз до: 19

Буфер самовивіз (хв):

Огляд
Статистика
КАТАЛОГ
ЗАМОВЛЕННЯ
КЛІЕНТИ
НАЛАШТУВАННЯ
Доставка та тарифи
Кур'єри
Юридичні сторінки

Рис. 3.64 – Налаштування годин доставки

Окремий розділ призначений для керування кур'єрами. Адміністратор може переглядати список працівників, змінювати їхній статус та керувати доступністю для виконання доставки. Сторінку керування кур'єрами наведено на рис. 3.65.

FLOWERSGO
Адмін-панель
Відкрити сайт

Кур'єри
Команда доставки

Кур'єри Авто-призначити готові Без кур'єра (0)

На зміні: 1 / 1 Без кур'єра: 0

пошук
Q: Ім'я, email

Ім'я	Особистий email	Робочий email	Зміна	На складі	В дорозі	Активних	Дії
Данило Даниленко	daniilo@gmail.com	skovliukmtv@gmail.com	На зміні	0	0	0	Зняти зі зміни Email

«На складі» — замовлення в комплектції/готові. «В дорозі» — статус shipped. Увімкнення зміни запускає автопризначення готових замовлень.

Огляд
Статистика
КАТАЛОГ
ЗАМОВЛЕННЯ
КЛІЕНТИ
НАЛАШТУВАННЯ
Доставка та тарифи
Кур'єри
Юридичні сторінки

Рис. 3.65 – Сторінка кур'єри

У розділі «Юридичні сторінки» адміністратор може редагувати інформаційні матеріали, які відображаються користувачам вебплатформи. Це дає змогу підтримувати актуальність інформації без внесення змін безпосередньо до програмного коду. Відповідний інтерфейс наведено на рис. 3.66.

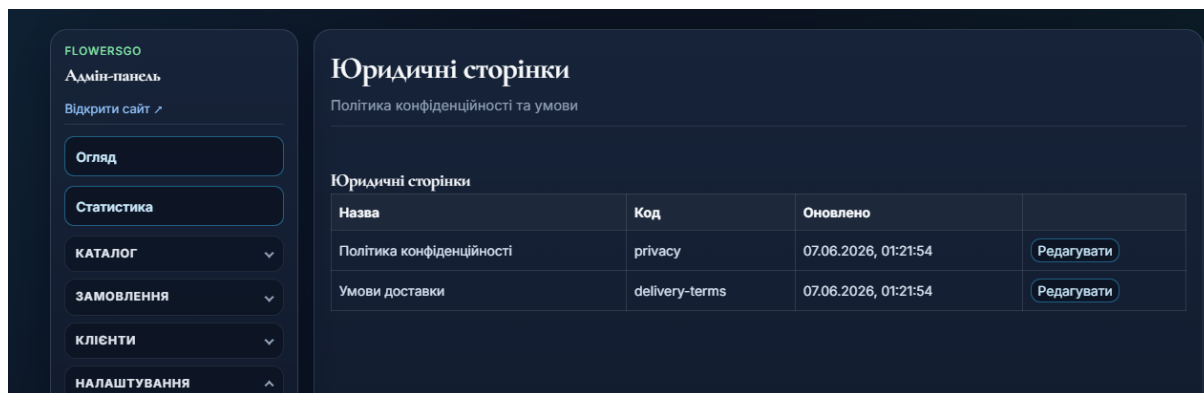


Рис. 3.66 – Сторінка юридичні сторінки

Інтерфейс працівника складу призначений для роботи із замовленнями, які пройшли модерацію адміністратора. У відповідному розділі формується черга замовлень, яку можна фільтрувати та сортувати за датою, пріоритетом і поточним статусом. Перегляд черги замовлень на складі наведено на рис. 3.67.

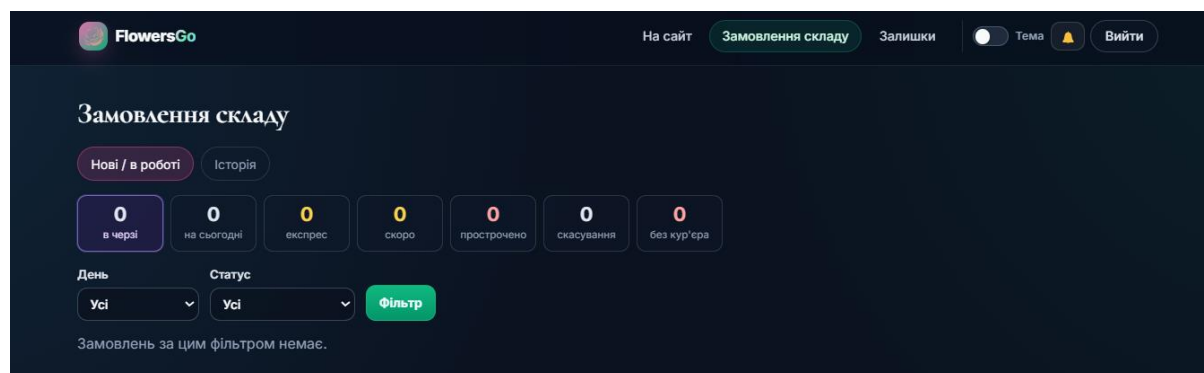


Рис. 3.67 – Перегляд нових замовлень на складі

Для кожного замовлення працівник може переглянути детальну інформацію, зокрема склад букета, примітки клієнта та QR-код. Після завершення комплектації працівник змінює відповідний статус, формує лист для складу та передає готове замовлення кур'єру або клієнту у випадку самовивозу. Історію оброблених замовлень наведено на рис. 3.68.

Історія замовлень

Нові / в роботі

Історія

Показати

День

Статус

Виконані

Усі

Усі

Фільтр

У фільтрі: 8

№20 14.06.2026, 09:00

619,99 грн

Завершено

Оплачено

Стандарт

ЗІБРАТИ:

Плюшевий зайчик 35 см ×1

Кому

Матвій Сковлюк

Клієнт

—

Кур'єр

Не призначено

Адреса

вулиця Чуднівська, 103, Житомир, Житомирська область, Україна, 10001, буд. 3

Дзвінок

Viber

Telegram

Змінити статус вручну

Деталі / друк

Рис. 3.68 – Історія замовлень на складі

Окремий розділ призначений для контролю залишків товарів. У ньому відображається доступна кількість продукції та її резерв під замовлення. Працівник складу може скоригувати залишок, при цьому відповідна операція зберігається в системі для подальшого контролю. Інтерфейс перегляду залишків наведено на рис. 3.69.

Залишки на складі

Списання — після підтвердження адміном · коригування фіксується в журналі

← Замовлення складу

Позицій

Мало (≤ 5)

Немає (0)

Дефіцит

У резерві

110

13

1

0

1

Пошук

Тип

Стан

Назва або колір

Усі

Усі

Застосувати

Товар	Тип	Категорія	На складі	Резерв	Вільно	Коригування
Опт-набір «25 троянд (10 букетів)»	Каталог	Оптові ціни	0 шт Немає	—	0	-1 ± +1 ОК
Букет «Преміум» — у шляпній коробці	Каталог	Півонії	3 шт Мало	—	3	-1 ± +1 ОК
Букет «Deluxe» — у hat-box	Каталог	Гортензії	3 шт Мало	—	3	-1 ± +1 ОК

Рис. 3.69 – Перегляд залишків на складі

Конструктор букетів призначений для формування індивідуальної композиції відповідно до вибору користувача. На сторінці конструктора можна обрати квіти, їх колір, кількість стебел та упаковку. Під час зміни параметрів система перераховує вартість сформованої композиції та перевіряє наявність необхідних позицій. Користувач може змінювати склад композиції до отримання бажаного варіанта, при цьому вибрані позиції та їх кількість відображаються в окремому блоці конструктора. Інтерфейс конструктора букетів наведено на рис. 3.70.

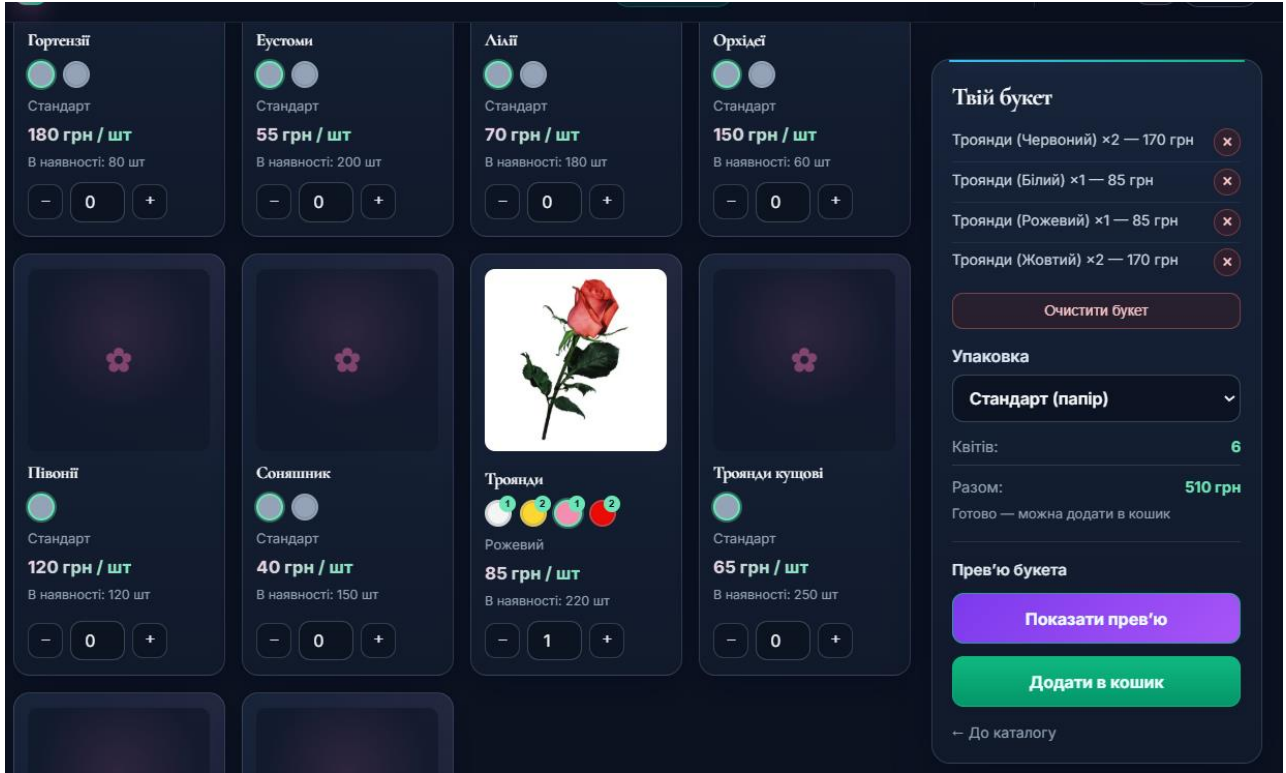


Рис. 3.70 – Конструктор букетів

Після завершення вибору система формує зібрану композицію та відображає її користувачу у вигляді ілюстративного прев'ю. Сформоване зображення може використовуватися під час подальшого оформлення замовлення та передаватися працівнику складу як додаткова інформація про склад букета. У блоці попереднього перегляду також відображаються основні параметри сформованої композиції та її підсумкова вартість. Приклад сформованого букета наведено на рис. 3.71.

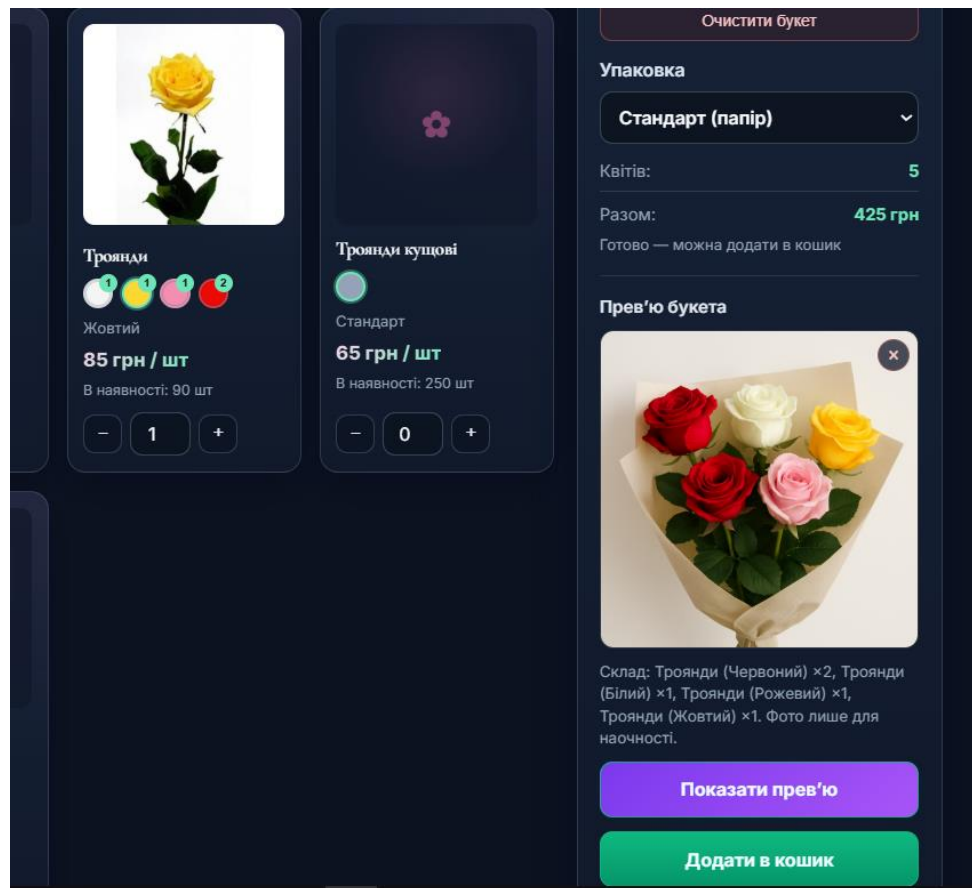


Рис. 3.71 – Генерація зібраного букета

Доставка у FlowersGo підтримує кілька варіантів отримання замовлення: стандартний часовий слот, доставку на точний час, експрес-доставку та самовивіз. Відповідні параметри та тарифи налаштовуються адміністратором у системі. Для роботи кур'єра передбачено окремий інтерфейс, у якому він може розпочати або завершити робочу зміну. Відповідний інтерфейс наведено на рис. 3.72.

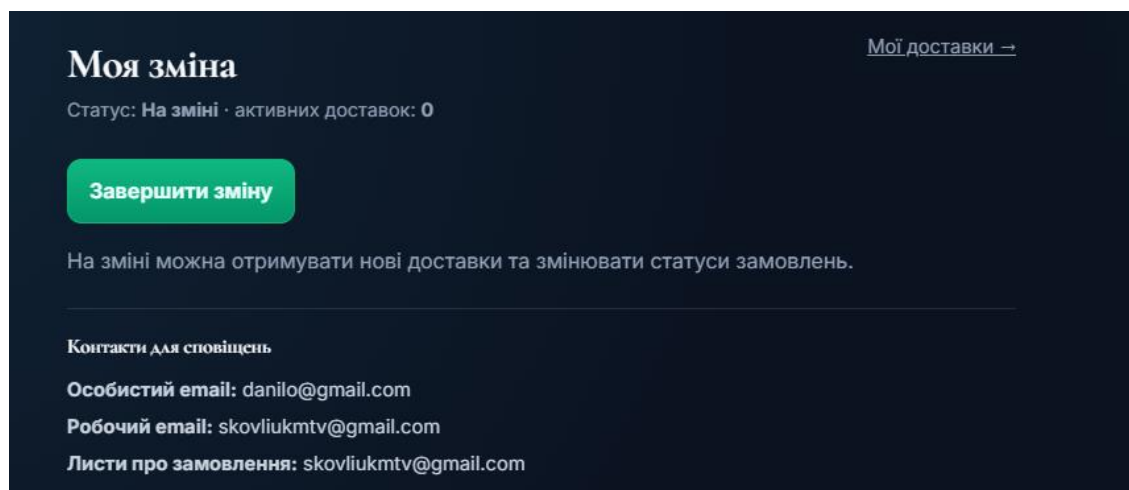


Рис. 3.72 – Сторінка вступу-завершення на зміни зі сторони кур'єра

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						93

Після збирання замовлення на складі воно може бути передане призначеному кур'єру для подальшої доставки. Інформація про отримане замовлення відображається у відповідному розділі кур'єрського інтерфейсу, що показано на рис. 3.73.

