НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри** Компютерних наук
_____**Ігор БОЛБОТ** (підпис)	_____**Белла ГОЛУБ** (підпис)
„____” _____ 2026 р.	„____” _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмний додаток для вирішення задач логістики виробничого підприємства»

Спеціальність 122 – «Комп’ютерні науки»
Освітньо-професійна програма «Комп’ютерні науки»

Гарант освітньої програми

д.е.н., професор _____ Роман РУДЕНСЬКИЙ

Керівник бакалаврської кваліфікаційної роботи (к.т.н., доцент) _____ Віталій СВАТКО

Виконав _____ Олександр СОПРУН

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
к.т.н., доцент _____ Белла ГОЛУБ
„____" _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Сопруну Олександру Юрійовичу

Спеціальність: "122 Комп'ютерні науки"
Освітньо-професійна програма: "122 Комп'ютерні науки"

Тема бакалаврської кваліфікаційної роботи: Програмний додаток для вирішення задач логістики виробничого підприємства

затверджена наказом від «_06_» _січня_2026 р. № __46__ "С"
Термін подання завершеного проекту на кафедру «_22_» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи:
Опис предметної області транспортної логістики виробничого підприємства, вимоги до функціональності системи з боку менеджера, логіста та працівника складу, дані про номенклатуру продукції, склади, клієнтів, замовлення, транспортні засоби й маршрути, а також вимоги до реалізації інформаційної системи у вигляді веб-застосунку з використанням СУБД PostgreSQL та мови програмування Python.
Перелік питань, які потрібно вирішити:
Виконати системний аналіз предметної області та сформулювати вимоги до системи, розробити комплекс моделей предметної області й інформаційної бази, спроєктувати логічну та фізичну моделі реляційної БД і підготувати DDL-скрипти, розробити організаційну структуру прикладного програмного забезпечення та обґрунтувати вибір інструментарію, реалізувати програмні модулі авторизації, управління замовленнями, залишками, маршрутами й звітами, провести модульне й інтеграційне тестування та підготувати рекомендації щодо впровадження системи..
Перелік графічних документів: презентація виконана в PowerPoint_____
Дата видачі завдання: _06.01.2026_.
Керівник бакалаврської кваліфікаційної роботи
___к.т.н., доцент_____ _____ _Віталій СВАТКО

Завдання прийняв до виконання___ _____ Олександр СОПРУН

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8

1.1 Постановка завдання....8

1.2 Огляд інформаційних джерел та існуючих рішень....11

1.3 Моделювання предметної області....14

Висновки до розділу 1....18

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 20

2.1 Логічна модель даних....20

2.2 Вибір системи управління інформаційною базою....27

2.3 Створення інформаційної бази....30

Висновки до розділу 2....41

РОЗДІЛ 3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 43

3.1 Організаційна структура програмного забезпечення....43

3.2 Вибір інструментарію для створення прикладного програмного забезпечення....47

3.3 Алгоритмізація та програмування програмних модулів....52

Висновки до розділу 3....61

РОЗДІЛ 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ....63

4.1 Тестування системи....63

4.2 Вимоги до апаратного та програмного забезпечення....67

4.3 Склад інсталяційного пакету....70

Висновки до розділу 4....73

ВИСНОВКИ 75

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 78

ВСТУП

Сучасні виробничі підприємства функціонують в умовах високої конкуренції, зростаючої варіативності попиту та необхідності оперативного реагування на зміну логістичних умов поставок. Одним із ключових чинників їх ефективної діяльності є належна організація логістичних процесів, зокрема транспортної логістики, що охоплює планування маршрутів доставки продукції, вибір транспортних засобів, формування графіків відвантажень і контроль виконання поставок. Недостатній рівень автоматизації цих процесів призводить до нераціонального використання транспортних ресурсів, збільшення витрат на доставку, затримок у постачанні та зниження рівня сервісу для клієнтів.

На практиці логістичні операції на багатьох підприємствах підтримуються розрізненими засобами обліку — електронними таблицями, локальними обліковими програмами, паперовими журналами, що не забезпечує необхідної інтеграції даних і підтримки прийняття рішень щодо вибору маршрутів доставки. У таких умовах актуальною є розробка спеціалізованих прикладних систем, орієнтованих на підтримку задач транспортної логістики конкретного виробничого підприємства, які дозволяють не тільки вести облік запасів та замовлень, а й здійснювати пошук та оцінювання альтернативних маршрутів доставки за визначеними критеріями.

У даній бакалаврській кваліфікаційній роботі розглядається абстрактне виробниче підприємство, логістичні процеси якого включають управління запасами готової продукції на складах, обробку клієнтських замовлень, планування та реєстрацію відвантажень, а також підтримку прийняття рішень щодо вибору маршрутів доставки із використанням транспортних засобів. Такий узагальнений підхід дає змогу побудувати модель транспортної логістики, яка може бути адаптована до умов реальних підприємств різних галузей.

Об’єктом дослідження є логістична діяльність абстрактного виробничого підприємства, пов’язана з управлінням запасами продукції, обробкою замовлень, організацією відвантажень і транспортною логістикою доставки готової продукції

до споживачів. Предметом дослідження є моделі, методи та програмні засоби автоматизації зазначених логістичних процесів, зокрема задач пошуку маршруту доставки та вибору оптимального маршруту серед можливих варіантів, у складі прикладної інформаційної системи.

Метою роботи є розробка прикладної системи підтримки транспортної логістики абстрактного виробничого підприємства, яка забезпечує облік залишків продукції на складах, обробку клієнтських замовлень, планування та реєстрацію відвантажень, пошук можливих маршрутів доставки, оцінювання їх за заданими критеріями та вибір оптимального маршруту, а також формування інформаційних звітів для прийняття управлінських рішень.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Виконати системний аналіз предметної області логістики абстрактного виробничого підприємства з акцентом на транспортну логістику, визначити основні сутності, процеси та інформаційні потоки, а також сформулювати вимоги до прикладної системи.
2. Розробити інформаційне забезпечення системи, зокрема логічну та фізичну моделі бази даних, що відображають структуру даних щодо продукції, складів, залишків, клієнтів, замовлень, відвантажень, транспортних засобів та маршрутів доставки.
3. Спроєктувати архітектуру прикладної системи, обґрунтувати вибір інструментальних засобів реалізації та розробити програмні модулі, які забезпечують виконання основних логістичних функцій, включно із задачами пошуку та вибору маршруту доставки.
4. Провести тестування розробленої системи, оцінити її працездатність та відповідність вимогам і сформулювати рекомендації щодо впровадження та подальшої експлуатації на виробничому підприємстві.

У роботі передбачається використання методів системного аналізу, структурного й об’єктно-орієнтованого моделювання (у тому числі UML-діаграм), методів проєктування реляційних баз даних, а також сучасних технологій розробки прикладних програмних систем. Для побудови та дослідження моделей задач

транспортної логістики можуть застосовуватися елементи комбінаторної оптимізації та евристичні підходи до маршрутизації.

Структура записки зумовлена поставленою метою та завданнями і відповідає вимогам методичних вказівок. Пояснювальна записка складається зі вступу, чотирьох розділів основної частини, висновків, списку використаних джерел та додатків. У першому розділі виконано системний аналіз предметної області та сформульовано вимоги до прикладної системи. У другому розділі розроблено інформаційне забезпечення системи, зокрема логічну та фізичну моделі бази даних. Третій розділ присвячено архітектурі та програмній реалізації прикладної системи, включно з алгоритмами задач транспортної логістики. У четвертому розділі наведено результати тестування системи, визначено вимоги до середовища її функціонування та подано рекомендації щодо впровадження й експлуатації. Додатки містять графічні матеріали, фрагменти програмного коду та інші допоміжні документи.

РОЗДІЛ 1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Логістична діяльність абстрактного виробничого підприємства являє собою сукупність взаємопов'язаних процесів управління матеріальними та інформаційними потоками від моменту виготовлення продукції до її доставки кінцевому споживачу. До складу цих процесів належать: управління запасами готової продукції на складах підприємства; приймання, обробка та формування клієнтських замовлень; планування і реєстрація відвантажень з призначенням транспортних засобів; а також підтримка прийняття рішень у галузі транспортної логістики, зокрема щодо пошуку можливих маршрутів доставки, їх оцінювання за визначеними критеріями та вибору оптимального варіанта.