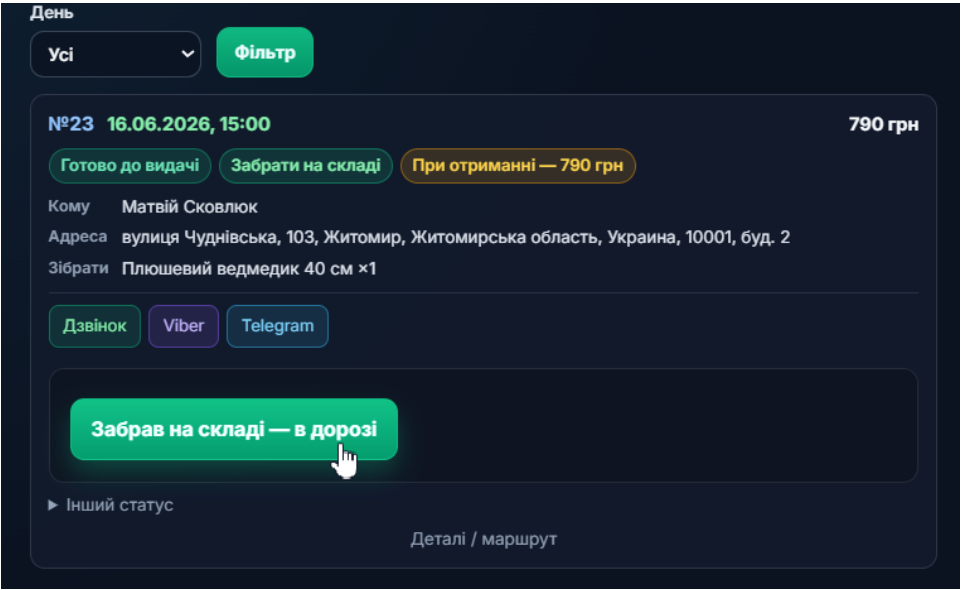


Рис. 3.73 – Замовлення зі складу передане кур'єру

Після отримання замовлення кур'єр може змінювати його поточний статус відповідно до етапу виконання доставки. Зміна статусу фіксується системою та використовується для подальшого інформування клієнта. Приклад зміни статусу наведено на рис. 3.74.

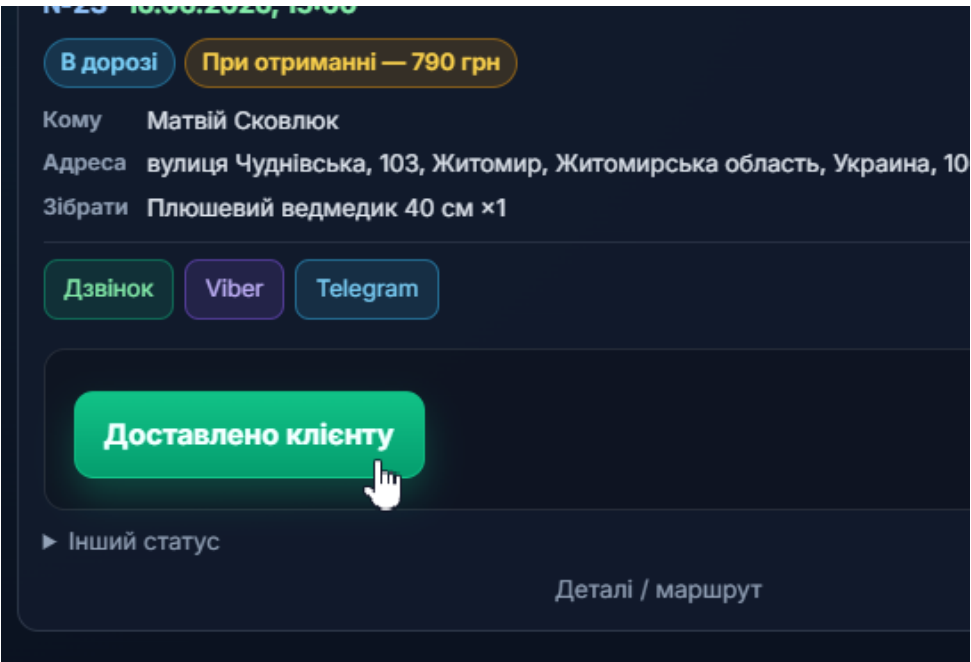


Рис. 3.74 – Зміна статусу кур'єром

Для замовлень, оформлених з оплатою при отриманні, кур'єр також виконує фіксацію факту отримання оплати від клієнта. Відповідний елемент інтерфейсу наведено на рис. 3.75.

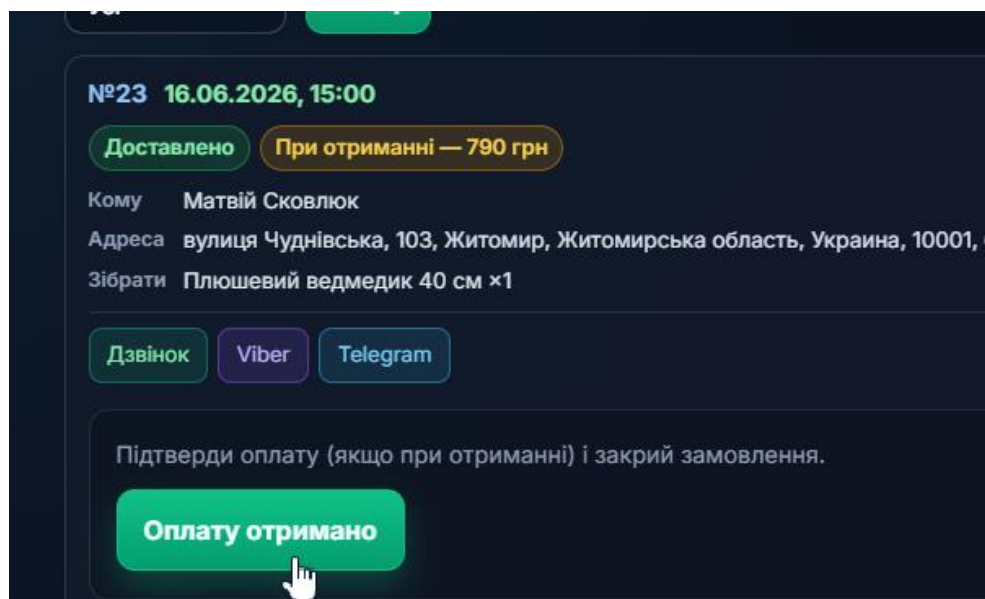


Рис. 3.75 – Отримання оплати якщо оплата не на сайті

Процес відновлення доступу до облікового запису передбачає послідовне виконання декількох етапів. Спочатку користувач переходить до форми відновлення пароля та вказує адресу електронної пошти, пов'язану з обліковим записом. Форма запиту на відновлення пароля наведена на рис. 3.76.

Рис. 3.76 – Форма запиту на відновлення пароля.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				95

Після надсилання запиту на вказану електронну адресу надсилається код підтвердження. Користувач вводить отриманий код у відповідне поле для підтвердження своєї особи та продовження процедури зміни пароля. Форма введення коду підтвердження наведена на рис. 3.77.

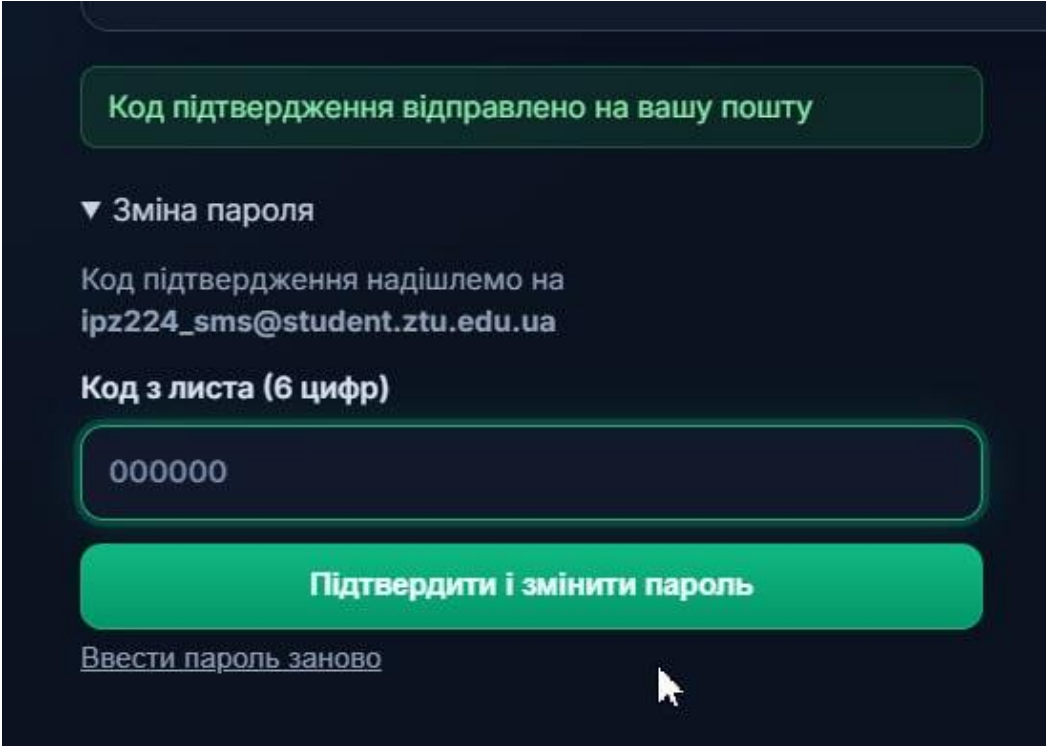


Рис. 3.77 – Форма підтвердження коду відновлення пароля.

Код підтвердження надсилається користувачу електронною поштою та використовується для завершення процедури відновлення доступу. Приклад листа з кодом безпеки наведено на рис. 3.78.

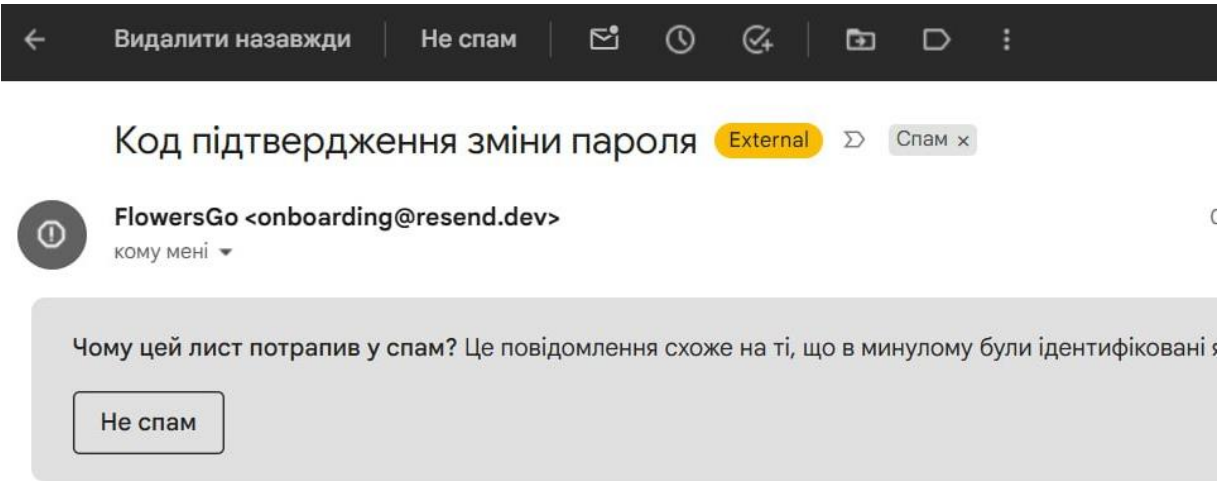


Рис. 3.78 – Лист із кодом підтвердження зміни пароля.

3.3. Тестування вебплатформи

Тестування є завершальним етапом розроблення програмного забезпечення, який дає змогу перевірити відповідність реалізованого функціоналу поставленим вимогам, виявити можливі помилки та оцінити стабільність роботи системи. Для вебплатформи FlowersGo було проведено комплексне тестування, яке охопило перевірку серверної логіки, продуктивності, адаптивності інтерфейсу та основних користувацьких сценаріїв.

Насамперед було перевірено коректність роботи серверного API. Для цього використовувався інструмент Postman [24], який дозволив протестувати REST API, перевірити правильність обробки HTTP-запитів, структуру відповідей сервера, роботу механізмів автентифікації, а також обробку помилок і валідацію вхідних даних. Приклад виконання запиту наведено на рис. 3.79.

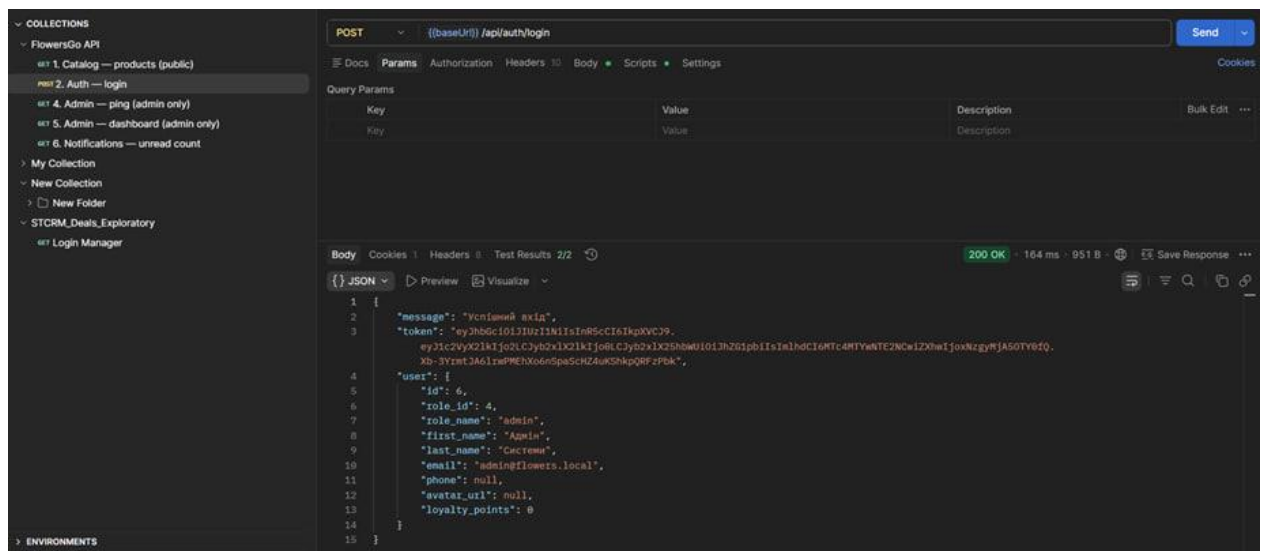


Рис. 3.79 – Приклад запиту для тесту API

Наступним етапом стала оцінка продуктивності вебплатформи. Для цього використано сервіс Google PageSpeed Insights [12], який виконує аналіз швидкодії вебсторінок та формує оцінки за показниками продуктивності, доступності, SEO і відповідності сучасним рекомендаціям. Особливу увагу приділено метрикам Core Web Vitals [13], зокрема Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS) та Interaction to Next Paint (INP), які безпосередньо впливають на швидкість взаємодії користувача із сайтом. Результати виконання тестування наведено на рис. 3.80.

		Сковлюк М.С.			ПТЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				97

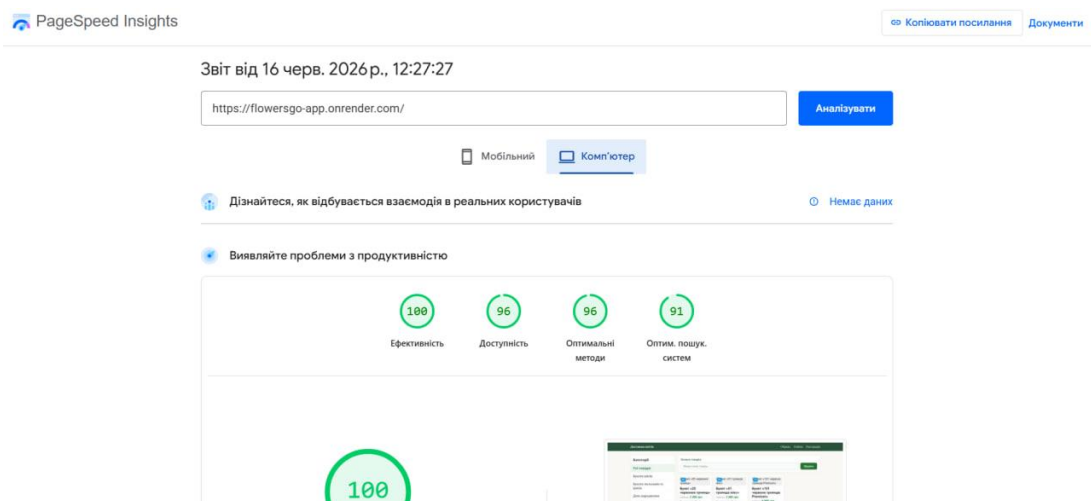


Рис. 3.80 – Результат виконання тестування PageSpeed

Для перевірки коректності відображення інтерфейсу на різних пристроях проведено тестування адаптивності із використанням інструментів розробника сучасних веббраузерів. Перевірку виконано для різних роздільних здатностей екрана, що дозволило оцінити правильність роботи адаптивної верстки на смартфонах, планшетах і персональних комп'ютерах. Під час перевірки також враховувалися рекомендації стандарту WCAG [30], спрямовані на забезпечення доступності вебресурсів для широкого кола користувачів. Приклад відображення головної сторінки на мобільному пристрої наведено на рис. 3.81.

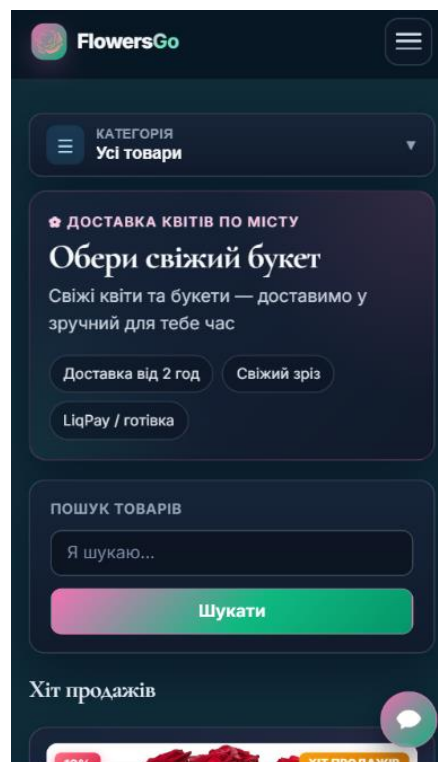


Рис. 3.81 – Відображення головної сторінки на мобільному пристрої

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				98

Після завершення розробки вебсистеми FlowersGo було проведено комплексне тестування її основних компонентів. Перевірено роботу REST API за допомогою інструмента Postman, оцінено швидкість завантаження публічної версії, розгорнутої на платформі Render, із використанням сервісу Google PageSpeed Insights [12] відповідно до метрик Web Vitals [13], а також проаналізовано коректність відображення інтерфейсу на екранах різних розмірів. Додатково виконано автоматизоване модульне тестування серверної логіки контролерів із застосуванням фреймворку Jest [15]. За результатами тестування виявлені окремі недоліки у функціоналі та інтерфейсі, які були усунуті. Це підтвердило відповідність системи вимогам, визначеним на етапі проєктування, зокрема щодо ролей користувачів (клієнт, працівник складу, кур'єр, адміністратор), оформлення замовлень, оплати, доставки, конструктора букетів та адміністративної частини.

Окремо було проаналізовано базові аспекти безпеки вебплатформи. У системі реалізовано автентифікацію на основі JWT, що зберігається в httpOnly-cookie, серверну валідацію вхідних даних, використання параметризованих SQL-запитів до MySQL, а також чітке розмежування доступу до публічних, користувацьких та адміністративних маршрутів. Такий підхід знижує ризики типових вебвразливостей, зокрема несанкціонованого доступу та ін'єкцій.

Загалом застосований підхід до тестування, що поєднує перевірку продуктивності, функціональності та безпеки, забезпечив стабільну роботу системи FlowersGo в умовах хмарного розгортання. Комбінація автоматизованого та ручного тестування дозволила своєчасно усунути виявлені помилки та сформулювати основу для подальшого розвитку платформи, зокрема розширення способів доставки, каталогу товарів і функціоналу без необхідності суттєвих змін архітектури.

Висновки до третього розділу

У третьому розділі виконано розгортання та тестування вебсистеми FlowersGo. Розроблена діаграма розгортання описує взаємодію між браузером

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						99

користувача, серверним застосунком, розміщеним на платформі Render, та базою даних MySQL у хмарному сервісі Aiven. Налаштовано хостинг і підключено зовнішні сервіси, необхідні для роботи інтернет-магазину.

Основний функціонал інтерфейсу користувача проілюстровано графічними зображеннями: каталог і картку товару, реєстрацію та авторизацію, конструктор букетів, оформлення замовлення та оплату, особистий кабінет (замовлення, відгуки, список бажань, сповіщення), а також робочі інтерфейси адміністратора, складу та кур'єра. Окремо показано модулі статистики та управління даними в адміністративній панелі.

Далі проведено ручне тестування серверної частини за допомогою Postman, що дало змогу перевірити коректність REST API (авторизація, каталог, особистий кабінет, адміністративні маршрути) та виявити й усунути окремі помилки в обробці запитів. Клієнтську частину вебсайту проаналізовано за допомогою Google PageSpeed Insights - оцінено швидкість завантаження, доступність, дотримання рекомендацій і SEO для публічної версії.

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						100

ВИСНОВКИ

У процесі виконання випускної кваліфікаційної роботи бакалавра було досліджено та проаналізовано сферу онлайн-продажу квітів і доставки букетів, а також наявні рішення на ринку e-commerce у цій галузі. Головною метою першого розділу було проведення аналізу конкурентів та порівняння їхніх сильних і слабких сторін із запропонованим проєктом «FlowersGo» - платформою для замовлення квітів із доставкою.

У межах виконання завдань вивчено актуальні підходи до організації інтернет-магазинів квітів: каталог товарів, кошик і оформлення замовлення, оплату, відстеження доставки, роботу з кількома ролями користувачів. На підставі цього аналізу сформульовано вимоги до функціональності системи, що розроблялась: підтримка клієнта, складу, кур'єра та адміністратора; конструктор букетів; інтеграцію з платіжною системою; особистий кабінет і адміністративну панель.

У роботі описано архітектурне проєктування системи, ключові сценарії використання та розроблено логічну структуру сховища даних (модель «сутність–зв'язок»). Запропоновані рішення відповідають предметній області інтернет-магазину квітів і створюють основу для програмної реалізації. Визначено мінімальні вимоги до серверного середовища; побудовано набір діаграм (варіантів використання, розгортання, компонентів тощо). Демонстрація інтерфейсів клієнтської та службової частини стала підтвердженням реалізованого функціоналу. Після розробки програмного продукту проведено тестування застосунку (Postman, PageSpeed Insights, перевірка адаптивності) та усунення виявлених недоліків.

Окремо проаналізовано основні алгоритми функціонування системи: обробку замовлень, управління статусами доставки, роботу з кошиком, авторизацію користувачів і розмежування прав доступу. Продемонстровано застосування архітектурного підходу з розмежуванням обов'язків між рівнями застосунку (маршрути, контролери, сервіси, моделі), що спрощує супровід і розширення коду. Реалізація виконана на базі Node.js (Express), засобу

шаблонізації EJS для серверного рендерингу сторінок та MySQL як системи керування базою даних; розгортання здійснено в хмарному середовищі Render з підключенням до БД Aiven.

Ефективність реалізованих функціональних модулів підтверджена під час тестування та практичної експлуатації. Конструктор букетів демонструє швидке перерахування вартості композиції залежно від обраних інгредієнтів завдяки серверному кешуванню даних і мінімізації кількості запитів до бази даних. Система обробки замовлень забезпечує надійну фіксацію транзакцій у середовищі MySQL, що гарантує коректне списання товарних залишків навіть за умови одночасного оформлення покупок кількома користувачами. Інтеграція з платіжним шлюзом LiqPay, авторизація через Google OAuth та механізм сповіщень, побудований на подієвій моделі, забезпечують безперервний супровід замовлення на всіх етапах — від створення до виконання.

Отримані результати підтверджують обґрунтованість обраних технологічних і архітектурних рішень і можуть слугувати базою для подальшого розвитку системи — розширення каталогу, додавання нових способів доставки та інтеграції із зовнішніми сервісами.

Практичне завдання виконано: розроблено вебсистему FlowersGo для замовлення квітів із доставкою, яка забезпечує повний цикл роботи інтернет-магазину від перегляду каталогу до оформлення, оплати та доставки замовлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ліцензійні умови провадження господарської діяльності з роздрібної торгівлі та логістики: матеріали Всеукр. конф. Житомир: ДУ «Житомирська політехніка», 2024. 140 с.
2. LiqPay. Документація API прийому платежів та автоматизації розрахунків. URL: https://www.liqpay.ua/doc/api/information/adding_data (дата звернення: 29.06.2026).
3. Anderson R. Guide to scalable Node.js apps. URL: <https://medium.com/@andersonrose1487/ultimate-guide-to-building-scalable-node-js-apps-with-design-patterns-1d80e34ed125> (дата звернення: 29.06.2026).
4. Aiven for MySQL documentation. URL: <https://aiven.io/docs/products/mysql> (дата звернення: 29.06.2026).
5. Axios Promise based HTTP client URL: <https://axios.rest/pages/advanced/api-reference.html> (дата звернення: 29.06.2026).
6. Bcryptjs Library. URL: <https://www.npmjs.com/package/bcryptjs> (дата звернення: 29.06.2026).
7. Clerk Authentication: identity, metadata extensions, and user management for web applications. URL: <https://clerk.com/docs> (дата звернення: 29.06.2026).
8. Dicentra: мережа квіткових магазинів. URL: <https://dicentra.ua/> (дата звернення: 29.06.2026).
9. Docker Engine Documentation: containerizing server runtime environments and Express application components. URL: <https://docs.docker.com/guides/nodejs/> (дата звернення: 29.06.2026).
10. Embedded JavaScript Templates (EJS): compiler and dynamic syntax specifications for Express server-side rendering. URL: <https://ejs.co/#docs> (дата звернення: 29.06.2026).
11. Express.js documentation for web applications. URL: <https://expressjs.com/en/starter/installing.html> (дата звернення: 29.06.2026).

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						103

12. Flowers.ua: національна логістична служба замовлення та доставки квітів.
URL: <https://flowers.ua/> (дата звернення: 29.06.2026).
13. Google PageSpeed Insights. URL: <https://pagespeed.web.dev/> (дата звернення: 29.06.2026).
14. Web Vitals. URL: <https://web.dev/articles/vitals> (дата звернення: 29.06.2026).
15. Jest Architecture Guide: setting up JavaScript unit testing framework for Express controllers. URL: <https://jestjs.io/docs/getting-started> (дата звернення: 29.06.2026).
16. LoraShen: студія преміального дизайну та флористичного ритейлу. URL: <https://lorashen.ua/> (дата звернення: 29.06.2026).
17. MDN Web Docs. Web Application Security: HTTP authentication cookies, authorization, and transport security controls. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Authentication> (дата звернення: 29.06.2026).
18. Microsoft Docs. Azure Architecture Center. URL: <https://learn.microsoft.com/en-us/azure/architecture/> (дата звернення: 29.06.2026).
19. MySQL 8.0 Reference Manual: database structure modeling, storage engines, transactional integrity, and key indexes. URL: <https://dev.mysql.com/doc/refman/8.0/en/> (дата звернення: 29.06.2026).
20. Node.js API Documentation: comprehensive reference for core modules, classes, and global objects. URL: <https://nodejs.org/docs/latest/api/> (дата звернення: 29.06.2026).
21. OWASP Foundation. Web Application Security Testing Guide. URL: <https://owasp.org/www-project-web-security-testing-guide/> (дата звернення: 29.06.2026).
22. OWASP Top Ten: standard awareness document for developers and web application security. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 29.06.2026).

		Сковлюк М.С.			ПІЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				104

23. PCI Security Standards Council. Payment Card Industry Data Security Standard (PCI DSS) v4.0 architecture rules. URL: <https://www.pcisecuritystandards.org/> (дата звернення: 29.06.2026).
24. Postman docs. URL: <https://learning.postman.com/docs/> (дата звернення: 29.06.2026).
25. Prisma ORM Documentation: data schemas modeling, database clients, type-safe queries, and migrations routing. URL: <https://www.prisma.io/docs> (дата звернення: 29.06.2026).
26. React Documentation. Managing application state and user interface components. URL: <https://react.dev/> (дата звернення: 29.06.2026).
27. Render deployment docs. URL: <https://render.com/docs/> (дата звернення: 29.06.2026).
28. Tailwind CSS Documentation: utility-first CSS framework for responsive UI interface design. URL: <https://tailwindcss.com/docs> (дата звернення: 29.06.2026).
29. TypeScript Documentation. Typed JavaScript handbook for production-scale engineering and static type definitions. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 29.06.2026).
30. Web Content Accessibility Guidelines (WCAG) 2.2. URL: <https://www.w3.org/WAI/standards-guidelines/wcag/> (дата звернення: 29.06.2026).

ДОДАТКИ

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				106

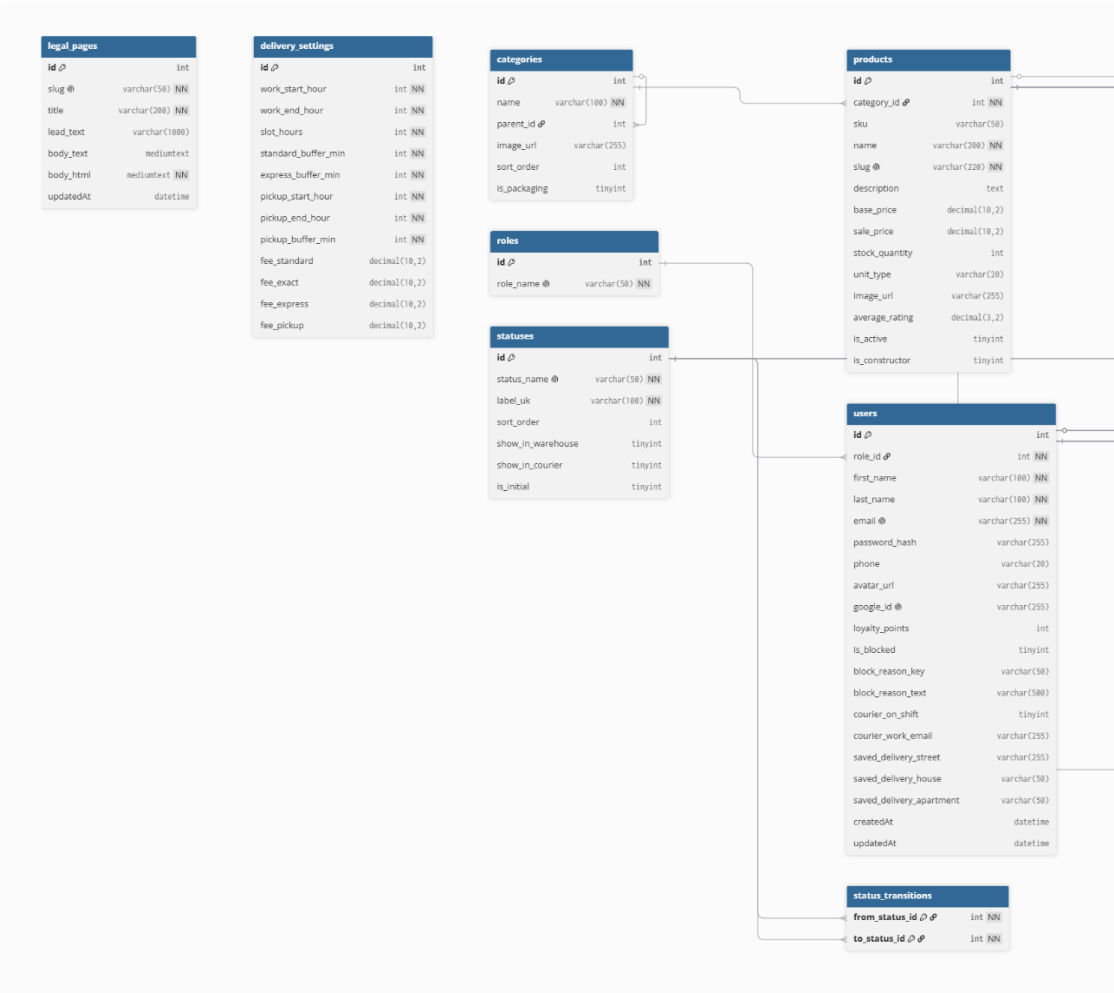


Рис. А.1 - Детальна модель таблиць бази даних (початок)