На рис. 1 наведено структурну схему основних логістичних процесів абстрактного виробничого підприємства.

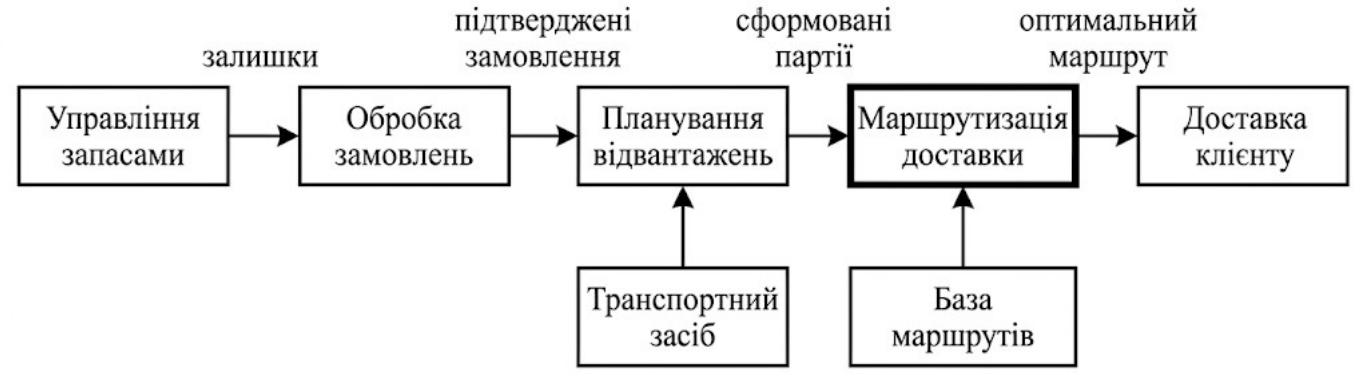

Рис. 1 Структурна схема основних логістичних процесів виробничого підприємства

Аналіз рисунка 1 свідчить про те, що логістичні процеси підприємства утворюють послідовний ланцюжок від управління запасами до доставки продукції клієнту. Ключовою ланкою, яка визначає ефективність усього ланцюга, є блок маршрутизації доставки — він взаємодіє як з базою маршрутів, так і з блоком планування відвантажень через дані про транспортні засоби. Саме цей елемент є предметом підвищеної уваги у даній роботі.

На практиці зазначені процеси на більшості виробничих підприємств реалізуються із застосуванням неоднорідних засобів обліку: дані про залишки продукції зберігаються в електронних таблицях, замовлення фіксуються у поштових клієнтах або в локальних облікових програмах, а планування маршрутів доставки здійснюється вручну на основі досвіду логіста без формалізованих алгоритмів та актуальних даних про завантаження транспортних засобів. Це породжує низку суттєвих проблем, що безпосередньо знижують ефективність роботи підприємства та якість обслуговування клієнтів.

Систематизацію основних логістичних процесів, їх поточного стану та проблем, що виникають, наведено в таблиці 1.1.

Таблиця 1.1

Основні логістичні процеси та проблеми їх реалізаці

Процес	Поточний стан	Проблеми
Управління запасами та замовленнями	Ведення окремих файлів MS Excel, паперових журналів для різних складів	Дублювання даних, неузгодженість залишків, відсутність актуальної зведеної інформації
Планування відвантажень	Ручне складання графіків відвантажень з урахуванням наявного транспорту	Відсутність інтеграції між замовленнями і транспортними даними, конфлікти в розкладі
Маршрутизація доставки	Вибір маршруту здійснюється на основі суб'єктивного досвіду логіста без формального порівняння альтернатив	Нераціональне використання транспорту, неможливість обґрунтувати та зафіксувати вибір маршруту

Аналіз даних таблиці 1.1 показує, що найбільш критичною є проблема відсутності автоматизованої підтримки задач маршрутизації доставки: вибір маршруту повністю залежить від компетентності окремого спеціаліста, не документується в єдиній системі і не дає можливості проводити порівняльний аналіз альтернативних варіантів. Це безпосередньо обумовлює необхідність розробки прикладної системи, яка б усувала зазначені недоліки.

Виходячи з аналізу проблемної ситуації, у межах даної роботи формулюється задача створення єдиної прикладної інформаційної системи, яка централізовано зберігає та обробляє дані щодо продукції, складів, клієнтів, замовлень, відвантажень, транспортних засобів і маршрутів доставки. З позицій класифікації інформаційних систем розроблювана система належить до класу інформаційно-

довідкових та облікових систем логістичного спрямування з елементами системи підтримки прийняття рішень у транспортній логістиці.[1]

Для формалізованого опису задачі вибору маршруту доставки введемо позначення. Нехай $P=\{p_1,p_2,\ldots,p_n\}$ — множина пунктів доставки (клієнтів), що обслуговуються в межах одного відвантаження. Для кожної пари пунктів (p_i,p_j) визначена відстань d_{ij} або орієнтовний час переміщення t_{ij}. Задача маршрутизації полягає у знаходженні такої послідовності обходу пунктів $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$, що починається і закінчується на базовому складі s_0 та мінімізує цільову функцію:

$$F(\pi)=d_{s_0,\pi_1}+\sum_{k=1}^{n-1} d_{\pi_k,\pi_{k+1}}+d_{\pi_n,s_0}\to min \qquad (1)$$

де $F(\pi)$ — загальна довжина маршруту. Залежно від пріоритетів підприємства критерієм оптимізації може виступати як мінімальна відстань, так і мінімальний сумарний час доставки або мінімальна вартість транспортування.

Основними інформаційними об'єктами системи є: «Продукція», «Склад», «Залишок», «Клієнт», «Замовлення», «Позиція замовлення», «Відвантаження», «Транспортний засіб», «Маршрут». Перелік операцій, що підлягають автоматизації, охоплює: реєстрацію замовлень і формування їх складу; резервування продукції та списання залишків зі складу; призначення транспортного засобу для відвантаження; пошук та оцінювання можливих маршрутів доставки за визначеними критеріями; вибір і фіксацію оптимального маршруту; реєстрацію фактів виконання доставки.

Результатами функціонування прикладної системи є не лише актуальна база даних, а й звітні документи таких типів: відомість залишків продукції на складах; реєстр поточних і виконаних замовлень; графік відвантажень із зазначенням транспортних засобів; зведена відомість маршрутів доставки з основними характеристиками (відстань, орієнтовний час, кількість пунктів доставки).[2]

Таким чином, постановка завдання визначає межі предметної області, склад інформаційних об'єктів, перелік операцій автоматизації, набір звітних документів і формальну постановку задачі маршрутизації. Отримані вимоги слугують

безпосередньою основою для побудови моделей предметної області у підрозділі 1.3 та розробки інформаційного забезпечення системи у розділі 2.

1.2 Огляд інформаційних джерел та існуючих рішень

Дослідження предметної області транспортної логістики та автоматизованих систем її підтримки передбачає аналіз як наукових публікацій і навчальних видань, так і існуючих програмних продуктів, що реалізують функції управління запасами, обробки замовлень, планування відвантажень і маршрутизації доставки.

У галузі теорії транспортної логістики фундаментальним є поняття задачі комівояжера (Travelling Salesman Problem, TSP), що формалізує задачу побудови оптимального маршруту обходу заданої множини пунктів з поверненням до початкової точки. Задача TSP належить до класу NP-важких комбінаторних задач оптимізації: точний алгоритм її розв'язання за поліноміальний час невідомий, тому на практиці широко застосовуються евристичні та метаевристичні підходи — жадібний алгоритм (nearest neighbour heuristic), алгоритм 2-opt, мурашині алгоритми, генетичні алгоритми тощо. Узагальненням TSP є задача маршрутизації транспортних засобів (Vehicle Routing Problem, VRP), яка враховує обмеження вантажопідйомності, часові вікна обслуговування та кількість доступних транспортних засобів.[3]

Серед навчальних та наукових видань з логістики і транспортного менеджменту слід відзначити роботи у сфері проєктування інформаційних систем підтримки логістичних процесів, де окремо розглядаються архітектурні підходи до побудови систем управління складом (WMS — Warehouse Management System), систем управління транспортом (TMS — Transportation Management System) та інтегрованих рішень класу ERP. Перелічені класи систем формують контекст, у якому розміщується розроблювана у даній роботі прикладна система.[4]

Серед програмних продуктів, що реалізують функції транспортної логістики на виробничому підприємстві, найбільш поширеними є такі рішення: SAP Transportation Management, Oracle Transportation Management, 1С:Управление транспортом, а також спеціалізовані TMS-рішення класу SMB (для малого та

середнього бізнесу) — наприклад, Route4Me, OptimoRoute, Onfleet. Для порівняльного аналізу зазначених рішень та їх відповідності потребам абстрактного виробничого підприємства в таблиці 1.2 наведено зведену характеристику основних аналогів за ключовими критеріями.

Таблиця 1.2

Порівняльний аналіз існуючих програмних рішень у галузі транспортної логістики

Назва системи	Управління запасами	Обробка замовлень	Маршрутизація	Масштаб	Вартість впровадження
SAP TM	Так	Так	Розширена (VRP, TW)	Великі підприємства	Висока
Oracle TM	Так	Так	Розширена (VRP, TW)	Великі підприємства	Висока
1С: Управління транспортом	Частково	Так	Базова	Середні підприємства	Середня
Route4Me	Ні	Ні	Так (оптимізація маршрутів)	Малі/середні	Середня (SaaS)
OptimoRoute	Ні	Ні	Так (TSP, VRP)	Малі/середні	Середня (SaaS)
Розроблювана система	Так	Так	Базова (TSP-евристика)	Абстрактне підприємство	Мінімальна

Аналіз таблиці 1.2 дозволяє зробити такі висновки. По-перше, потужні комерційні системи (SAP TM, Oracle TM) забезпечують повний функціональний охват задач транспортної логістики, однак їх впровадження є економічно недоцільним для малих і середніх підприємств через надмірну складність і високу вартість. По-друге, спеціалізовані SaaS-рішення (Route4Me, OptimoRoute) забезпечують якісну маршрутизацію, але не інтегровані з обліком запасів та обробкою замовлень, що унеможливлює формування цілісної картини логістичної діяльності підприємства. По-третє, система 1С надає частковий функціонал, однак орієнтована на специфіку пострадянського облікового середовища і потребує значного налаштування. Таким чином, жодне з наявних рішень не забезпечує оптимального поєднання обліку запасів, обробки замовлень і базової

маршрутизації в єдиній прикладній системі за мінімальних витрат на впровадження.

На рис. 2 наведено порівняльну діаграму функціональних можливостей розглянутих систем.

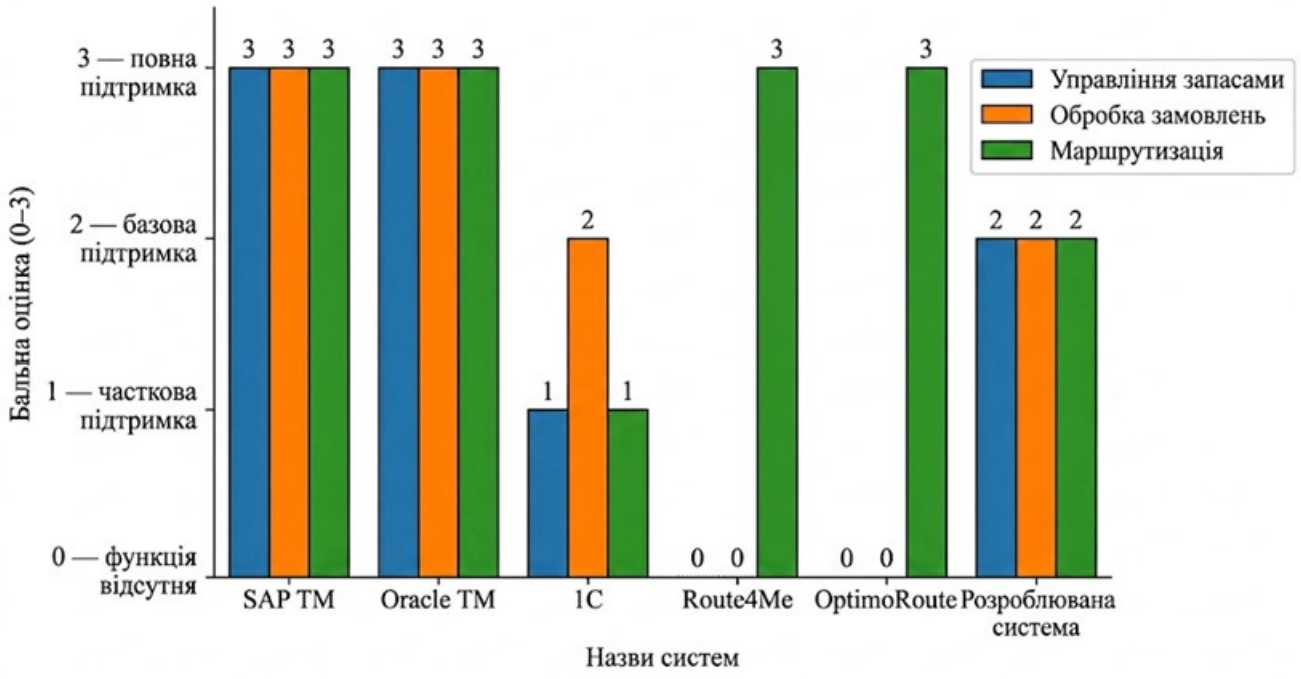

Рис. 2. – Порівняльна діаграма функціональних можливостей систем транспортної логістики

Аналіз рисунка 2 підтверджує, що розроблювана система займає збалансовану позицію: на відміну від вузькоспеціалізованих рішень (Route4Me, OptimoRoute), вона охоплює всі три функціональні напрями на базовому рівні, що цілком достатньо для потреб абстрактного виробничого підприємства. На відміну від великих корпоративних систем, вона не потребує значних ресурсів для впровадження і може бути адаптована до конкретних умов діяльності підприємства.

Огляд інформаційних джерел дозволяє також визначити методологічну базу для вирішення задачі маршрутизації в рамках даної роботи. З огляду на обмежений масштаб задачі (кількість пунктів доставки $n \leq 15–20$ у межах одного відвантаження), застосування точних методів (метод гілок і меж, динамічне програмування) є обґрунтованим для малих екземплярів задачі, тоді як жадібний

алгоритм найближчого сусіда є достатнім евристичним підходом для оперативного формування маршруту в реальних умовах роботи системи.[4,5]

Таким чином, огляд інформаційних джерел та існуючих рішень підтверджує актуальність розробки спеціалізованої прикладної системи підтримки транспортної логістики для абстрактного виробничого підприємства, визначає теоретичну базу для алгоритмізації задачі маршрутизації та обґрунтовує вибір підходів до її розв'язання в розділі 3 даної роботи.

1.3 Моделювання предметної області

Моделювання предметної області є обов'язковим етапом системного аналізу, що дозволяє формалізувати знання про об'єкт автоматизації, визначити структуру його основних елементів, функціональні зв'язки між ними та сценарії взаємодії користувачів із системою. Відповідно до методичних вказівок, комплекс моделей предметної області подається у вигляді функціональних, контекстних діаграм та діаграм UML. У даному підрозділі розроблено такі моделі: контекстна діаграма системи, діаграма варіантів використання (use case), діаграма діяльності для процесу вибору маршруту доставки.[6]

Контекстна діаграма визначає межі системи, що розробляється, і описує її взаємодію із зовнішнім оточенням — акторами та суміжними інформаційними об'єктами. Зовнішніми акторами прикладної системи підтримки транспортної логістики є: менеджер з продажу (реєструє замовлення, переглядає стан їх виконання); працівник складу (фіксує залишки продукції, підтверджує відвантаження); логіст (планує відвантаження, обирає маршрут доставки, призначає транспортний засіб); адміністратор (керує довідниками системи). На рис. 3 наведено контекстну діаграму прикладної системи.