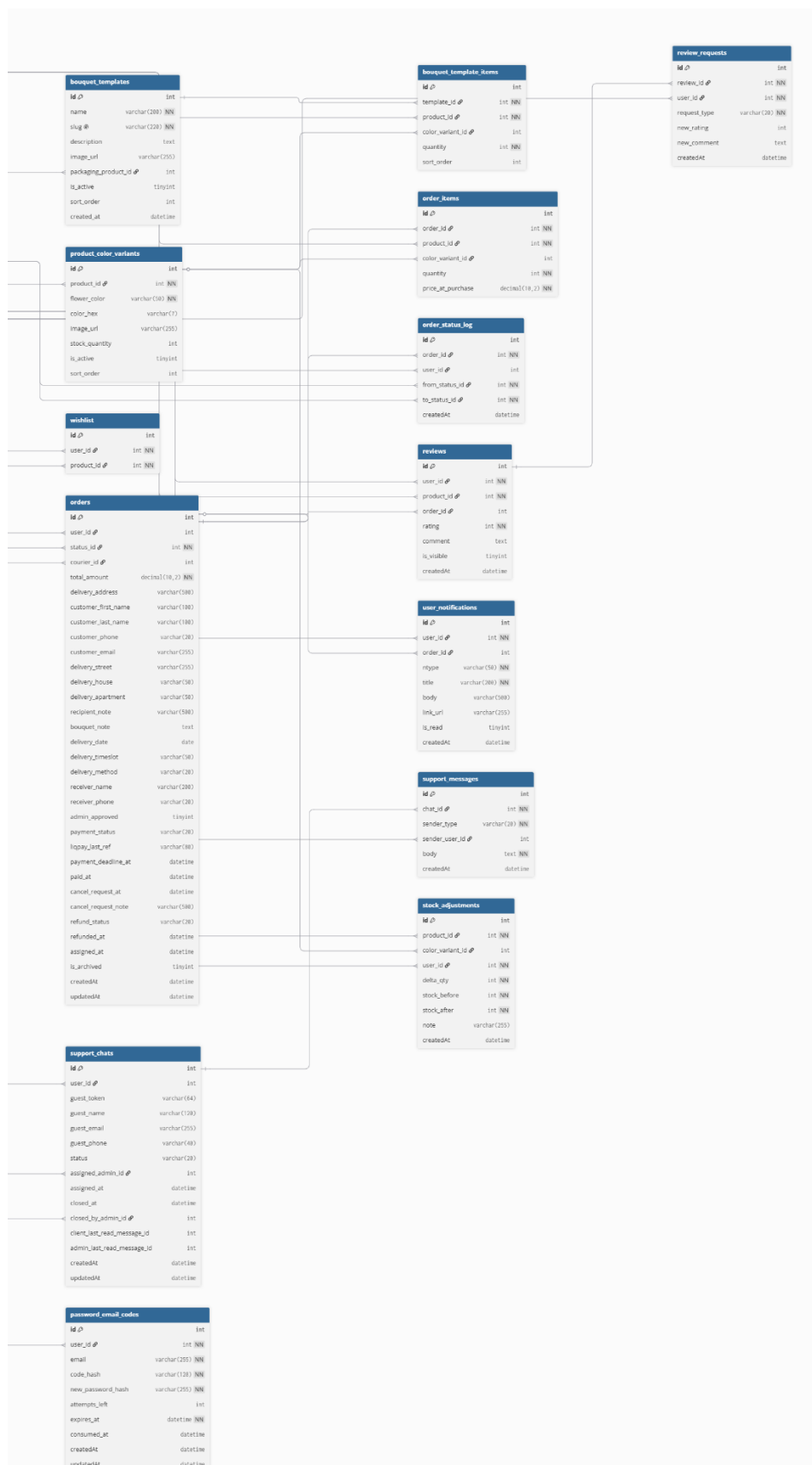


Рис. А.2 - Детальна модель таблиць бази даних (продовження)

Лістинг коду файлу orderPriceServise.js

```

const db = require('../config/db');
const { mergeCartItems } = require('../utils/cartItemMerge');

const roundMoney = (value) => {
  const n = Number(value);
  if (!Number.isFinite(n)) {
    return 0;
  }
  return Math.round(n * 100) / 100;
};

const mergeOrderItems = (rawItems) => {
  return mergeCartItems(rawItems);
};

const resolveItemsPrices = async(rawItems) => {
  const merged = mergeOrderItems(rawItems);
  if (merged.length === 0) {
    return { ok: false, message : 'Немає товарів у замовленні' };
  }

  const resolved = [];

  for (let i = 0; i < merged.length; i += 1) {
    const item = merged[i];
    const product_id = Number(item.product_id);
    const quantity = Number(item.quantity);
    let variantId = item.color_variant_id;

    const [prows] = await db.execute(
      `SELECT id, name, sale_price, is_active
      FROM products
      WHERE id = ?
      LIMIT 1`,
      [product_id]
    );

    if (!prows || !prows[0]) {
      return { ok: false, message : 'Один із товарів більше недоступний' };
    }

    if (Number(prows[0].is_active) !== 1) {
      return { ok: false, message : 'Товар «' +
prows[0].name + '» більше недоступний' };
    }

    if (variantId) {
      const [vrows] = await db.execute(

```

```

        `SELECT id FROM product_color_variants WHERE id =
? AND product_id = ? LIMIT 1`,
        [variantId, product_id]
    );
    if (!vrows || !vrows[0]) {
        return { ok: false, message : 'Один із варіантів
товару більше недоступний' };
    }
}

    const unit_price = roundMoney(prows[0].sale_price || 0);
    if (unit_price < 0) {
        return { ok: false, message : 'Невірна ціна товару «'
+ prows[0].name + '»' };
    }

    resolved.push({
        product_id,
        color_variant_id: variantId,
        quantity,
        unit_price
    });
}

    if (resolved.length === 0) {
        return { ok: false, message : 'Додай хоча б один товар до
замовлення' };
    }

    let itemsTotal = 0;
    for (let j = 0; j < resolved.length; j++) {
        itemsTotal += resolved[j].quantity *
resolved[j].unit_price;
    }

    return {
        ok: true,
        items : resolved,
        itemsTotal : roundMoney(itemsTotal)
    };
};

module.exports = {
    roundMoney,
    resolveItemsPrices
};

```

Лістинг коду файлу deliveryService.js

```

const DEFAULTS = {
    workStartHour: 9,
    workEndHour : 23,
    slotHours : 2,

```

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				110

```

        standardBufferMin : 120,
        expressBufferMin : 60,
        pickupStartHour : 9,
        pickupEndHour : 19,
        pickupBufferMin : 60,
        fees : {
            standard: 0,
            exact : 99,
            express : 99,
            pickup : 0
        }
    };

    const buildConfig = (row) => {
        if (!row) {
            return {
                workStartHour: DEFAULTS.workStartHour,
                workEndHour : DEFAULTS.workEndHour,
                slotHours : DEFAULTS.slotHours,
                standardBufferMin : DEFAULTS.standardBufferMin,
                expressBufferMin : DEFAULTS.expressBufferMin,
                pickupStartHour : DEFAULTS.pickupStartHour,
                pickupEndHour : DEFAULTS.pickupEndHour,
                pickupBufferMin : DEFAULTS.pickupBufferMin,
                fees : {
                    standard: DEFAULTS.fees.standard,
                    exact : DEFAULTS.fees.exact,
                    express : DEFAULTS.fees.express,
                    pickup : DEFAULTS.fees.pickup
                }
            };
        }

        return {
            workStartHour: Number(row.work_start_hour),
            workEndHour : Number(row.work_end_hour),
            slotHours : Number(row.slot_hours),
            standardBufferMin : Number(row.standard_buffer_min),
            expressBufferMin : Number(row.express_buffer_min),
            pickupStartHour : Number(row.pickup_start_hour),
            pickupEndHour : Number(row.pickup_end_hour),
            pickupBufferMin : Number(row.pickup_buffer_min),
            fees : {
                standard: Number(row.fee_standard),
                exact : Number(row.fee_exact),
                express : Number(row.fee_express),
                pickup : Number(row.fee_pickup)
            }
        };
    };

    const getDeliveryFee = (method, config) => {
        const fees = config && config.fees ? config.fees : {};

```

```

        const fee = fees[method];
        if (fee === undefined) {
            return null;
        }
        return fee;
    };

    const pad = (n) => String(n).padStart(2, '0');

    const buildDateTime = (dateStr, timeStr) => {
        if (typeof dateStr !== 'string' || typeof timeStr !==
'string') {
            return null;
        }
        const dateParts = dateStr.split('-');
        const timeParts = timeStr.split(':');
        if (dateParts.length !== 3 || timeParts.length < 2) {
            return null;
        }

        const year = Number(dateParts[0]);
        const month = Number(dateParts[1]);
        const day = Number(dateParts[2]);
        const hour = Number(timeParts[0]);
        const minute = Number(timeParts[1]);

        if (![year, month, day, hour, minute].every(Number.isFinite))
    {
        return null;
    }

        const dt = new Date(year, month - 1, day, hour, minute, 0, 0);
        if (Number.isNaN(dt.getTime())) {
            return null;
        }
        return dt;
    };

    const formatDateTime = (dt) => {
        return `${dt.getFullYear()} - ${ pad(dt.getMonth() + 1) } - ${
pad(dt.getDate()) } ${ pad(dt.getHours()) }:${
pad(dt.getMinutes()) }`;
    };

    const validateSelection = (payload, now, config) => {
        const method = payload && payload.method;
        const fee = getDeliveryFee(method, config);
        if (fee === null) {
            return { ok: false, message : 'Невірний спосіб доставки'
        };
        }

        const nowTime = now instanceof Date ? now : new Date();

```

```

        if (method === 'express') {
            const dt = new Date(nowTime.getTime() +
config.expressBufferMin * 60000);
            if (dt.getHours() >= config.workEndHour) {
                return { ok: false, message : 'Експрес-доставка вже
недоступна на сьогодні' };
            }
            if (dt.getHours() < config.workStartHour) {
                dt.setHours(config.workStartHour, 0, 0, 0);
            }
            return { ok: true, fee, deliveryDatetime :
formatDateTime(dt) };
        }

        const dt = buildDateTime(payload.date, payload.time);
        if (!dt) {
            return { ok: false, message : 'Вкажіть дату й час
доставки' };
        }

        if (method === 'pickup') {
            const hour = dt.getHours();
            if (hour < config.pickupStartHour || hour >=
config.pickupEndHour) {
                return { ok: false, message : `Самовивіз доступний з
${config.pickupStartHour}:00 до ${config.pickupEndHour} : 00` };
            }
            const earliest = new Date(nowTime.getTime() +
config.pickupBufferMin * 60000);
            if (dt.getTime() < earliest.getTime()) {
                return { ok: false, message : 'Оберіть пізніший час -
потрібен час на складання' };
            }
            return { ok: true, fee, deliveryDatetime :
formatDateTime(dt) };
        }

        const hour = dt.getHours();
        if (hour < config.workStartHour || hour >= config.workEndHour)
        {
            return { ok: false, message : `Доставка доступна з
${config.workStartHour}:00 до ${config.workEndHour} : 00` };
        }
        const earliest = new Date(nowTime.getTime() +
config.standardBufferMin * 60000);
        if (dt.getTime() < earliest.getTime()) {
            return { ok: false, message : 'Оберіть пізніший час -
потрібен час на складання й доставку' };
        }

        return { ok: true, fee, deliveryDatetime : formatDateTime(dt)
};

```

```

};

module.exports = {
  buildConfig,
  getDeliveryFee,
  validateSelection
};

```

Лістинг коду файлу orderStockService.js

```

const db = require('../config/db');
const { mergeCartItem } = require('../utils/cartItemMerge');

const pendingWhere = `
o.admin_approved = 0
AND o.cancel_request_at IS NULL
AND(
  o.payment_status IN('paid', 'cod')
  OR(
    o.payment_status = 'unpaid'
    AND(o.payment_deadline_at IS NULL OR o.payment_deadline_at
> NOW())
  )
);

const getPendingQtyForVariant = async(variantId, conn) => {
  const vid = Number(variantId);
  if (!Number.isFinite(vid) || vid <= 0) {
    return 0;
  }

  const runner = conn && typeof conn.execute === 'function' ?
conn : db;

  const [rows] = await runner.execute(
    `SELECT COALESCE(SUM(oi.quantity), 0) AS pending_qty
FROM order_items oi
INNER JOIN orders o ON oi.order_id = o.id
WHERE oi.color_variant_id = ?
AND ${ pendingWhere }`,
    [vid]
  );

  if (!rows || !rows[0]) {
    return 0;
  }
  return Number(rows[0].pending_qty) || 0;
};

const getPendingQtyForProduct = async(productId, conn) => {
  const pid = Number(productId);
  if (!Number.isFinite(pid) || pid <= 0) {
    return 0;
  }

```

```

    const runner = conn && typeof conn.execute === 'function' ?
conn : db;

    const [rows] = await runner.execute(
        `SELECT COALESCE(SUM(oi.quantity), 0) AS pending_qty
        FROM order_items oi
        INNER JOIN orders o ON oi.order_id = o.id
        WHERE oi.product_id = ?
        AND oi.color_variant_id IS NULL
        AND ${ pendingWhere }`,
        [pid]
    );

    if (!rows || !rows[0]) {
        return 0;
    }
    return Number(rows[0].pending_qty) || 0;
};

const validateItemsStock = async(items) => {
    const merged = mergeCartItems(items);
    if (!Array.isArray(merged) || merged.length === 0) {
        return { ok: false, message : 'Немає товарів у замовленні'
    };
    }

    for (let i = 0; i < merged.length; i += 1) {
        const item = merged[i];
        const qty = Number(item.quantity);
        const pid = Number(item.product_id);
        let variantId = null;
        if (item.color_variant_id !== null && item.color_variant_id
!= = '') {
            variantId = Number(item.color_variant_id);
        }

        if (!Number.isFinite(qty) || qty <= 0 ||
!Number.isFinite(pid) || pid <= 0) {
            continue;
        }

        if (variantId && variantId > 0) {
            const [rows] = await db.execute(
                `SELECT v.stock_quantity, p.name AS product_name,
v.flower_color
                FROM product_color_variants v
                INNER JOIN products p ON p.id = v.product_id
                WHERE v.id = ?
                LIMIT 1`,
                [variantId]
            );
            if (!rows || !rows[0]) {

```

```

        return { ok: false, message : 'Один із варіантів
товару більше недоступний' };
    }
    const stock = Number(rows[0].stock_quantity) || 0;
    const pending = await
getPendingQtyForVariant(variantId, null);
    if (stock - pending < qty) {
        let name = rows[0].product_name;
        if (rows[0].flower_color) {
            name = name + ' (' + rows[0].flower_color +
' )';
        }
        return { ok: false, message : 'Недостатньо на
складі: ' + name };
    }
    continue;
}

const [rows] = await db.execute(
    'SELECT stock_quantity, name FROM products WHERE id =
? LIMIT 1',
    [pid]
);
if (!rows || !rows[0]) {
    return { ok: false, message : 'Один із товарів більше
недоступний' };
}
const stock = Number(rows[0].stock_quantity) || 0;
const pending = await getPendingQtyForProduct(pid, null);
if (stock - pending < qty) {
    return { ok: false, message : 'Недостатньо на складі:
' + rows[0].name };
}

return { ok: true };
};

const validateItemsStockInConn = async(conn, items) => {
    if (!conn || !Array.isArray(items) || items.length === 0) {
        return { ok: false, message : 'Немає товарів у замовленні'
};
    }

    for (let i = 0; i < items.length; i++) {
        const item = items[i];
        const qty = Number(item.quantity);
        const pid = Number(item.product_id);
        let variantId = null;
        if (item.color_variant_id !== null && item.color_variant_id
!== '') {
            variantId = Number(item.color_variant_id);
        }
    }
}

```

```

        if (!Number.isFinite(qty) || qty <= 0 ||
!Number.isFinite(pid) || pid <= 0) {
            continue;
        }

        if (variantId && variantId > 0) {
            const [rows] = await conn.execute(
                `SELECT v.stock_quantity, p.name AS product_name,
v.flower_color
FROM product_color_variants v
INNER JOIN products p ON p.id = v.product_id
WHERE v.id = ?
LIMIT 1
FOR UPDATE`,
                [variantId]
            );
            if (!rows || !rows[0]) {
                return { ok: false, message : 'Один із варіантів
товару більше недоступний' };
            }
            const stock = Number(rows[0].stock_quantity) || 0;
            const pending = await
getPendingQtyForVariant(variantId, conn);
            if (stock - pending < qty) {
                let name = rows[0].product_name;
                if (rows[0].flower_color) {
                    name = name + ' (' + rows[0].flower_color +
' )';
                }
                return { ok: false, message : 'Недостатньо на
складі: ' + name };
            }
            continue;
        }

        const [rows] = await conn.execute(
            'SELECT stock_quantity, name FROM products WHERE id =
? LIMIT 1 FOR UPDATE',
            [pid]
        );
        if (!rows || !rows[0]) {
            return { ok: false, message : 'Один із товарів більше
недоступний' };
        }
        const stock = Number(rows[0].stock_quantity) || 0;
        const pending = await getPendingQtyForProduct(pid, conn);
        if (stock - pending < qty) {
            return { ok: false, message : 'Недостатньо на складі:
' + rows[0].name };
        }
    }
}

```

```

        return { ok: true };
    };

    module.exports = {
        validateItemsStock,
        validateItemsStockInConn,
        getPendingQtyForProduct,
        getPendingQtyForVariant
    };