Рис. 3 Контекстна діаграма прикладної системи підтримки транспортної логістики

Аналіз рисунка 3 показує, що система взаємодіє з чотирма категоріями внутрішніх користувачів та двома зовнішніми інформаційними об'єктами. Центральне місце у взаємодії з системою займає логіст — саме він є основним споживачем функцій транспортної логістики та ініціює процедуру вибору маршруту доставки.

Діаграма варіантів використання (use case). Діаграма варіантів використання описує функціональні можливості системи з точки зору її користувачів і визначає перелік дій, які кожен актор може виконувати в системі. У таблиці 1.3 наведено перелік варіантів використання системи в розрізі акторів.

Таблиця 1.3

Варіанти використання прикладної системи в розрізі акторів

Актор	Варіант використання
Менеджер з продажу	Реєстрація нового замовлення; формування складу замовлення; перегляд стану виконання замовлень; формування реєстру замовлень
Працівник складу	Перегляд і коригування залишків продукції; підтвердження відвантаження; формування відомості залишків
Логіст	Планування відвантаження; призначення транспортного засобу; пошук маршрутів доставки; оцінювання маршрутів за критеріями; вибір

	оптимального маршруту; фіксація виконання доставки; формування графіка відвантажень
Адміністратор	Ведення довідників (продукція, клієнти, склади, транспортні засоби, пункти маршрутів); управління обліковими записами користувачів

Аналіз таблиці 1.3 свідчить про те, що найбільший перелік функцій зосереджено в ролі логіста, що підтверджує його центральну роль у функціонуванні системи. Функції, пов'язані з пошуком і вибором маршруту, є унікальними для цієї ролі і не доступні іншим акторам. На рис. наведено діаграму варіантів використання прикладної системи.

Рис. 4 Діаграма варіантів використання прикладної системи

Аналіз рисунка 4 показує, що між варіантами використання «Пошук маршрутів» і «Вибір оптимального маршруту» існує відношення включення (<<include>>): вибір маршруту неможливий без попереднього виконання пошуку та оцінювання альтернатив. Це відображає логіку роботи модуля маршрутизації і буде деталізовано при розробці алгоритмів у розділі 3.[7]

Діаграма діяльності процесу вибору маршруту. Для детального опису логіки найбільш складного і важливого процесу — вибору оптимального маршруту доставки — розроблено діаграму діяльності (activity diagram) у нотації UML.

Діаграма відображає послідовність дій логіста та системи від моменту ініціювання планування відвантаження до моменту фіксації обраного маршруту. На рис. 5 наведено діаграму діяльності процесу вибору маршруту доставки.

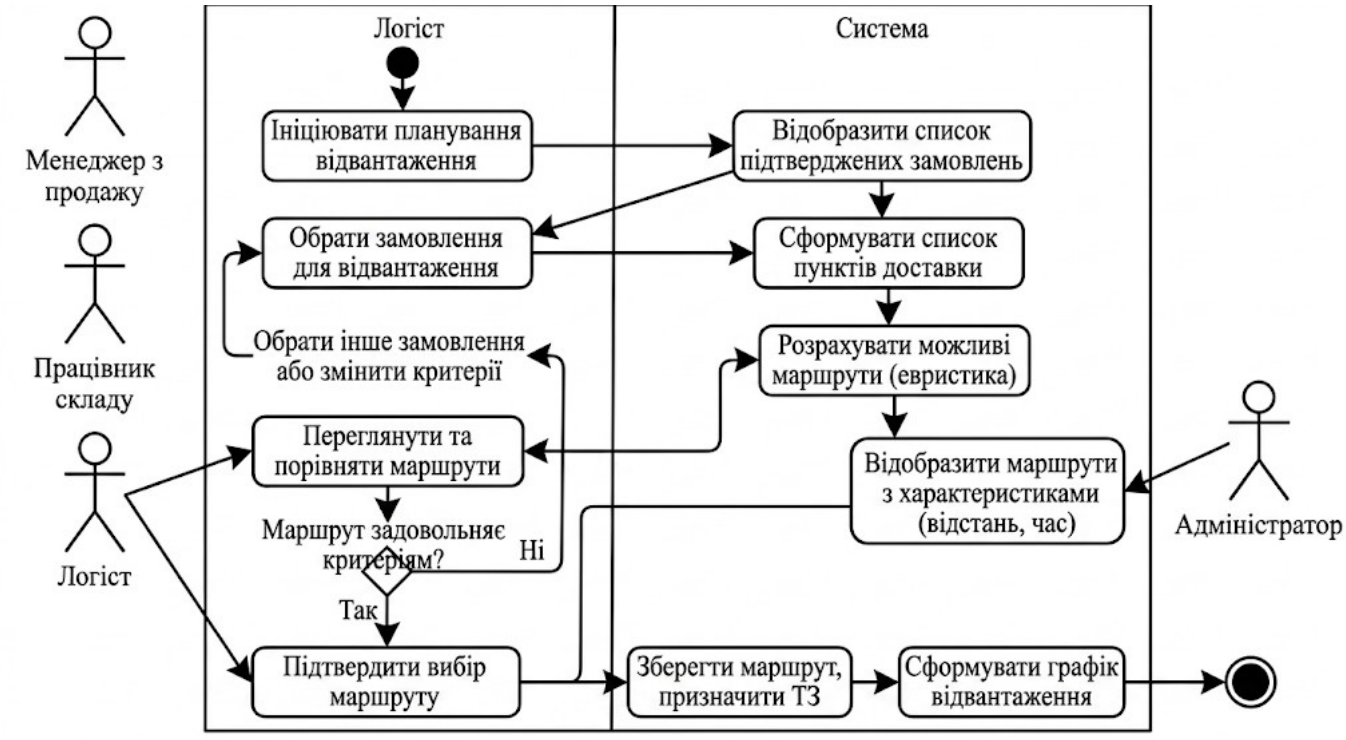

Рис. 5 Діаграма діяльності процесу вибору маршруту доставки

Аналіз рисунка 5 дозволяє встановити, що процес вибору маршруту є ітеративним: у разі якщо жоден із запропонованих системою маршрутів не задовольняє вимоги логіста, він може повернутися до попереднього кроку і переглянути параметри відбору. Система бере на себе обчислювальну частину – розрахунок і впорядкування маршрутів за характеристиками, — тоді як остаточне рішення залишається за людиною. Такий підхід відповідає концепції системи підтримки прийняття рішень і зумовлює архітектурне рішення щодо виділення окремого модуля маршрутизації у складі прикладної системи, яке буде розвинуте у розділі 3.[8]

Таким чином, розроблено комплекс моделей предметної області, що охоплює контекстну діаграму системи, діаграму варіантів використання та діаграму діяльності для ключового процесу маршрутизації. Розроблені моделі з достатньою повнотою описують функціональні процеси й інформаційні взаємодії у предметній

області та формують основу для проєктування інформаційного і програмного забезпечення системи у наступних розділах.[9

Висновки до розділу 1

У першому розділі бакалаврської кваліфікаційної роботи виконано системний аналіз предметної області транспортної логістики виробничого підприємства та сформовано методичну основу для подальшого проєктування інформаційної системи.

Проведено аналіз логістичної діяльності підприємства з виділенням основних процесів: управління запасами готової продукції, обробки клієнтських замовлень, планування відвантажень, використання транспортних засобів і підтримки прийняття рішень щодо вибору маршруту доставки. Показано, що застосування розрізнених засобів обліку (електронних таблиць, локальних програм, паперових документів) призводить до дублювання даних, неузгодженості інформації, підвищення ймовірності помилок та нераціонального використання транспортних ресурсів.

Сформульовано об’єкт дослідження (логістична діяльність абстрактного виробничого підприємства) та предмет дослідження (моделі, методи й програмні засоби автоматизації логістичних процесів з акцентом на транспортну логістику). Визначено задачу, яку має розв’язувати прикладна система: забезпечення інтегрованого обліку запасів, замовлень, відвантажень, транспортних засобів і маршрутів доставки, а також підтримка процедури пошуку та вибору маршруту доставки на основі заданих критеріїв.