```

Лістинг коду файлу orderStatusService

```

const OrderModel = require('../models/Order');
const StatusModel = require('../models/Status');

const OrderStatusLogModel = require('../models/OrderStatusLog');

const applySystemStatus = async (orderId, toStatusName) => {
    const order = await
    OrderModel.getByIdForStatusChange (orderId);
    if (!order) {
        return false;
    }

    const toId = await OrderModel.getStatusIdByName (toStatusName);
    if (!toId) {
        return false;
    }

    if (Number(order.status_id) === Number(toId)) {
        return true;
    }

    const can = await StatusModel.canTransition (order.status_id,
    toId);
    if (!can) {
        return false;
    }

    const fromId = Number(order.status_id);
    const ok = await OrderModel.updateStatusIfCurrent (orderId,
    fromId, toId);
    if (ok&& fromId !== toId) {
        try {
            await OrderStatusLogModel.insert ({
                order_id: orderId,
                user_id : null,
                from_status_id : fromId,
                to_status_id : toId
            });
        }
        catch (logErr) {
            console.error('system status log:', logErr.message);

```

```

        }
    }
    return ok;
};

const onPaymentConfirmed = async (orderId) => {
    return applySystemStatus (orderId, 'confirmed');
};

const onCodConfirmed = async (orderId) => {
    return applySystemStatus (orderId, 'confirmed');
};

module.exports = {
    applySystemStatus,
    onPaymentConfirmed,
    onCodConfirmed
};

```

Лістинг коду файлу orderStatusService

```

const OrderModel = require('../models/Order');
const StatusModel = require('../models/Status');

const OrderStatusLogModel = require('../models/OrderStatusLog');

const applySystemStatus = async (orderId, toStatusName) => {
    const order = await
    OrderModel.getByIdForStatusChange (orderId);
    if (!order) {
        return false;
    }

    const toId = await OrderModel.getStatusIdByName (toStatusName);
    if (!toId) {
        return false;
    }

    if (Number(order.status_id) === Number(toId)) {
        return true;
    }

    const can = await StatusModel.canTransition (order.status_id,
    toId);
    if (!can) {
        return false;
    }

    const fromId = Number(order.status_id);
    const ok = await OrderModel.updateStatusIfCurrent (orderId,
    fromId, toId);
    if (ok&& fromId !== toId) {
        try {

```

```

        await OrderStatusLogModel.insert({
            order_id: orderId,
            user_id : null,
            from_status_id : fromId,
            to_status_id : toId
        });
    }
    catch (logErr) {
        console.error('system status log:', logErr.message);
    }
}
return ok;
};

const onPaymentConfirmed = async(orderId) => {
    return applySystemStatus(orderId, 'confirmed');
};

const onCodConfirmed = async(orderId) => {
    return applySystemStatus(orderId, 'confirmed');
};

module.exports = {
    applySystemStatus,
    onPaymentConfirmed,
    onCodConfirmed
};

```

Лістинг коду файлу orderController.js

```

const OrderModel = require('../models/Order');
const StatusModel = require('../models/Status');
const OrderStatusLogModel = require('../models/OrderStatusLog');
const orderStatusService =
    require('../services/orderStatusService');
const orderWarehouseNotifyService =
    require('../services/orderWarehouseNotifyService');
const orderRoleNotifyService =
    require('../services/orderRoleNotifyService');
const courierAssignService =
    require('../services/courierAssignService');
const paymentToken = require('../utils/paymentToken');
const deliveryService = require('../services/deliveryService');
const DeliverySettings = require('../models/DeliverySettings');
const phoneValidator = require('../validators/phoneValidator');
const emailValidator = require('../validators/emailValidator');
const constructorService =
    require('../services/constructorService');
const bouquetPreviewService =
    require('../services/bouquetPreviewService');
const orderStockService =
    require('../services/orderStockService');

```

```

const orderPriceService =
require('../services/orderPriceService');
const UserModel = require('../models/User');

const getUserId = (res) => {
    const raw = res.locals.currentUser &&
res.locals.currentUser.user_id;
    const userId = Number(raw);
    if (!Number.isFinite(userId) || userId <= 0) {
        return null;
    }
    return userId;
};

const wantsJson = (req) => {
    const accept = req.headers.accept || '';
    return accept.includes('application/json');
};

const parseItemsFromBody = (raw) => {
    if (raw == null || raw == '') {
        return [];
    }
    if (Array.isArray(raw)) {
        return raw;
    }
    if (typeof raw == 'string') {
        try {
            const parsed = JSON.parse(raw);
            return Array.isArray(parsed) ? parsed : [] ;
        }
        catch (err) {
            return [];
        }
    }
    return [];
};

const buildWarehouseRedirect = (req, body, suffix) => {
    const returnTo = typeof body.return_to == 'string' ?
body.return_to.trim() : '';
    if (returnTo == 'detail') {
        const orderId = Number(req.params.id);
        return '/warehouse/orders/' + orderId + suffix;
    }

    const parts = [];
    const day = typeof body.return_day == 'string' ?
body.return_day.trim() : '';
    const status = typeof body.return_status == 'string' ?
body.return_status.trim() : '';
    const tab = typeof body.return_tab == 'string' ?
body.return_tab.trim() : '';

```

```

    if (day) {
        parts.push('day=' + encodeURIComponent(day));
    }
    if (status) {
        parts.push('status=' + encodeURIComponent(status));
    }
    if (tab) {
        parts.push('tab=' + encodeURIComponent(tab));
    }

    let url = '/warehouse/orders' + suffix;
    if (parts.length > 0) {
        url += (suffix.indexOf('?') === -1 ? '?' : '&') +
parts.join('&');
    }
    return url;
};

const parseDeliveryTimeFromBody = (body) => {
    const exact =
        typeof body.delivery_time_exact === 'string' &&
body.delivery_time_exact.trim() !== ''
        ? body.delivery_time_exact.trim()
        : '';
    if (exact) {
        return exact;
    }

    const slot = typeof body.delivery_slot === 'string' ?
body.delivery_slot.trim() : '';
    if (slot) {
        return slot;
    }

    return typeof body.delivery_time === 'string' ?
body.delivery_time.trim() : '';
};

const createOrder = async(req, res) => {
    try {
        const userId = getUserId(res);

        if (userId) {
            const blocked = await UserModel.isBlocked(userId);
            if (blocked) {
                if (wantsJson(req)) {
                    return res.status(403).json({ message:
'Обліковий запис заблоковано' });
                }
                return res.status(403).send('Обліковий запис
заблоковано');
            }
        }
    }
}

```

```

        const rawItems = parseItemsFromBody(req.body.items);
        const priceResult = await
orderPriceService.resolveItemsPrices(rawItems);
        if (!priceResult.ok) {
            if (wantsJson(req)) {
                return res.status(400).json({ message:
priceResult.message });
            }
            return res.status(400).send(priceResult.message);
        }

        const items = priceResult.items;
        if (items.length == 0) {
            if (wantsJson(req)) {
                return res.status(400).json({ message: 'Додай хоча
б один товар до замовлення' });
            }
            return res.status(400).send('Додай хоча б один товар
до замовлення');
        }

        const first = typeof req.body.customer_first_name == =
'string' ? req.body.customer_first_name.trim() : '';
        const last = typeof req.body.customer_last_name == =
'string' ? req.body.customer_last_name.trim() : '';
        let phone = typeof req.body.customer_phone == = 'string' ?
req.body.customer_phone.trim() : '';
        let emailOpt = typeof req.body.customer_email == =
'string' ? req.body.customer_email.trim() : '';

        if (!first || !last || !phone) {
            if (wantsJson(req)) {
                return res.status(400).json({ message: "Заповни
ім'я, прізвище та телефон" });
            }
            return res.status(400).send("Заповни ім'я, прізвище та
телефон");
        }

        const phoneCheck = phoneValidator(phone);
        if (!phoneCheck.ok) {
            if (wantsJson(req)) {
                return res.status(400).json({ message:
phoneCheck.message });
            }
            return res.status(400).send(phoneCheck.message);
        }
        phone = phoneCheck.phone;

        if (emailOpt) {
            const emailCheck = emailValidator(emailOpt);
            if (!emailCheck.ok) {

```

```

        if (wantsJson(req)) {
            return res.status(400).json({ message:
emailCheck.message });
        }
        return res.status(400).send(emailCheck.message);
    }
    emailOpt = emailCheck.email;
}

const deliverySettingsRow = await DeliverySettings.get();
const deliveryConfig =
deliveryService.buildConfig(deliverySettingsRow);

const delivery_method = req.body.delivery_method;
const deliveryCheck = deliveryService.validateSelection({
    method: delivery_method,
    date : typeof req.body.delivery_date == = 'string' ?
req.body.delivery_date.trim() : '',
    time : parseDeliveryTimeFromBody(req.body)
}, new Date(), deliveryConfig);

if (!deliveryCheck.ok) {
    if (wantsJson(req)) {
        return res.status(400).json({ message:
deliveryCheck.message });
    }
    return res.status(400).send(deliveryCheck.message);
}

const delivery_fee = deliveryCheck.fee;
const delivery_datetime = deliveryCheck.deliveryDatetime;

let street = typeof req.body.delivery_street == = 'string'
? req.body.delivery_street.trim() : '';
let house = typeof req.body.delivery_house == = 'string' ?
req.body.delivery_house.trim() : '';
let apartment = typeof req.body.delivery_apartment == =
'string' ? req.body.delivery_apartment.trim() : '';

if (delivery_method == = 'pickup') {
    street = null;
    house = null;
    apartment = null;
}
else if (!street || !house) {
    if (wantsJson(req)) {
        return res.status(400).json({ message: 'Вкажи
вулицю та будинок' });
    }
    return res.status(400).send('Вкажи вулицю та
будинок');
}

```

```

        const recipient_mode = req.body.recipient_mode ===
'other' ? 'other' : 'self';
        const recFirst =
            typeof req.body.recipient_first_name === 'string' ?
req.body.recipient_first_name.trim() : '';
        const recLast =
            typeof req.body.recipient_last_name === 'string' ?
req.body.recipient_last_name.trim() : '';
        let recPhone =
            typeof req.body.recipient_phone === 'string' ?
req.body.recipient_phone.trim() : '';
        const recNote =
            typeof req.body.recipient_note === 'string' ?
req.body.recipient_note.trim() : '';

        if (recipient_mode === 'other') {
            if (!recFirst || !recLast || !recPhone) {
                if (wantsJson(req)) {
                    return res.status(400).json({ message:
"Заповни ім'я, прізвище та телефон одержувача" });
                }
                return res.status(400).send("Заповни ім'я,
прізвище та телефон одержувача");
            }

            const recPhoneCheck = phoneValidator(recPhone);
            if (!recPhoneCheck.ok) {
                if (wantsJson(req)) {
                    return res.status(400).json({ message:
'Tелефон одержувача: ' + recPhoneCheck.message });
                }
                return res.status(400).send('Телефон одержувача: '
+ recPhoneCheck.message);
            }
            recPhone = recPhoneCheck.phone;
        }

        const bouquetNote =
constructorService.getNoteFromRequest(req);

        let receiver_name = '';
        let receiver_phone = '';
        if (recipient_mode === 'other' && recFirst && recLast &&
recPhone) {
            receiver_name = `${recFirst
} ${recLast}`.replace(/ \s + / g, ' ').trim();
            receiver_phone = recPhone;
        }
        else if (first && last) {
            receiver_name = `${first
} ${last}`.replace(/ \s + / g, ' ').trim();
            receiver_phone = phone;
        }
    }

```

```

const itemsTotal = priceResult.itemsTotal;
const total_price = orderPriceService.roundMoney(itemsTotal +
delivery_fee);

const stockCheck = await
orderStockService.validateItemsStock(items);
if (!stockCheck.ok) {
    if (wantsJson(req)) {
        return res.status(400).json({ message: stockCheck.message
});
    }
    return res.status(400).send(stockCheck.message);
}

const orderId = await OrderModel.createWithTransaction({
    user_id: userId || null,
    customer_first_name : first,
    customer_last_name : last,
    customer_phone : phone,
    customer_email : emailOpt || null,
    delivery_street : delivery_method == 'pickup' ? null :
street,
    delivery_house : delivery_method == 'pickup' ? null : house,
    delivery_apartment : delivery_method == 'pickup' ? null :
apartment || null,
    recipient_note : recipient_mode == 'other' ? recNote || null
: null,
    bouquet_note : bouquetNote || null,
    delivery_method,
    delivery_datetime,
    total_price,
    receiver_name,
    receiver_phone,
    items
});

if (!orderId) {
    if (wantsJson(req)) {
        return res.status(400).json({ message: 'Не вдалося
оформити замовлення' });
    }
    return res.status(400).send('Не вдалося оформити замовлення');
}

const payment_method = req.body.payment_method == 'cod' ? 'cod'
: 'card';
if (payment_method == 'cod') {
    await OrderModel.updatePaymentStatus(orderId, 'cod');
    await orderStatusService.onCodConfirmed(orderId);
    try {
        await orderRoleNotifyService.onNewOrderForAdmin(orderId);
    }
}

```

```

        catch (notifyErr) {
            console.error('onNewOrderForAdmin:', notifyErr.message);
        }
        try {
            await
orderWarehouseNotifyService.notifyCustomerOrderPlaced(
                orderId,
                'Оплата при отриманні. Очікуйте підтвердження
адміністратора.'
            );
        }
        catch (notifyErr) {
            console.error('notifyCustomerOrderPlaced:',
notifyErr.message);
        }
    }
    else {
        await OrderModel.updatePaymentStatus(orderId, 'unpaid');
        try {
            await
orderWarehouseNotifyService.notifyCustomerOrderPlaced(
                orderId,
                'Завершіть оплату протягом 10 хвилин на сторінці
оплати.'
            );
        }
        catch (notifyErr) {
            console.error('notifyCustomerOrderPlaced:',
notifyErr.message);
        }
    }
}

if (String(req.body.checkout_source || '') === 'cart') {
    res.clearCookie('cart_items');
    constructorService.clearNoteCookie(res);
    bouquetPreviewService.clearPreviewCookie(res);
}

const token = paymentToken.makeForOrder(orderId, userId || 0);
const t = encodeURIComponent(token);

let redirectUrl = '/cabinet?ok=order_created';
if (!userId) {
    if (payment_method === 'cod') {
        redirectUrl = '/order/success/' + orderId + '?t=' + t +
'&ok=order_created';
    }
    else {
        redirectUrl = '/payment/' + orderId + '?t=' + t;
    }
}
else if (payment_method !== 'cod') {
    redirectUrl = '/payment/' + orderId + '?t=' + t;
}

```

```

    }
    if (wantsJson(req)) {
        return res.status(200).json({ ok: true, order_id : orderId,
        redirect : redirectUrl });
    }
    return res.redirect(redirectUrl);
    }
    catch (err) {
        console.error('createOrder:', err.message);
        if (wantsJson(req)) {
            return res.status(500).json({ message: 'помилка' });
        }
        return res.status(500).send('помилка');
    }
};

const updateOrderStatusForWarehouse = async(req, res) => {
    try {
        const orderId = Number(req.params.id);
        const statusId = Number(req.body.status_id);
        if (!Number.isFinite(orderId) || orderId <= 0) {
            return res.redirect(buildWarehouseRedirect(req,
            req.body, '?err=bad_order'));
        }
        if (!Number.isFinite(statusId) || statusId <= 0) {
            return res.redirect(buildWarehouseRedirect(req,
            req.body, '?err=bad_status'));
        }

        const allowedStatuses = await
        StatusModel.listForWarehouse();
        const currentOrder = await
        OrderModel.getByIdForWarehouse(orderId);
        if (!currentOrder) {
            return res.redirect(buildWarehouseRedirect(req,
            req.body, '?err=not_found'));
        }
        if (currentOrder.cancel_request_at) {
            return res.redirect(buildWarehouseRedirect(req,
            req.body, '?err=cancel_pending'));
        }

        const allowedStatusIds = allowedStatuses.map((row) =>
        Number(row.id));
        if (!allowedStatusIds.includes(statusId)) {
            return res.redirect(buildWarehouseRedirect(req,
            req.body, '?err=status_unavailable'));
        }

        const transitionOk = await
        StatusModel.canTransition(currentOrder.status_id, statusId);
        if (!transitionOk) {

```

```

        return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=invalid_transition'));
    }

    let targetStatusName = '';
    for (const row of allowedStatuses) {
        if (Number(row.id) === statusId) {
            targetStatusName = row.status_name;
            break;
        }
    }
    if (!targetStatusName) {
        const targetRow = await StatusModel.getById(statusId);
        if (targetRow) {
            targetStatusName = targetRow.status_name;
        }
    }

    if (targetStatusName === 'delivered') {
        if (currentOrder.delivery_method !== 'pickup' ||
currentOrder.status_name !== 'ready_for_pickup') {
            return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=invalid_transition'));
        }
        return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=use_pickup_complete'));
    }

    if (targetStatusName === 'shipped') {
        if (currentOrder.delivery_method === 'pickup') {
            return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=invalid_transition'));
        }
        if (currentOrder.status_name !== 'ready_for_pickup')
        {
            return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=invalid_transition'));
        }
        if (!currentOrder.courier_id) {
            return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=no_courier'));
        }
    }

    const ok = await
OrderModel.updateStatusByWarehouse(orderId, statusId,
Number(currentOrder.status_id));
    if (!ok) {
        return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=update_failed'));
    }

    const oldStatusId = Number(currentOrder.status_id);