На основі огляду теоретичних підходів до розв’язання задач маршрутизації (задача комівояжера, задача маршрутизації транспортних засобів) та аналізу існуючих програмних продуктів класу WMS, TMS і інтегрованих ERP-систем встановлено, що наявні рішення або орієнтовані на великі підприємства і характеризуються високою вартістю та складністю впровадження, або не забезпечують поєднання обліку запасів, обробки замовлень і базової маршрутизації в єдиній системі. Це обґрунтовує доцільність розробки спеціалізованої прикладної

системи підтримки транспортної логістики для абстрактного виробничого підприємства.

Побудовано комплекс моделей предметної області, що включає контекстну діаграму системи, діаграму варіантів використання та діаграму діяльності процесу вибору маршруту доставки. Зазначені моделі формалізують межі системи, ролі користувачів, їхні функції та внутрішню логіку ключових логістичних процесів, забезпечуючи концептуальну основу для проєктування логічної моделі даних та архітектури програмного забезпечення.

Таким чином, результати першого розділу дозволили сформувати цілісне уявлення про предметну область, уточнити формулювання мети й завдань роботи у контексті транспортної логістики та підготувати необхідне теоретичне й методичне підґрунтя для розробки інформаційного та прикладного програмного забезпечення в наступних розділах.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Інформаційне забезпечення прикладної системи підтримки транспортної логістики являє собою сукупність єдиної системи класифікації, уніфікованої системи документації та інформаційної бази, організованої на основі реляційної моделі даних. Проєктування інформаційного забезпечення починається зі створення логічної моделі даних, яка відображає структуру предметної області у вигляді сутностей, їхніх атрибутів і зв'язків між ними без прив'язки до конкретної системи управління базами даних.[1]

Логічна модель даних розробленої системи включає десять основних сутностей, визначених у підрозділі 1.1: «Продукція», «Склад», «Залишок», «Клієнт», «Замовлення», «Позиція_замовлення», «Відвантаження», «Транспортний_засіб», «Маршрут», «Пункт_маршруту». Усі зв'язки типу «багато-до-багатьох» реалізовано через асоціативні сутності, що відповідає вимогам третьої нормальної форми реляційної бази даних. Далі наведено детальний опис кожної сутності.

Сутність «Продукція» є базовим довідником номенклатури готової продукції підприємства. Вона зберігає інформацію, необхідну для однозначної ідентифікації кожного виду продукції, визначення одиниці виміру та масових характеристик, які використовуються при плануванні відвантажень і перевірці вантажопідйомності транспортних засобів. Атрибутний склад сутності «Продукція» наведено в табл. 2.1.

Таблиця 2.1

Атрибутний склад сутності «Продукція»

Атрибут	Тип даних	Призначення
id_product (PK)	INTEGER	Унікальний ідентифікатор продукції
name	VARCHAR(150)	Найменування продукції
unit	VARCHAR(20)	Одиниця виміру (шт., кг, л тощо)
weight_per_unit	DECIMAL(10,3)	Маса одиниці продукції (кг)
description	TEXT	Довільний опис продукції

Сутність «Продукція» не зберігає залишків безпосередньо — вони виносяться в окрему сутність «Залишок», що дозволяє незалежно вести облік однієї й тієї самої продукції на різних складах підприємства.

Сутність «Склад» містить довідникову інформацію про складські об'єкти підприємства, на яких зберігається готова продукція. Ця сутність необхідна для прив'язки залишків продукції до конкретних фізичних об'єктів та для визначення місця відвантаження при формуванні замовлень. Атрибутний склад сутності «Склад» наведено в таблиці 2.2.

Таблиця 2.2

Атрибутний склад сутності «Склад»

Атрибут	Тип даних	Призначення
id_warehouse (PK)	INTEGER	Унікальний ідентифікатор складу
name	VARCHAR(100)	Найменування складу
address	VARCHAR(250)	Адреса розташування складу
capacity	DECIMAL(12,2)	Максимальна ємність складу (кг або м³)

Атрибут «capacity» є основою для реалізації перевірки допустимого наповнення складу в майбутніх версіях системи та забезпечує можливість розширення функціоналу без зміни структури бази даних.

Сутність «Залишок» реалізує зв'язок між сутностями «Склад» і «Продукція» і зберігає поточну кількість кожного виду продукції на кожному складі. Ця сутність є центральною для функції управління запасами: саме її дані перевіряються при формуванні замовлення і оновлюються при підтвердженні відвантаження. Атрибутний склад сутності «Залишок» наведено в таблиці 2.3.

Таблиця 2.3

Атрибутний склад сутності «Залишок»

Атрибут	Тип даних	Призначення
id_stock (PK)	INTEGER	Унікальний ідентифікатор запису про залишок
id_warehouse (FK)	INTEGER	Посилання на склад
id_product (FK)	INTEGER	Посилання на продукцію
quantity	DECIMAL(12,3)	Поточна кількість продукції на складі

updated_at	DATETIME	Дата і час останнього оновлення залишку

Пара атрибутів (id_warehouse, id_product) утворює природний унікальний ключ сутності, що унеможливлює дублювання записів про залишок одного виду продукції на одному складі. Атрибут «updated_at» забезпечує можливість відстеження актуальності даних.

Сутність «Клієнт» є довідником замовників підприємства і зберігає контактні та адресні дані, необхідні як для оформлення замовлень, так і для функцій транспортної логістики. Особливої ваги набувають географічні координати адреси доставки, які використовуються алгоритмом маршрутизації при розрахунку відстаней між пунктами. Атрибутний склад сутності «Клієнт» наведено в таблиці 2.4.

Таблиця 2.4

Атрибутний склад сутності «Клієнт»

Атрибут	Тип даних	Призначення
id_client (PK)	INTEGER	Унікальний ідентифікатор клієнта
name	VARCHAR(200)	Повне найменування клієнта
contact_person	VARCHAR(150)	ПІБ контактної особи
phone	VARCHAR(20)	Контактний телефон
address	VARCHAR(250)	Адреса доставки за замовчуванням
latitude	DECIMAL(9,6)	Географічна широта адреси доставки
longitude	DECIMAL(9,6)	Географічна довгота адреси доставки

Атрибути «latitude» і «longitude» є ключовими для функціоналу маршрутизації: саме на основі цих координат модуль транспортної логістики розраховує матрицю відстаней між пунктами доставки та формує варіанти маршрутів відповідно до цільової функції, визначеної у підрозділі 1.1.

Сутність «Замовлення» фіксує факт звернення клієнта до підприємства з запитом на отримання продукції та є вихідним документом для всього подальшого ланцюжка логістичних операцій. Вона зберігає реквізити замовлення, часові параметри та поточний статус виконання, що забезпечує прозорість процесу обробки замовлень для всіх категорій користувачів системи. Атрибутний склад сутності «Замовлення» наведено в таблиці 2.5.

Таблиця 2.5

Атрибутний склад сутності «Замовлення»

Атрибут	Тип даних	Призначення
id_order (PK)	INTEGER	Унікальний ідентифікатор замовлення
id_client (FK)	INTEGER	Посилання на клієнта
order_date	DATE	Дата реєстрації замовлення
desired_delivery_date	DATE	Бажана дата доставки
status	VARCHAR(30)	Статус замовлення (нове, підтверджено, відвантажено, виконано)
notes	TEXT	Коментарі менеджера

Атрибут «status» реалізує простий, але ефективний механізм відстеження стану виконання замовлення на всіх етапах його обробки — від реєстрації менеджером до підтвердження доставки логістом. Перехід статусів є основою для формування звіту про стан виконання замовлень.

Сутність «Позиція_замовлення» деталізує склад кожного замовлення до рівня окремих видів продукції та їх кількості. Вона реалізує зв'язок «багато-до-багатьох» між сутностями «Замовлення» і «Продукція» та є аналогом рядка в товарній накладній або рахунку-фактурі. Атрибутний склад сутності «Позиція_замовлення» наведено в таблиці 2.6.

Таблиця 2.6

Атрибутний склад сутності «Позиція_замовлення»

Атрибут	Тип даних	Призначення
id_order_item (PK)	INTEGER	Унікальний ідентифікатор позиції
id_order (FK)	INTEGER	Посилання на замовлення
id_product (FK)	INTEGER	Посилання на продукцію
quantity	DECIMAL(12,3)	Кількість продукції у замовленні
id_warehouse (FK)	INTEGER	Посилання на склад відвантаження

Наявність атрибута «id_warehouse» на рівні позиції, а не замовлення в цілому, дозволяє системі підтримувати сценарій, при якому різні позиції одного замовлення відвантажуються з різних складів підприємства, що є типовим для розподіленої складської інфраструктури.

Сутність «Транспортний_засіб» є довідником рухомого складу підприємства і зберігає характеристики, необхідні для планування відвантажень і перевірки

можливості виконання доставки. Вона забезпечує логісту оперативний доступ до інформації про доступність транспортних засобів та їх вантажопідйомність при призначенні ТЗ на відвантаження. Атрибутний склад сутності «Транспортний_засіб» наведено в таблиці 2.7.

Таблиця 2.7

Атрибутний склад сутності «Транспортний_засіб»

Атрибут	Тип даних	Призначення
id_vehicle (PK)	INTEGER	Унікальний ідентифікатор транспортного засобу
reg_number	VARCHAR(20)	Реєстраційний номер
model	VARCHAR(100)	Марка та модель ТЗ
capacity_kg	DECIMAL(10,2)	Вантажопідйомність (кг)
status	VARCHAR(30)	Поточний статус (доступний, у рейсі, на обслуговуванні)

Атрибут «capacity_kg» у поєднанні з атрибутом «weight_per_unit» сутності «Продукція» забезпечує можливість реалізації автоматичної перевірки відповідності загальної маси вантажу вантажопідйомності призначеного транспортного засобу.

Сутність «Відвантаження» є центральною сполучною ланкою між обліковою і транспортною частинами системи. Вона фіксує факт планування та виконання доставки і поєднує в одному записі замовлення клієнта, призначений транспортний засіб і сформований маршрут. Саме через цю сутність забезпечується інтеграція функцій обліку запасів і транспортної логістики. Атрибутний склад сутності «Відвантаження» наведено в таблиці 2.8.

Таблиця 2.8

Атрибутний склад сутності «Відвантаження»

Атрибут	Тип даних	Призначення
id_shipment (PK)	INTEGER	Унікальний ідентифікатор відвантаження
id_order (FK)	INTEGER	Посилання на замовлення
id_vehicle (FK)	INTEGER	Посилання на транспортний засіб
id_route (FK)	INTEGER	Посилання на маршрут доставки

Закігчення таблиці 2.8

planned_date	DATE	Запланована дата відвантаження
actual_date	DATE	Фактична дата виконання доставки

status	VARCHAR(30)	Статус (заплановано, виконується, виконано)

Наявність атрибутів «planned_date» і «actual_date» забезпечує можливість контролю дотримання термінів доставки та формування аналітичних звітів про відхилення фактичних дат від запланованих, що є важливим показником якості роботи транспортного підрозділу підприємства.

Сутність «Маршрут» зберігає узагальнені характеристики сформованого маршруту доставки і є результатом роботи модуля маршрутизації. Вона дозволяє зберігати і порівнювати кілька альтернативних маршрутів для одного відвантаження та фіксувати той, який логіст обрав як оптимальний. Атрибутний склад сутності «Маршрут» наведено в таблиці 2.9.

Таблиця 2.9

Атрибутний склад сутності «Маршрут»

Атрибут	Тип даних	Призначення
id_route (PK)	INTEGER	Унікальний ідентифікатор маршруту
name	VARCHAR(150)	Умовна назва маршруту
total_distance_km	DECIMAL(10,2)	Загальна протяжність маршруту (км)
estimated_time_min	INTEGER	Орієнтовний час виконання маршруту (хв)
points_count	INTEGER	Кількість пунктів доставки
created_at	DATETIME	Дата і час формування маршруту
is_optimal	BOOLEAN	Ознака оптимального маршруту

Атрибут «is_optimal» дозволяє системі позначати рекомендований маршрут серед розрахованих альтернатив відповідно до цільової функції $F(\pi) \to min$, визначеної в підрозділі 1.1. Атрибут «total_distance_km» є основним критерієм порівняння маршрутів за замовчуванням.

Сутність «Пункт_маршруту» деталізує маршрут до рівня окремих пунктів обходу і зберігає впорядковану послідовність клієнтів, яких необхідно обслужити в межах конкретного маршруту. Ця сутність є прямим відображенням результату роботи алгоритму маршрутизації, що визначає оптимальну послідовність обслуговування клієнтів. Атрибутний склад сутності «Пункт_маршруту» наведено в таблиці 2.10.

Таблиця 2.10

Атрибутний склад сутності «Пункт_маршруту»

Атрибут	Тип даних	Призначення
id_point (PK)	INTEGER	Унікальний ідентифікатор пункту
id_route (FK)	INTEGER	Посилання на маршрут
id_client (FK)	INTEGER	Посилання на клієнта — пункт доставки
visit_order	INTEGER	Порядковий номер відвідування в маршруті
distance_from_prev_km	DECIMAL(8,2)	Відстань від попереднього пункту (км)
estimated_arrival_min	INTEGER	Орієнтовний час прибуття від початку маршруту (хв)

Атрибут «visit_order» фіксує оптимальну послідовність обслуговування клієнтів, визначену алгоритмом маршрутизації, а «distance_from_prev_km» забезпечує можливість детального аналізу структури маршруту і перевірки коректності роботи алгоритму.

Зв'язки між сутностями логічної моделі даних мають такий характер: «Склад» і «Продукція» пов'язані через «Залишок» відношенням «багато-до-багатьох»; «Клієнт» пов'язаний з «Замовленням» відношенням «один-до-багатьох»; «Замовлення» і «Продукція» пов'язані через «Позицію_замовлення» відношенням «багато-до-багатьох»; «Відвантаження» пов'язане з «Замовленням», «Транспортним_засобом» і «Маршрутом»; «Маршрут» і «Клієнт» пов'язані через «Пункт_маршруту» відношенням «багато-до-багатьох» зі збереженням послідовності обходу. Для унаочнення структури розробленої логічної моделі даних на рис. 6 наведено її ER-діаграму у нотації Crow's Foot.