```

```

        if (oldStatusId !== statusId) {
            const workerId = getUserId(res);
            try {
                await OrderStatusLogModel.insert({
                    order_id: orderId,
                    user_id: workerId,
                    from_status_id: oldStatusId,
                    to_status_id: statusId
                });
            }
            catch (logErr) {
                console.error('order status log:',
logErr.message);
            }
        }

        let newStatusName = targetStatusName ||
currentOrder.status_name;

        if (newStatusName !== currentOrder.status_name) {
            try {
                await
orderWarehouseNotifyService.notifyCustomerOnStatus(orderId,
newStatusName);
            }
            catch (notifyErr) {
                console.error('notifyCustomerOnStatus:',
notifyErr.message);
            }
        }

        if (newStatusName === 'ready_for_pickup') {
            try {
                await courierAssignService.tryAutoAssign(orderId);
            }
            catch (assignErr) {
                console.error('tryAutoAssign:',
assignErr.message);
            }
        }

        if (newStatusName === 'ready_for_pickup') {
            try {
                const updated = await
OrderModel.getByIdForAssign(orderId);
                if (updated && updated.courier_id) {
                    await
orderRoleNotifyService.onOrderReadyForCourier(orderId,
updated.courier_id);
                }
            }
            catch (readyErr) {

```

```

        console.error('onOrderReadyForCourier:',
readyErr.message);
    }
}

    return res.redirect(buildWarehouseRedirect(req, req.body,
'?ok=1'));
}
    catch (err) {
        console.error('updateOrderStatusForWarehouse:',
err.message);
        return res.redirect(buildWarehouseRedirect(req, req.body,
'?err=server'));
    }
};

const completePickupByWarehouse = async(req, res) => {
    try {
        const orderId = Number(req.params.id);
        if (!Number.isFinite(orderId) || orderId <= 0) {
            return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=bad_order'));
        }

        const confirmCod =
            req.body.confirm_cod == '1' ||
            req.body.confirm_cod == 'on' ||
            req.body.confirm_cod == true;

        const result = await
OrderModel.completePickupByWarehouse(orderId, { confirmCod });
        if (!result.ok) {
            const codeMap = {
                not_found: 'not_found',
                not_ready : 'pickup_not_ready',
                need_cod_confirm : 'pickup_need_cod',
                not_paid : 'pickup_not_paid',
                invalid_transition : 'invalid_transition',
                no_status : 'bad_status'
            };
            const errCode = codeMap[result.code] ||
'update_failed';
            return res.redirect(buildWarehouseRedirect(req,
req.body, '?err=' + errCode));
        }

        const workerId = getUserId(res);
        if (result.from_status_id&& result.to_status_id&&
result.from_status_id !== result.to_status_id) {
            try {
                await OrderStatusLogModel.insert({
                    order_id: orderId,
                    user_id : workerId,

```

```

        from_status_id : result.from_status_id,
        to_status_id : result.to_status_id
    });
    }
    catch (logErr) {
        console.error('pickup complete log:',
logErr.message);
    }
}

    try {
        await
orderWarehouseNotifyService.notifyCustomerOnStatus(orderId,
'accepted');
    }
    catch (notifyErr) {
        console.error('notifyCustomerOnStatus pickup:',
notifyErr.message);
    }

    try {
        await
orderRoleNotifyService.onPickupCompleted(orderId);
    }
    catch (notifyErr) {
        console.error('onPickupCompleted:',
notifyErr.message);
    }

    return res.redirect(buildWarehouseRedirect(req, req.body,
'?ok=pickup_done'));
    }
    catch (err) {
        console.error('completePickupByWarehouse:', err.message);
        return res.redirect(buildWarehouseRedirect(req, req.body,
'?err=server'));
    }
};

module.exports = {
    createOrder,
    updateOrderStatusForWarehouse,
    completePickupByWarehouse
};

```

Лістинг коду файлу Order.js

```

const db = require('../config/db');
const paymentService = require('../services/paymentService');
const orderStockService =
require('../services/orderStockService');
const ProductColorVariant = require('../ProductColorVariant');
const StatusModel = require('../Status');

```

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				132

```

const OrderStatusLog = require('./OrderStatusLog');
const orderDeliveryFields =
require('../utils/orderDeliveryFields');

const ORDER_DELIVERY_STRUCT_SQL = `
o.customer_first_name,
o.customer_last_name,
o.customer_phone,
o.customer_email,
o.delivery_street,
o.delivery_house,
o.delivery_apartment,
o.recipient_note,
o.bouquet_note, `;

const getStatusIdByName = async(name) => {
    const [rows] = await db.execute(
        'SELECT id FROM statuses WHERE status_name = ? LIMIT 1',
        [name]
    );

    if (!rows || rows.length === 0) {
        return null;
    }

    return Number(rows[0].id) || null;
};

const getPendingStatusId = async() => {
    return getStatusIdByName('pending');
};

const normalizeItems = (items) => {
    if (!Array.isArray(items)) {
        return [];
    }

    const cleaned = [];
    for (const item of items) {
        const product_id = Number(item && item.product_id);
        const quantity = Number(item && item.quantity);
        const unit_price = Number(item && item.unit_price);

        if (!Number.isFinite(product_id) || product_id <= 0) {
            continue;
        }
        if (!Number.isFinite(quantity) || quantity <= 0) {
            continue;
        }
        if (!Number.isFinite(unit_price) || unit_price < 0) {
            continue;
        }
    }
}

```

```

        cleaned.push({
            product_id,
            color_variant_id:
                item.color_variant_id != null &&
item.color_variant_id != ''
                ? Number(item.color_variant_id) || null
                : null,
            quantity,
            unit_price
        });
    }

    return cleaned;
};

const createWithTransaction = async(payload) => {
    let user_id = payload && payload.user_id;
    if (user_id != null && user_id != '') {
        user_id = Number(user_id);
        if (!user_id || user_id <= 0) {
            user_id = null;
        }
    }
    else {
        user_id = null;
    }

    const deliveryMeta =
orderDeliveryFields.normalizeOrderDeliveryPayload(payload);
    const customerName =
orderDeliveryFields.customerNameFromRow(deliveryMeta);

    if (!customerName || !deliveryMeta.customer_phone) {
        return null;
    }

    if (deliveryMeta.delivery_method != 'pickup') {
        if (!deliveryMeta.delivery_street ||
!deliveryMeta.delivery_house) {
            return null;
        }
    }

    const items = normalizeItems(payload && payload.items);
    if (items.length == 0) {
        return null;
    }

    const status_id = await getPendingStatusId();
    if (!status_id) {
        return null;
    }
}

```

```

const total_price = Number(payload && payload.total_price);
if (!Number.isFinite(total_price) || total_price < 0) {
    return null;
}

let delivery_date = null;
let delivery_timeslot = null;
const rawDatetime =
    typeof payload.delivery_datetime === 'string' ?
payload.delivery_datetime.trim() : '';
if (rawDatetime !== '') {
    const parts = rawDatetime.split(' ');
    delivery_date = parts[0];
    if (parts[1]) {
        delivery_timeslot = parts[1].slice(0, 5);
    }
}

if (!delivery_date) {
    const today = new Date();
    const yyyy = today.getFullYear();
    const mm = String(today.getMonth() + 1).padStart(2, '0');
    const dd = String(today.getDate()).padStart(2, '0');
    delivery_date = `${yyyy}
} - ${ mm } - ${ dd }`;
}

const receiver_name =
typeof payload.receiver_name === 'string' &&
payload.receiver_name.trim() !== ''
? payload.receiver_name.trim()
: null;
const receiver_phone =
typeof payload.receiver_phone === 'string' &&
payload.receiver_phone.trim() !== ''
? payload.receiver_phone.trim()
: null;

const summaryPayload = {
    ...deliveryMeta,
    receiver_name,
    receiver_phone
};
const delivery_address =
orderDeliveryFields.buildDeliverySummary(summaryPayload);

const conn = await db.getConnection();
try {
    await conn.beginTransaction();

    const stockCheck = await
orderStockService.validateItemsStockInConn(conn, items);

```

```

    if (!stockCheck.ok) {
        await conn.rollback();
        return null;
    }

    const payWindowMin = paymentService.PAYMENT_WINDOW_MINUTES;

    const [orderResult] = await conn.execute(
        `INSERT INTO orders(
            user_id, status_id, delivery_address,
            customer_first_name, customer_last_name,
customer_phone, customer_email,
            delivery_street, delivery_house, delivery_apartment,
            recipient_note, bouquet_note,
            delivery_date, delivery_timeslot, delivery_method,
            total_amount, receiver_name, receiver_phone,
admin_approved, payment_deadline_at
        ) VALUES(? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ?
, ? , ? , ? , ? , ? , 0, DATE_ADD(NOW(), INTERVAL ? MINUTE))`,
        [
            user_id,
            status_id,
            delivery_address,
            deliveryMeta.customer_first_name,
            deliveryMeta.customer_last_name,
            deliveryMeta.customer_phone,
            deliveryMeta.customer_email,
            deliveryMeta.delivery_street,
            deliveryMeta.delivery_house,
            deliveryMeta.delivery_apartment,
            deliveryMeta.recipient_note,
            deliveryMeta.bouquet_note,
            delivery_date,
            delivery_timeslot,
            deliveryMeta.delivery_method,
            total_price,
            receiver_name,
            receiver_phone,
            payWindowMin
        ]
    );

    const orderId = Number(orderResult && orderResult.insertId);
    if (!Number.isFinite(orderId) || orderId <= 0) {
        await conn.rollback();
        return null;
    }

    for (const item of items) {

        const variantId =
            item.color_variant_id != null && item.color_variant_id
!= = ''

```

```

        ? Number(item.color_variant_id)
        : null;
    const [itemResult] = await conn.execute(
        `INSERT INTO order_items(order_id, product_id,
quantity, price_at_purchase, color_variant_id)
VALUES(? , ? , ? , ? , ? )`,
        [orderId, item.product_id, item.quantity,
item.unit_price, variantId]
    );
    if (!itemResult || itemResult.affectedRows <= 0) {
        await conn.rollback();
        return null;
    }
}

    await conn.commit();
    return orderId;
}
catch (err) {
    await conn.rollback();
    throw err;
}
finally {
    conn.release();
}
};

const listByUserId = async(userId) => {
    const uid = Number(userId);
    if (!Number.isFinite(uid) || uid <= 0) {
        return[];
    }

    const [rows] = await db.execute(
        `SELECT o.id,
        o.total_amount AS total_price,
        o.delivery_address, ${ ORDER_DELIVERY_STRUCT_SQL }
        o.delivery_date,
        o.delivery_timeslot,
        o.delivery_method,
        o.admin_approved,
        o.payment_status,
        o.refund_status,
        o.cancel_request_at,
        o.cancel_request_note,
        o.createdAt,
        o.payment_deadline_at,
        COALESCE(o.is_archived, 0) AS is_archived,
        o.status_id, s.status_name AS status_name, s.label_uk AS
status_label
        FROM orders o
        INNER JOIN statuses s ON o.status_id = s.id
        WHERE o.user_id = ?`
    );

```

```

        ORDER BY o.id DESC`,
        [uid]
    );

    return rows;
};

const listForWarehouse = async({ day, status_id } = {}) => {
    const where = [
        'o.admin_approved = 1',
        "o.payment_status IN ('paid', 'cod')",
        "s.status_name IN ('confirmed', 'processing',
        'ready_for_pickup')"
    ];
    const params = [];

    const dayFilter = typeof day === 'string' ? day.trim() : '';
    if (dayFilter === 'today') {
        where.push('DATE(o.delivery_date) = CURDATE()');
    }
    else if (dayFilter === 'tomorrow') {
        where.push('DATE(o.delivery_date) = DATE_ADD(CURDATE(),
        INTERVAL 1 DAY)');
    }

    const sid = Number(status_id);
    if (Number.isFinite(sid) && sid > 0) {
        where.push('o.status_id = ?');
        params.push(sid);
    }

    const [rows] = await db.execute(
        `SELECT o.id, o.user_id,
        o.total_amount AS total_price,
        o.delivery_address, ${ ORDER_DELIVERY_STRUCT_SQL }
        o.delivery_date,
        o.delivery_timeslot,
        o.delivery_method,
        o.payment_status,
        o.receiver_name,
        o.receiver_phone,
        o.cancel_request_at,
        o.cancel_request_note,
        o.courier_id,
        o.status_id, s.status_name AS status_name,
        s.label_uk AS status_label,
        COALESCE(u.email, '') AS customer_email,
        COALESCE(
            NULLIF(TRIM(CONCAT(COALESCE(u.first_name, ''), ' ',
        COALESCE(u.last_name, ''))), ''),
            o.receiver_name,
            'Гість'
        ) AS customer_name,

```

```

        (
            SELECT GROUP_CONCAT(CONCAT(p.name, ' ×', oi.quantity)
SEPARATOR ', ')
            FROM order_items oi
            INNER JOIN products p ON p.id = oi.product_id
            WHERE oi.order_id = o.id
            ) AS products_summary,
        NULLIF(
            TRIM(CONCAT(COALESCE(cu.first_name, ''), ' ',
COALESCE(cu.last_name, ''))),
            ''
        ) AS courier_name
    FROM orders o
    INNER JOIN statuses s ON o.status_id = s.id
    LEFT JOIN users u ON o.user_id = u.id
    LEFT JOIN users cu ON o.courier_id = cu.id
    WHERE ${ where.join(' AND ' ) }
    ORDER BY o.delivery_date ASC, o.delivery_timeslot ASC,
o.id ASC`,
        params
    );

    return rows;
};

const listForWarehouseStats = async() => {
    const [rows] = await db.execute(
        `SELECT o.id,
o.delivery_date,
o.delivery_timeslot,
o.delivery_method,
o.courier_id,
o.cancel_request_at,
s.status_name
FROM orders o
INNER JOIN statuses s ON o.status_id = s.id
WHERE o.admin_approved = 1
AND o.payment_status IN('paid', 'cod')
AND s.status_name IN('confirmed', 'processing',
'ready_for_pickup')
ORDER BY o.id ASC`
    );

    return rows || [];
};

const listForCourier = async({ courier_id, day, status_id } = {})
=> {
    const cid = Number(courier_id);
    if (!Number.isFinite(cid) || cid <= 0) {
        return[];
    }
}

```

```

const where = [
    'o.admin_approved = 1',
    "o.payment_status IN ('paid', 'cod')",
    'o.courier_id = ?',
    "s.status_name IN ('processing', 'ready_for_pickup',
'shipped', 'delivered')"
];
const params = [cid];

const dayFilter = typeof day === 'string' ? day.trim() : '';
if (dayFilter === 'today') {
    where.push('DATE(o.delivery_date) = CURDATE()');
}
else if (dayFilter === 'tomorrow') {
    where.push('DATE(o.delivery_date) = DATE_ADD(CURDATE(),
INTERVAL 1 DAY)');
}

const sid = Number(status_id);
if (Number.isFinite(sid) && sid > 0) {
    where.push('o.status_id = ?');
    params.push(sid);
}

const [rows] = await db.execute(
    `SELECT o.id,
    o.total_amount AS total_price,
    o.delivery_address, ${ ORDER_DELIVERY_STRUCT_SQL }
    o.delivery_date,
    o.delivery_timeslot,
    o.delivery_method,
    o.payment_status,
    o.receiver_name,
    o.receiver_phone,
    o.courier_id,
    o.assigned_at,
    o.status_id,
    s.status_name,
    s.label_uk AS status_label,
    (
        SELECT GROUP_CONCAT(CONCAT(p.name, ' ×', oi.quantity)
SEPARATOR ', ')
        FROM order_items oi
        INNER JOIN products p ON p.id = oi.product_id
        WHERE oi.order_id = o.id
    ) AS products_summary
    FROM orders o
    INNER JOIN statuses s ON o.status_id = s.id
    WHERE ${ where.join(' AND ') }
    ORDER BY o.delivery_date ASC, o.delivery_timeslot ASC,
o.id ASC`,
    params
);

```

```

        return rows || [];
    };

    const listForCourierStats = async(courierId) => {
        const cid = Number(courierId);
        if (!Number.isFinite(cid) || cid <= 0) {
            return [];
        }

        const [rows] = await db.execute(
            `SELECT o.id,
                o.delivery_date,
                o.delivery_timeslot,
                o.delivery_method,
                s.status_name
            FROM orders o
            INNER JOIN statuses s ON o.status_id = s.id
            WHERE o.courier_id = ?
            AND o.admin_approved = 1
            AND s.status_name IN('processing', 'ready_for_pickup',
            'shipped', 'delivered')`,
            [cid]
        );

        return rows || [];
    };

    const getIdForAssign = async(orderId) => {
        const oid = Number(orderId);
        if (!Number.isFinite(oid) || oid <= 0) {
            return null;
        }

        const [rows] = await db.execute(
            `SELECT o.id,
                o.courier_id,
                o.delivery_method,
                o.delivery_date,
                o.delivery_timeslot,
                s.status_name
            FROM orders o
            INNER JOIN statuses s ON o.status_id = s.id
            WHERE o.id = ?
            AND o.admin_approved = 1
            AND o.payment_status IN('paid', 'cod')
            AND o.delivery_method < > 'pickup'
            LIMIT 1`,
            [oid]
        );

        return rows && rows.length > 0 ? rows[0] : null;
    };

```

```

const assignCourier = async(orderId, courierId) => {
    const oid = Number(orderId);
    const cid = Number(courierId);
    if (!Number.isFinite(oid) || oid <= 0 || !Number.isFinite(cid)
|| cid <= 0) {
        return false;
    }

    const [result] = await db.execute(
        `UPDATE orders o
        INNER JOIN statuses s ON o.status_id = s.id
        SET o.courier_id = ? , o.assigned_at = NOW()
        WHERE o.id = ?
        AND o.admin_approved = 1
        AND o.payment_status IN('paid', 'cod')
        AND o.delivery_method < > 'pickup'
        AND o.cancel_request_at IS NULL
        AND s.status_name IN('processing', 'ready_for_pickup',
'shipped')
        AND(o.courier_id IS NULL OR o.courier_id = ? )`,
        [cid, oid, cid]
    );

    return result && result.affectedRows > 0;
};

const unassignCourier = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

    const [result] = await db.execute(
        `UPDATE orders o
        INNER JOIN statuses s ON o.status_id = s.id
        SET o.courier_id = NULL, o.assigned_at = NULL
        WHERE o.id = ?
        AND o.courier_id IS NOT NULL
        AND s.status_name IN('processing', 'ready_for_pickup')`,
        [oid]
    );

    return result && result.affectedRows > 0;
};

const listActiveAssignmentsForDispatch = async() => {
    const [rows] = await db.execute(
        `SELECT o.courier_id,
        o.delivery_date,
        o.delivery_timeslot,
        s.status_name
        FROM orders o

```

```

        INNER JOIN statuses s ON o.status_id = s.id
        WHERE o.courier_id IS NOT NULL
        AND o.admin_approved = 1
        AND s.status_name IN('ready_for_pickup', 'shipped')`
    );

    return rows;
};

const listAwaitingPaymentForAdmin = async() => {
    const [rows] = await db.execute(
        `SELECT o.id,
        o.total_amount AS total_price,
        o.delivery_address,
        o.payment_status,
        o.payment_deadline_at,
        o.createdAt,
        u.first_name,
        u.last_name,
        u.email AS customer_email
        FROM orders o
        LEFT JOIN users u ON o.user_id = u.id
        WHERE o.admin_approved = 0
        AND o.payment_status = 'unpaid'
        AND(o.payment_deadline_at IS NULL OR o.payment_deadline_at
> NOW())
        ORDER BY o.id DESC`
    );

    return rows;
};

const deductStockForOrder = async(conn, orderId) => {
    const [items] = await conn.execute(
        `SELECT product_id, quantity, color_variant_id FROM
order_items WHERE order_id = ?`,
        [orderId]
    );

    for (const row of items || []) {
        const qty = Number(row.quantity);
        const pid = Number(row.product_id);
        const variantId =
            row.color_variant_id != null && row.color_variant_id
!= = ''
            ? Number(row.color_variant_id)
            : null;

        if (!Number.isFinite(qty) || qty <= 0 ||
!Number.isFinite(pid) || pid <= 0) {
            return false;
        }
    }
}

```

```

        if (variantId && Number.isFinite(variantId) && variantId >
0) {
            const ok = await ProductColorVariant.deductStock(conn,
variantId, qty);
            if (!ok) {
                return false;
            }
            continue;
        }

        const [stockResult] = await conn.execute(
            `UPDATE products
            SET stock_quantity = stock_quantity - ?
            WHERE id = ?
            AND stock_quantity >= ? `,
            [qty, pid, qty]
        );
        if (!stockResult || stockResult.affectedRows <= 0) {
            return false;
        }
    }

    return true;
};

const approveForAdmin = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return { ok: false, code : 'bad_id' };
    }

    const conn = await db.getConnection();
    try {
        await conn.beginTransaction();

        const [orderRows] = await conn.execute(
            `SELECT o.id, o.status_id, o.payment_status,
s.status_name
            FROM orders o
            INNER JOIN statuses s ON o.status_id = s.id
            WHERE o.id = ?
            AND o.admin_approved = 0
            AND o.cancel_request_at IS NULL
            LIMIT 1
            FOR UPDATE`,
            [oid]
        );

        const order = orderRows && orderRows[0];
        if (!order) {
            await conn.rollback();
            return { ok: false, code : 'not_found' };
        }
    }

```

```

    }

    const pay = String(order.payment_status || '');
    if (pay !== 'paid' && pay !== 'cod') {
        await conn.rollback();
        return { ok: false, code : 'not_paid' };
    }

    const stockOk = await deductStockForOrder(conn, oid);
    if (!stockOk) {
        await conn.rollback();
        return { ok: false, code : 'no_stock' };
    }

    let nextStatusId = Number(order.status_id);
    if (order.status_name === 'pending') {
        const confirmedId = await
getStatusIdByName('confirmed');
        if (confirmedId) {
            const can = await
StatusModel.canTransition(order.status_id, confirmedId);
            if (can) {
                nextStatusId = confirmedId;
            }
        }
    }

    const [result] = await conn.execute(
        `UPDATE orders
        SET admin_approved = 1,
        status_id = ?
        WHERE id = ? AND admin_approved = 0`,
        [nextStatusId, oid]
    );

    if (!result || result.affectedRows <= 0) {
        await conn.rollback();
        return { ok: false, code : 'not_found' };
    }

    await conn.commit();
    return { ok: true };
}
catch (err) {
    await conn.rollback();
    throw err;
}
finally {
    conn.release();
}
};

const rejectForAdmin = async(orderId) => {

```

```

const oid = Number(orderId);
if (!Number.isFinite(oid) || oid <= 0) {
    return false;
}

const cancelledId = await getStatusIdByName('cancelled');
if (!cancelledId) {
    return false;
}

const [beforeRows] = await db.execute(
    'SELECT status_id FROM orders WHERE id = ? AND
admin_approved = 0 LIMIT 1',
    [oid]
);
if (!beforeRows || !beforeRows[0]) {
    return false;
}

const fromStatusId = Number(beforeRows[0].status_id);

const [result] = await db.execute(
    `UPDATE orders
    SET admin_approved = -1, status_id = ?
    WHERE id = ? AND admin_approved = 0`,
    [cancelledId, oid]
);

if (!result || result.affectedRows <= 0) {
    return false;
}

try {
    await OrderStatusLog.insert({
        order_id: oid,
        user_id : null,
        from_status_id : fromStatusId,
        to_status_id : cancelledId
    });
} catch (err) {
    console.error('rejectForAdmin log:', err.message);
}

return true;
};

const updateStatusById = async(orderId, statusId) => {
    const oid = Number(orderId);
    const sid = Number(statusId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

```

```

    if (!Number.isFinite(sid) || sid <= 0) {
        return false;
    }

    const [result] = await db.execute(
        `UPDATE orders
        SET status_id = ?
        WHERE id = ? `,
        [sid, oid]
    );

    return result && result.affectedRows > 0;
};

const updateStatusIfCurrent = async(orderId, fromStatusId,
toStatusId) => {
    const oid = Number(orderId);
    const fromId = Number(fromStatusId);
    const toId = Number(toStatusId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }
    if (!Number.isFinite(fromId) || fromId <= 0 ||
!Number.isFinite(toId) || toId <= 0) {
        return false;
    }
    if (fromId === toId) {
        return true;
    }

    const [result] = await db.execute(
        `UPDATE orders
        SET status_id = ?
        WHERE id = ? AND status_id = ? `,
        [toId, oid, fromId]
    );

    return result && result.affectedRows > 0;
};

const updateStatusByWarehouse = async(orderId, statusId,
fromStatusId) => {
    if (Number.isFinite(fromStatusId) && fromStatusId > 0) {
        return updateStatusIfCurrent(orderId, fromStatusId,
statusId);
    }
    return updateStatusById(orderId, statusId);
};

const updateStatusByCourier = async(orderId, statusId,
fromStatusId) => {
    if (Number.isFinite(fromStatusId) && fromStatusId > 0) {

```

```

        return updateStatusIfCurrent(orderId, fromStatusId,
statusId);
    }
    return updateStatusById(orderId, statusId);
};

const getByIdForStatusChange = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT id, status_id, payment_status, admin_approved
FROM orders
WHERE id = ?
LIMIT 1`,
        [oid]
    );

    return rows && rows.length > 0 ? rows[0] : null;
};

const belongsToUser = async(orderId, userId) => {
    const oid = Number(orderId);
    const uid = Number(userId);
    if (!Number.isFinite(oid) || oid <= 0 || !Number.isFinite(uid)
|| uid <= 0) {
        return false;
    }

    const [rows] = await db.execute(
        'SELECT id FROM orders WHERE id = ? AND user_id = ? LIMIT
1',
        [oid, uid]
    );

    return rows && rows.length > 0;
};

const getSummaryById = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT o.id, o.user_id, o.total_amount, o.payment_status,
o.delivery_address,
o.receiver_name, o.receiver_phone, o.createdAt,
o.admin_approved
FROM orders o
WHERE o.id = ?

```

```

        LIMIT 1`,
        [oid]
    );

    return rows && rows.length > 0 ? rows[0] : null;
};

const getByIdForPayment = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT id, user_id, total_amount, payment_status,
        createdAt, payment_deadline_at, admin_approved, liqpay_last_ref,
        cancel_request_at
        FROM orders
        WHERE id = ?
        LIMIT 1`,
        [oid]
    );

    =

    return rows && rows.length > 0 ? rows[0] : null;
};

const updateLiqpayLastRef = async(orderId, liqpayRef) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

    const ref = typeof liqpayRef === 'string' ? liqpayRef.trim()
: '';
    if (!ref) {
        return false;
    }

    const [result] = await db.execute(
        `UPDATE orders
        SET liqpay_last_ref = ?
        WHERE id = ? `,
        [ref, oid]
    );

    return result && result.affectedRows > 0;
};

const updatePaymentStatus = async(orderId, paymentStatus) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

```

```

    }

    let sql = 'UPDATE orders SET payment_status = ?';
    const params = [paymentStatus];

    if (paymentStatus === 'paid') {
        sql += ', paid_at = COALESCE(paid_at, NOW())';
    }

    sql += ' WHERE id = ?';
    params.push(oid);

    const [result] = await db.execute(sql, params);

    return result && result.affectedRows > 0;
};

const listForAdminAll = async({ filter, search }) => {
    let sql = `
        SELECT o.id,
            o.total_amount AS total_price,
            o.delivery_address,
            o.delivery_date,
            o.delivery_timeslot,
            o.payment_status,
            o.paid_at,
            o.receiver_name,
            o.receiver_phone,
            o.admin_approved,
            o.cancel_request_at,
            o.cancel_request_note,
            o.refund_status,
            o.createdAt,
            o.delivery_method,
            o.courier_id,
            s.status_name,
            s.label_uk AS status_label,
            u.first_name,
            u.last_name,
            u.email AS customer_email,
            u.phone AS customer_phone,
            NULLIF(
                TRIM(CONCAT(COALESCE(cu.first_name, ''), ' ',
COALESCE(cu.last_name, ''))),
                ''
            ) AS courier_name
        FROM orders o
        INNER JOIN statuses s ON o.status_id = s.id
        LEFT JOIN users u ON o.user_id = u.id
        LEFT JOIN users cu ON o.courier_id = cu.id
    `;

    const params = [];

```

```

const where = [];

const f = typeof filter === 'string' ? filter.trim() : 'all';
if (f === 'pending') {
    where.push('o.admin_approved = 0');
}
else if (f === 'rejected') {
    where.push('o.admin_approved = -1');
}
else if (f === 'warehouse') {
    where.push('o.admin_approved = 1');
    where.push("s.status_name IN ('confirmed', 'processing',
'ready_for_pickup', 'shipped')");
}
else if (f === 'unassigned') {
    where.push('o.admin_approved = 1');
    where.push("s.status_name IN ('processing',
'ready_for_pickup')");
    where.push('o.courier_id IS NULL');
    where.push("o.delivery_method <> 'pickup'");
}
else if (f === 'cancel_requests') {
    where.push('o.cancel_request_at IS NOT NULL');
    where.push("s.status_name NOT IN ('cancelled')");
}
else if (f === 'awaiting_unpaid') {
    where.push('o.admin_approved = 0');
    where.push("o.payment_status = 'unpaid'");
    where.push('(o.payment_deadline_at IS NULL OR
o.payment_deadline_at > NOW())');
}

const q = typeof search === 'string' ? search.trim() : '';
if (q !== '') {
    const num = Number(q);
    if (Number.isFinite(num) && num > 0) {
        where.push('o.id = ?');
        params.push(num);
    }
    else {
        const like = ` ${ q } `;
        where.push(`(
            u.phone LIKE ? OR o.receiver_phone LIKE ? OR
u.email LIKE ?
            OR u.first_name LIKE ? OR u.last_name LIKE ?
            OR o.receiver_name LIKE ? OR
o.delivery_address LIKE ?
        )`);
        params.push(like, like, like, like, like, like, like);
    }
}

if (where.length > 0) {

```

```

        sql += ' WHERE ' + where.join(' AND ');
    }

    sql += ' ORDER BY o.id DESC LIMIT 200';

    const [rows] = await db.execute(sql, params);
    return rows;
};

const countUnassignedCourier = async() => {
    const [rows] = await db.execute(
        `SELECT COUNT(*) AS c FROM orders o
        INNER JOIN statuses s ON o.status_id = s.id
        WHERE o.admin_approved = 1
        AND s.status_name IN('processing', 'ready_for_pickup')
        AND o.courier_id IS NULL
        AND o.delivery_method < > 'pickup'`
    );
    return Number(rows && rows[0] ? rows[0].c : 0) || 0;
};

const getDetailForAdmin = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [orderRows] = await db.execute(
        `SELECT o.id,
        o.total_amount AS total_price,
        o.delivery_address, ${ ORDER_DELIVERY_STRUCT_SQL }
        o.delivery_date,
        o.delivery_timeslot,
        o.payment_status,
        o.paid_at,
        o.payment_deadline_at,
        o.refunded_at,
        o.receiver_name,
        o.receiver_phone,
        o.admin_approved,
        o.cancel_request_at,
        o.cancel_request_note,
        o.refund_status,
        o.liqpay_last_ref,
        o.createdAt,
        o.delivery_method,
        o.courier_id,
        s.status_name,
        s.label_uk AS status_label,
        u.first_name,
        u.last_name,
        u.email AS customer_email,
        u.phone AS customer_phone,`
    );

```

```

        NULLIF(
            TRIM(CONCAT(COALESCE(cu.first_name, ''), ' ',
COALESCE(cu.last_name, ''))),
            ''
        ) AS courier_name
    FROM orders o
    INNER JOIN statuses s ON o.status_id = s.id
    LEFT JOIN users u ON o.user_id = u.id
    LEFT JOIN users cu ON o.courier_id = cu.id
    WHERE o.id = ?
    LIMIT 1`,
    [oid]
);

if (!orderRows || orderRows.length === 0) {
    return null;
}

const [items] = await db.execute(
    `SELECT oi.quantity,
oi.price_at_purchase AS unit_price,
p.name AS product_name,
pcv.flower_color AS color_name
FROM order_items oi
INNER JOIN products p ON p.id = oi.product_id
LEFT JOIN product_color_variants pcv ON pcv.id =
oi.color_variant_id
WHERE oi.order_id = ? `,
    [oid]
);

const status_log = await OrderStatusLog.listById(oid);

return {
    order: orderRows[0],
    items : items || [],
    status_log : status_log || []
};
};

const getByIdForCancelCheck = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT o.id,
o.user_id,
o.total_amount,
o.payment_status,
o.admin_approved,
o.delivery_date,