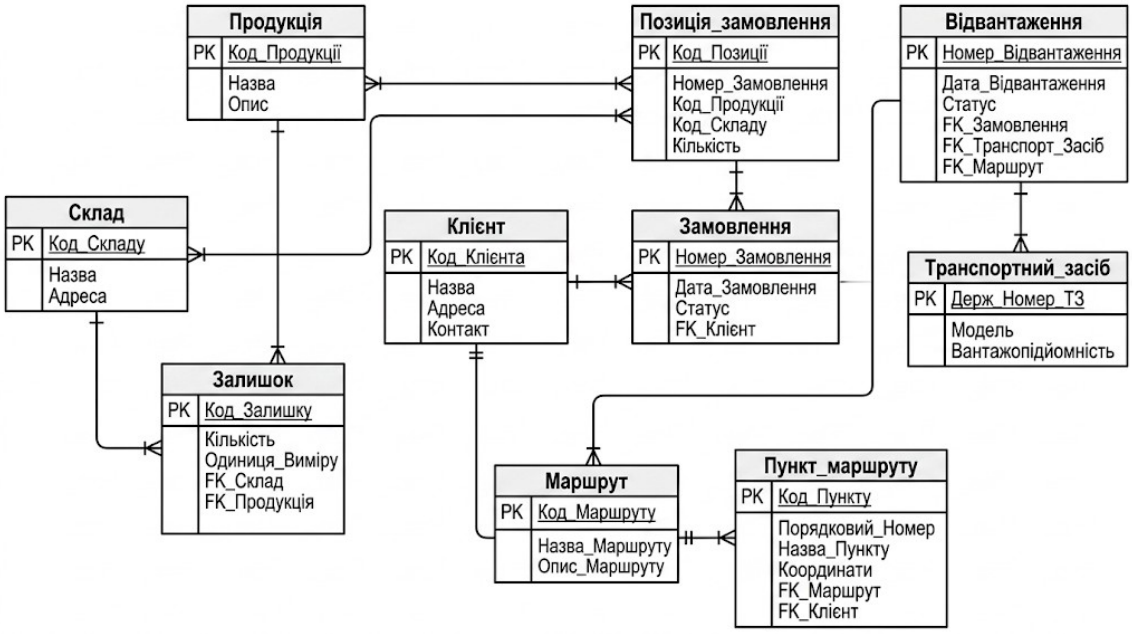

Рис.6 ER-діаграма логічної моделі даних прикладної системи

Аналіз рисунка 6 підтверджує, що логічна модель даних відповідає вимогам реляційності та третій нормальній формі: кожна сутність має первинний ключ, всі зв'язки «багато-до-багатьох» реалізовано через асоціативні сутності, а транзитивних залежностей між неключовими атрибутами не виявлено. Центральне місце в моделі займає сутність «Відвантаження», яка інтегрує облікову та транспортну складові системи, поєднуючи в одному записі замовлення, транспортний засіб і маршрут доставки. Розроблена логічна модель слугує безпосередньою основою для вибору системи управління базою даних та побудови фізичної моделі у підрозділах 2.2 та 2.3.

2.2 Вибір системи управління інформаційною базою

Вибір системи управління базою даних (СУБД) є одним із ключових проєктних рішень при розробці прикладної системи, оскільки він безпосередньо визначає надійність зберігання даних, продуктивність виконання запитів, масштабованість системи та трудомісткість її розробки і супроводу. Відповідно до логічної моделі даних, розробленої у підрозділі 2.1, інформаційна база прикладної системи є реляційною, що звужує коло розглядуваних кандидатів до реляційних СУБД, які підтримують мову SQL.

При виборі СУБД для прикладної системи підтримки транспортної логістики абстрактного виробничого підприємства доцільно враховувати такі критерії: відповідність реляційній моделі даних і підтримка цілісності посилань; продуктивність при виконанні запитів до взаємопов'язаних таблиць; можливість розгортання на типовому серверному обладнанні підприємства; вартість ліцензування та наявність безкоштовних варіантів; зрілість і наявність документації; простота адміністрування для команди розробників малого та середнього масштабу. Для порівняльної оцінки кандидатів розглянемо найбільш поширені реляційні СУБД, що застосовуються у прикладних системах подібного масштабу. Порівняльну характеристику розглянутих СУБД за ключовими критеріями наведено в таблиці 2.11.

Таблиця 2.11

Порівняльна характеристика реляційних СУБД за ключовими критеріями

Критерій	MySQL	PostgreSQL	MS SQL Server	SQLite
Ліцензія	GPL / комерційна	MIT (безкоштовна)	Комерційна	Public Domain
Підтримка зовнішніх ключів	Так	Так	Так	Обмежена
Продуктивність (середнє навантаження)	Висока	Висока	Висока	Низька
Підтримка транзакцій (ACID)	Так	Так	Так	Так
Складність адміністрування	Низька	Середня	Висока	Мінімальна
Масштабованість	Середня	Висока	Висока	Низька
Придатність для серверного розгортання	Так	Так	Так	Ні
Інструменти візуального адміністрування	MySQL Workbench	pgAdmin	SSMS	DB Browser

Аналіз таблиці 2.11 показує, що SQLite, незважаючи на мінімальну складність адміністрування, не є придатною для серверного розгортання і має обмежену підтримку зовнішніх ключів, що суперечить вимогам цілісності логічної моделі даних. MS SQL Server є потужним рішенням, однак потребує комерційного ліцензування і характеризується підвищеною складністю адміністрування, що є надмірним для масштабу розроблюваної системи. MySQL і PostgreSQL є найбільш

збалансованими кандидатами з точки зору функціональності, продуктивності та доступності.

Для більш детального обґрунтування вибору між MySQL і PostgreSQL проведемо їх порівняння за аспектами, що є найбільш важливими саме для розроблюваної системи. Деталізоване порівняння двох фінальних кандидатів наведено в таблиці 2.12.

Таблиця 2.12

Деталізоване порівняння MySQL та PostgreSQL

Аспект	MySQL	PostgreSQL
Підтримка складних JOIN-запитів	Задовільна	Відмінна
Розширені типи даних (DECIMAL, BOOLEAN)	Так	Так
Підтримка географічних типів даних	Обмежена (через плагін)	Вбудована (PostGIS)
Повнота підтримки стандарту SQL	Часткова	Висока
Підтримка тригерів і збережених процедур	Так	Так
Активність спільноти і документація	Дуже висока	Висока
Інтеграція з популярними фреймворками	Відмінна	Відмінна

Аналіз таблиці 2.12 дозволяє зробити висновок на користь PostgreSQL. По-перше, логічна модель даних системи передбачає зберігання географічних координат клієнтів (атрибути «latitude» і «longitude» сутності «Клієнт») і потенційне розширення функціоналу маршрутизації на основі реальних геоданих, для чого PostgreSQL з розширенням PostGIS є більш природним середовищем. По-друге, складність запитів до бази даних, зумовлена великою кількістю зв'язаних таблиць і необхідністю агрегування даних про маршрути і відвантаження, вимагає повноцінної підтримки стандарту SQL, яку PostgreSQL забезпечує краще за MySQL. По-третє, PostgreSQL розповсюджується під ліцензією MIT, що не накладає жодних обмежень на комерційне використання розробленої системи.

З точки зору організації інформаційної бази, розроблювана система використовує централізовану архітектуру: єдиний екземпляр СУБД PostgreSQL розгортається на сервері підприємства, а всі клієнтські застосунки звертаються до нього через мережу [8]. На рис. 7 наведено структурну схему організації інформаційної бази системи.

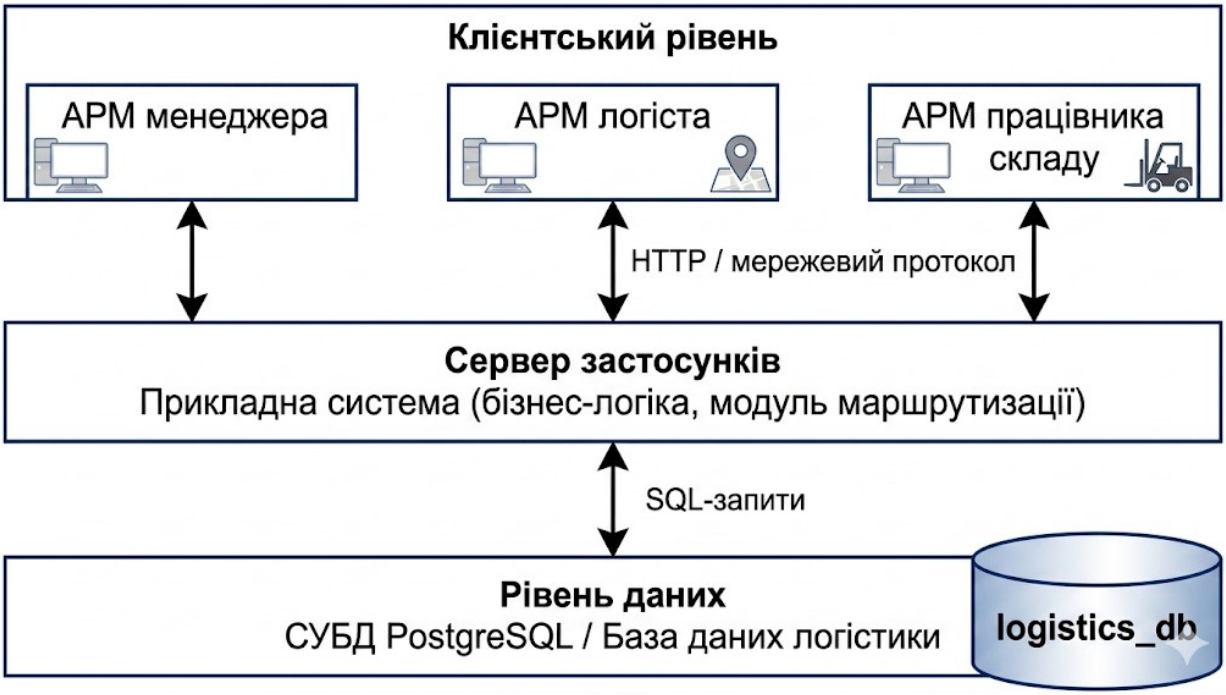

Рис.7 Структурна схема організації інформаційної бази прикладної системи

Аналіз рисунка 2.2 показує, що централізована архітектура інформаційної бази забезпечує єдину точку зберігання і доступу до даних для всіх категорій користувачів системи, що унеможливлює виникнення неузгодженостей між різними копіями даних. Сервер застосунків виступає посередником між клієнтськими робочими місцями і СУБД, реалізуючи бізнес-логіку та модуль маршрутизації незалежно від клієнтських засобів відображення.