```

```

        o.delivery_timeslot,
        o.cancel_request_at,
        o.liqpay_last_ref,
        s.status_name
    FROM orders o
    INNER JOIN statuses s ON s.id = o.status_id
    WHERE o.id = ?
    LIMIT 1`,
    [oid]
);

return rows && rows.length > 0 ? rows[0] : null;
};

const createCancelRequestForOrder = async(orderId, note) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

    const text = typeof note === 'string' ? note.trim().slice(0,
500) : '';

    const [result] = await db.execute(
        `UPDATE orders
        SET cancel_request_at = NOW(), cancel_request_note = ?
        WHERE id = ? AND cancel_request_at IS NULL`,
        [text || null, oid]
    );

    return result && result.affectedRows > 0;
};

const getByIdForUserCancel = async(orderId, userId) => {
    const oid = Number(orderId);
    const uid = Number(userId);
    if (!Number.isFinite(oid) || oid <= 0 || !Number.isFinite(uid)
|| uid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT o.id,
        o.user_id,
        o.total_amount,
        o.payment_status,
        o.admin_approved,
        o.delivery_date,
        o.delivery_timeslot,
        o.cancel_request_at,
        o.liqpay_last_ref,
        s.status_name
        FROM orders o

```

```

        INNER JOIN statuses s ON s.id = o.status_id
        WHERE o.id = ? AND o.user_id = ?
        LIMIT 1`,
        [oid, uid]
    );

    return rows && rows.length > 0 ? rows[0] : null;
};

const createCancelRequest = async(orderId, userId, note) => {
    const order = await getByIdForUserCancel(orderId, userId);
    if (!order || order.cancel_request_at) {
        return false;
    }

    const text = typeof note === 'string' ? note.trim().slice(0,
500) : '';

    const [result] = await db.execute(
        `UPDATE orders
        SET cancel_request_at = NOW(), cancel_request_note = ?
        WHERE id = ? AND user_id = ? AND cancel_request_at IS
NULL`,
        [text || null, Number(orderId), Number(userId)]
    );

    return result && result.affectedRows > 0;
};

const getByIdForCancelAdmin = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT o.id,
        o.total_amount,
        o.payment_status,
        o.admin_approved,
        o.liqpay_last_ref,
        o.courier_id,
        o.cancel_request_at,
        o.cancel_request_note,
        s.status_name
        FROM orders o
        INNER JOIN statuses s ON s.id = o.status_id
        WHERE o.id = ?
        LIMIT 1`,
        [oid]
    );

    return rows && rows.length > 0 ? rows[0] : null;
};

```

```

};

const rejectCancelRequest = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

    const [result] = await db.execute(
        `UPDATE orders
        SET cancel_request_at = NULL, cancel_request_note = NULL
        WHERE id = ? AND cancel_request_at IS NOT NULL`,
        [oid]
    );

    return result && result.affectedRows > 0;
};

const restoreStockForOrder = async(conn, orderId) => {
    const [items] = await conn.execute(
        'SELECT product_id, quantity, color_variant_id FROM
order_items WHERE order_id = ?',
        [orderId]
    );

    for (const row of items || []) {
        const qty = Number(row.quantity);
        const pid = Number(row.product_id);
        const variantId =
            row.color_variant_id != null && row.color_variant_id
!= = ''
            ? Number(row.color_variant_id)
            : null;

        if (!Number.isFinite(qty) || qty <= 0) {
            continue;
        }

        if (variantId && Number.isFinite(variantId) && variantId >
0) {
            await ProductColorVariant.restoreStock(conn,
variantId, qty);
            continue;
        }

        if (Number.isFinite(pid) && pid > 0) {
            await conn.execute(
                'UPDATE products SET stock_quantity =
stock_quantity + ? WHERE id = ?',
                [qty, pid]
            );
        }
    }
}

```

```

};

const finishCancelAfterApprove = async(orderId, refundStatus) = >
{
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return false;
    }

    const status = typeof refundStatus == 'string' ?
refundStatus.trim() : 'not_needed';
    const cancelledId = await getStatusIdByName('cancelled');
    if (!cancelledId) {
        return false;
    }

    const conn = await db.getConnection();
    try {
        await conn.beginTransaction();

        const [orderRows] = await conn.execute(
            `SELECT id, status_id, payment_status, admin_approved,
cancel_request_at
            FROM orders
            WHERE id = ?
            LIMIT 1
            FOR UPDATE`,
            [oid]
        );

        const order = orderRows && orderRows[0];
        if (!order || !order.cancel_request_at) {
            await conn.rollback();
            return false;
        }

        const fromStatusId = Number(order.status_id);

        if (Number(order.admin_approved) == 1) {
            await restoreStockForOrder(conn, oid);
        }

        let paymentStatus = order.payment_status;
        let refundedAt = null;

        if (status == 'refunded') {
            paymentStatus = 'refunded';
            refundedAt = new Date();
        }

        let adminApproved = order.admin_approved;
        if (Number(adminApproved) == 0) {
            adminApproved = -1;
        }
    }
}

```

```

    }

    await conn.execute(
      `UPDATE orders
      SET status_id = ? ,
      payment_status = ? ,
      refund_status = ? ,
      refunded_at = ? ,
      admin_approved = ? ,
      courier_id = NULL,
      assigned_at = NULL
      WHERE id = ? ` ,
      [cancelledId, paymentStatus, status, refundedAt,
adminApproved, oid]
    );

    try {
      await OrderStatusLog.insert({
        order_id: oid,
        user_id : null,
        from_status_id : fromStatusId,
        to_status_id : cancelledId
      });
    }
    catch (logErr) {
      console.error('finishCancelAfterApprove log:',
logErr.message);
    }

    await conn.commit();
    return true;
  }
  catch (err) {
    await conn.rollback();
    throw err;
  }
  finally {
    conn.release();
  }
};

const archiveByUserId = async(orderId, userId) = > {
  const oid = Number(orderId);
  const uid = Number(userId);
  if (!Number.isFinite(oid) || oid <= 0 || !Number.isFinite(uid)
|| uid <= 0) {
    return false;
  }

  const [result] = await db.execute(
    `UPDATE orders
    SET is_archived = 1

```

```

        WHERE id = ? AND user_id = ? AND COALESCE(is_archived, 0)
= 0`,
        [oid, uid]
    );

    return result && result.affectedRows > 0;
};

const getByIdForWarehouse = async(orderId) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return null;
    }

    const [rows] = await db.execute(
        `SELECT o.id, o.status_id, o.delivery_method,
o.courier_id, o.payment_status, s.status_name, o.cancel_request_at
FROM orders o
INNER JOIN statuses s ON o.status_id = s.id
WHERE o.id = ?
AND o.admin_approved = 1
AND o.payment_status IN('paid', 'cod')
LIMIT 1`,
        [oid]
    );

    return rows && rows.length > 0 ? rows[0] : null;
};

const completePickupByWarehouse = async(orderId, { confirmCod } =
{}) => {
    const oid = Number(orderId);
    if (!Number.isFinite(oid) || oid <= 0) {
        return { ok: false, code : 'bad_id' };
    }

    const conn = await db.getConnection();
    try {
        await conn.beginTransaction();

        const [orderRows] = await conn.execute(
            `SELECT o.id, o.status_id, o.payment_status,
o.delivery_method, s.status_name
FROM orders o
INNER JOIN statuses s ON o.status_id = s.id
WHERE o.id = ?
AND o.admin_approved = 1
AND o.delivery_method = 'pickup'
AND o.cancel_request_at IS NULL
LIMIT 1
FOR UPDATE`,
            [oid]
        );
    }
};

```

```

const order = orderRows && orderRows[0];
if (!order) {
    await conn.rollback();
    return { ok: false, code : 'not_found' };
}

if (order.status_name !== 'ready_for_pickup' &&
order.status_name !== 'delivered') {
    await conn.rollback();
    return { ok: false, code : 'not_ready' };
}

const pay = String(order.payment_status || '');
if (pay === 'cod') {
    if (!confirmCod) {
        await conn.rollback();
        return { ok: false, code : 'need_cod_confirm' };
    }
    await conn.execute(
        `UPDATE orders SET payment_status = 'paid',
paid_at = COALESCE(paid_at, NOW()) WHERE id = ? `,
        [oid]
    );
}
else if (pay !== 'paid') {
    await conn.rollback();
    return { ok: false, code : 'not_paid' };
}

let currentStatusId = Number(order.status_id);
let currentStatusName = order.status_name;

if (currentStatusName === 'ready_for_pickup') {
    const deliveredId = await
getStatusIdByName('delivered');
    if (!deliveredId) {
        await conn.rollback();
        return { ok: false, code : 'no_status' };
    }
    const canDeliver = await
StatusModel.canTransition(currentStatusId, deliveredId);
    if (!canDeliver) {
        await conn.rollback();
        return { ok: false, code : 'invalid_transition' };
    }
    await conn.execute('UPDATE orders SET status_id = ?
WHERE id = ?', [deliveredId, oid]);
    currentStatusId = deliveredId;
    currentStatusName = 'delivered';
}

const acceptedId = await getStatusIdByName('accepted');

```

```

        if (!acceptedId) {
            await conn.rollback();
            return { ok: false, code : 'no_status' };
        }

        const canAccept = await
        StatusModel.canTransition(currentStatusId, acceptedId);
        if (!canAccept) {
            await conn.rollback();
            return { ok: false, code : 'invalid_transition' };
        }

        await conn.execute('UPDATE orders SET status_id = ? WHERE
id = ?', [acceptedId, oid]);

        await conn.commit();
        return {
            ok: true,
            from_status_id : Number(order.status_id),
            to_status_id : acceptedId,
            via_status_name : currentStatusName
        };
    }
    catch (err) {
        await conn.rollback();
        throw err;
    }
    finally {
        conn.release();
    }
};

const cancelExpiredUnpaidOrders = async() => {
    const cancelledId = await getStatusIdByName('cancelled');
    if (!cancelledId) {
        return { count: 0, ids : [] };
    }

    const [rows] = await db.execute(
        `SELECT o.id, o.status_id
        FROM orders o
        INNER JOIN statuses s ON o.status_id = s.id
        WHERE o.payment_status = 'unpaid'
        AND o.admin_approved = 0
        AND o.payment_deadline_at IS NOT NULL
        AND o.payment_deadline_at <= NOW()
        AND s.status_name = 'pending'
        AND o.cancel_request_at IS NULL`
    );

    if (!rows || rows.length === 0) {
        return { count: 0, ids : [] };
    }
}

```

```

const cancelledIds = [];

for (let i = 0; i < rows.length; i++) {
  const row = rows[i];
  const oid = Number(row.id);
  const fromStatusId = Number(row.status_id);

  const [result] = await db.execute(
    `UPDATE orders
     SET status_id = ? , admin_approved = -1
     WHERE id = ? AND admin_approved = 0 AND payment_status
= 'unpaid'`,
    [cancelledId, oid]
  );

  if (!result || result.affectedRows <= 0) {
    continue;
  }

  cancelledIds.push(oid);

  try {
    await OrderStatusLog.insert({
      order_id: oid,
      user_id : null,
      from_status_id : fromStatusId,
      to_status_id : cancelledId
    });
  }
  catch (logErr) {
    console.error('cancelExpired log:', logErr.message);
  }
}

return { count: cancelledIds.length, ids : cancelledIds };
};

const getDetailForWarehouse = async(orderId) => {
  const oid = Number(orderId);
  if (!Number.isFinite(oid) || oid <= 0) {
    return null;
  }

  const [orderRows] = await db.execute(
    `SELECT o.id,
     o.total_amount AS total_price,
     o.delivery_address,
     o.customer_first_name,
     o.customer_last_name,
     o.customer_phone,
     o.delivery_street,

```

```

        o.delivery_house,
        o.delivery_apartment,
        o.recipient_note,
        o.bouquet_note,
        o.delivery_date,
        o.delivery_timeslot,
        o.delivery_method,
        o.payment_status,
        o.receiver_name,
        o.receiver_phone,
        o.cancel_request_at,
        o.cancel_request_note,
        o.courier_id,
        o.user_id,
        o.status_id,
        s.status_name,
        s.label_uk AS status_label,
        NULLIF(
            TRIM(CONCAT(COALESCE(cu.first_name, ''), ' ',
COALESCE(cu.last_name, ''))),
            ''
        ) AS courier_name,
        COALESCE(NULLIF(TRIM(o.customer_email), ''), u.email, '')
AS customer_email,
        COALESCE(
            NULLIF(TRIM(CONCAT(COALESCE(u.first_name, ''), ' ',
COALESCE(u.last_name, ''))), ''),
            NULLIF(TRIM(CONCAT(COALESCE(o.customer_first_name,
''), ' ', COALESCE(o.customer_last_name, ''))), ''),
            o.receiver_name,
            'Гість'
        ) AS customer_name
FROM orders o
INNER JOIN statuses s ON o.status_id = s.id
LEFT JOIN users u ON o.user_id = u.id
LEFT JOIN users cu ON o.courier_id = cu.id
WHERE o.id = ?
AND o.admin_approved = 1
AND o.payment_status IN('paid', 'cod')
AND s.status_name NOT IN('cancelled')
LIMIT 1`,
[oid]
);

if (!orderRows || orderRows.length === 0) {
    return null;
}

const [items] = await db.execute(
    `SELECT oi.quantity,
oi.price_at_purchase AS unit_price,
oi.color_variant_id,
p.name AS product_name,

```

```

        p.unit_type,
        COALESCE(v.stock_quantity, p.stock_quantity) AS
stock_quantity,
        v.flower_color
    FROM order_items oi
    INNER JOIN products p ON p.id = oi.product_id
    LEFT JOIN product_color_variants v ON v.id =
oi.color_variant_id
    WHERE oi.order_id = ? `,
        [oid]
    );

    return {
        order: orderRows[0],
        items : items || []
    };
};

const findLastDeliveredHighlight = async() => {
    const [fromLog] = await db.execute(
        `SELECT o.id AS order_id,
o.delivery_address,
o.delivery_method,
p.id AS product_id,
p.name AS product_name,
p.slug AS product_slug,
p.image_url,
log_row.delivered_at
FROM(
        SELECT l.order_id, l.createdAt AS delivered_at, l.id
AS log_id
        FROM order_status_log l
        INNER JOIN statuses ts ON l.to_status_id = ts.id AND
ts.status_name = 'delivered'
        ORDER BY l.createdAt DESC, l.id DESC
        LIMIT 1
    ) log_row
    INNER JOIN orders o ON o.id = log_row.order_id
    INNER JOIN statuses os ON o.status_id = os.id AND
os.status_name = 'delivered'
    INNER JOIN order_items oi ON oi.order_id = o.id
    INNER JOIN products p ON p.id = oi.product_id
    WHERE IFNULL(p.is_constructor, 0) = 0
    ORDER BY oi.id ASC
    LIMIT 1`
    );

    if (fromLog && fromLog.length > 0) {
        return fromLog[0];
    }

    const [fallback] = await db.execute(
        `SELECT o.id AS order_id,

```

```

        o.delivery_address,
        o.delivery_method,
        p.id AS product_id,
        p.name AS product_name,
        p.slug AS product_slug,
        p.image_url,
        NULL AS delivered_at
    FROM orders o
    INNER JOIN statuses s ON o.status_id = s.id AND
s.status_name = 'delivered'
    INNER JOIN order_items oi ON oi.order_id = o.id
    INNER JOIN products p ON p.id = oi.product_id
    WHERE IFNULL(p.is_constructor, 0) = 0
    ORDER BY o.id DESC, oi.id ASC
    LIMIT 1`
);

if (!fallback || fallback.length == 0) {
    return null;
}

return fallback[0];
};

module.exports = {
    createWithTransaction,
    normalizeItems,
    getStatusIdByName,
    getPendingStatusId,
    listByUserId,
    listForWarehouse,
    listForWarehouseHistory,
    listForWarehouseStats,
    listForCourier,
    listForCourierHistory,
    listForCourierStats,
    listIdsUnassignedReady,
    listActiveAssignmentsForDispatch,
    listAllForWarehouse,
    listPendingForAdmin,
    listAwaitingPaymentForAdmin,
    approveForAdmin,
    rejectForAdmin,
    updateStatusById,
    updateStatusIfCurrent,
    updateStatusByWarehouse,
    updateStatusByCourier,
    confirmCodPaidByCourier,
    closeByCourier,
    getByIdForAssign,
    assignCourier,
    unassignCourier,
    getByIdForStatusChange,

```

```

        belongsToUser,
        getSummaryById,
        getByIdForPayment,
        updateLiqqpayLastRef,
        updatePaymentStatus,
        archiveByUserId,
        listForAdminAll,
        countUnassignedCourier,
        getDetailForAdmin,
        getByIdForWarehouse,
        getByIdForCourier,
        getDetailForWarehouse,
        getDetailForCourier,
        getByIdForUserCancel,
        getByIdForCancelCheck,
        createCancelRequestForOrder,
        createCancelRequest,
        getByIdForCancelAdmin,
        rejectCancelRequest,
        finishCancelAfterApprove,
        completePickupByWarehouse,
        cancelExpiredUnpaidOrders,
        findLastDeliveredHighlight
    };

```

		Сковлюк М.С.			ПЗ.КР.Б-121-26-ПЗ	Арк.
		Власенко О.В.				
						166