З метою забезпечення цілісності, безпеки та надійності інформаційної бази передбачаються такі заходи: використання механізму транзакцій (ACID) при виконанні операцій списання залишків і оновлення статусів; визначення зовнішніх ключів з каскадним оновленням і забороною видалення батьківських записів за наявності пов'язаних дочірніх; розмежування прав доступу до таблиць на рівні СУБД відповідно до ролей користувачів; регулярне резервне копіювання бази даних засобами pg_dump.

Таким чином, для реалізації інформаційної бази прикладної системи підтримки транспортної логістики обрано СУБД PostgreSQL, що забезпечує необхідний рівень функціональності, надійності і масштабованості при

мінімальних витратах на ліцензування. Обрана СУБД є основою для розробки фізичної моделі даних у підрозділі 2.3.

2.3 Створення інформаційної бази

Фізична модель даних є результатом перетворення логічної моделі, розробленої у підрозділі 2.1, у конкретні об'єкти бази даних з урахуванням особливостей обраної СУБД — PostgreSQL. На відміну від логічної моделі, фізична модель визначає точні типи даних для кожного атрибута, механізми забезпечення цілісності (первинні та зовнішні ключі, обмеження), індекси для прискорення виконання запитів, а також послідовності (sequences) для автоматичного генерування значень первинних ключів.

Фізична реалізація інформаційної бази прикладної системи виконується у вигляді реляційної бази даних PostgreSQL з іменем logistics_db, яка містить десять таблиць, що відповідають сутностям логічної моделі. Усі таблиці розташовуються в схемі public. Нижче наведено фізичну модель кожної таблиці у вигляді DDL-скрипта з відповідними коментарями та аналізом прийнятих рішень.

Таблиця products (Продукція). Таблиця products є довідником номенклатури готової продукції підприємства. Для точного збереження маси одиниці продукції обрано тип NUMERIC(10,3), що запобігає виникненню похибок округлення, характерних для типів з рухомою комою. DDL-скрипт для створення таблиці products наведено в таблиці 2.13.

Таблиця 2.13

DDL-скрипт таблиці products

Елемент	Визначення
Ім'я таблиці	products
Первинний ключ	id_product SERIAL PRIMARY KEY
Обов'язкові поля	name NOT NULL, unit NOT NULL
Обмеження	weight_per_unit > 0

CREATE TABLE products (

```
    id_product    SERIAL          PRIMARY KEY,
    name          VARCHAR(150)    NOT NULL,
    unit          VARCHAR(20)     NOT NULL,
     weight_per_unit NUMERIC(10,3) NOT NULL CHECK (weight_per_unit >
0),
    description   TEXT
);
```

Використання типу SERIAL забезпечує автоматичне генерування первинного ключа засобами PostgreSQL. Обмеження CHECK (weight_per_unit > 0) реалізує бізнес-правило, згідно з яким маса одиниці продукції не може бути від'ємною або нульовою, що є необхідним для коректного розрахунку вантажопідйомності транспортних засобів.

Таблиця warehouses (Склад). Таблиця warehouses зберігає довідкову інформацію про складські об'єкти підприємства. Атрибут ємності складу визначено як NUMERIC(12,2) з обмеженням на невід'ємне значення, що забезпечує коректність вхідних даних при плануванні розміщення продукції. DDL-скрипт для створення таблиці warehouses наведено в таблиці 2.14.

Таблиця 2.14

DDL-скрипт таблиці warehouses

Елемент	Визначення
Ім'я таблиці	warehouses
Первинний ключ	id_warehouse SERIAL PRIMARY KEY
Обов'язкові поля	name NOT NULL, address NOT NULL
Обмеження	capacity >= 0

```
CREATE TABLE warehouses (
    id_warehouse  SERIAL          PRIMARY KEY,
    name          VARCHAR(100)    NOT NULL,
    address       VARCHAR(250)    NOT NULL,
    capacity      NUMERIC(12,2)   CHECK (capacity >= 0)
);
```

Поле capacity визначено як необов'язкове (NULL допускається), оскільки на початковому етапі впровадження системи ємність окремих складів може бути

невідомою. Це рішення забезпечує гнучкість при наповненні довідника без порушення цілісності даних.

Таблиця stock (Залишок). Таблиця stock реалізує зв'язок «багато-до-багатьох» між warehouses і products та є основним джерелом даних для перевірки наявності продукції при формуванні замовлень. Для унеможливлення дублювання записів про залишки визначено складений унікальний ключ. DDL-скрипт для створення таблиці stock наведено в таблиці 2.15.

Таблиця 2.15

DDL-скрипт таблиці stock

Елемент	Визначення
Ім'я таблиці	stock
Первинний ключ	id_stock SERIAL PRIMARY KEY
Зовнішні ключі	id_warehouse → warehouses, id_product → products
Унікальний ключ	(id_warehouse, id_product)
Обмеження	quantity >= 0

```
CREATE TABLE stock (
  id_stock      SERIAL         PRIMARY KEY,
  id_warehouse  INTEGER        NOT NULL
                REFERENCES warehouses(id_warehouse)
                ON DELETE RESTRICT,
  id_product    INTEGER        NOT NULL
                REFERENCES products(id_product)
                ON DELETE RESTRICT,
  quantity      NUMERIC(12,3)  NOT NULL DEFAULT 0
                CHECK (quantity >= 0),
  updated_at    TIMESTAMP      NOT NULL DEFAULT NOW(),
  UNIQUE (id_warehouse, id_product)
);
```

Директива ON DELETE RESTRICT для обох зовнішніх ключів унеможливлює видалення складу або продукції, якщо в таблиці залишків існують пов'язані записи. Значення DEFAULT 0 для поля quantity забезпечує автоматичну

ініціалізацію нового запису нульовим залишком при додаванні нової позиції до довідника складу.

Таблиця clients (Клієнт). Таблиця clients є довідником замовників підприємства і містить географічні координати адреси доставки, які безпосередньо використовуються модулем маршрутизації. Для збереження координат обрано тип NUMERIC(9,6), що забезпечує точність до шести знаків після коми, достатню для геолокації з точністю до ~0,1 м. DDL-скрипт для створення таблиці clients наведено в таблиці 2.16.

Таблиця 2.16

DDL-скрипт таблиці clients

Елемент	Визначення
Ім'я таблиці	clients
Первинний ключ	id_client SERIAL PRIMARY KEY
Обов'язкові поля	name NOT NULL
Обмеження	latitude між −90 і +90; longitude між −180 і +180

```
CREATE TABLE clients (
    id_client       SERIAL          PRIMARY KEY,
    name            VARCHAR(200)    NOT NULL,
    contact_person  VARCHAR(150),
    phone           VARCHAR(20),
    address         VARCHAR(250),
    latitude        NUMERIC(9,6)    CHECK (latitude  BETWEEN -90  AND 90),
     longitude        NUMERIC(9,6)     CHECK (longitude BETWEEN -180 AND
180)
);
```

Обмеження CHECK на діапазони широти і довготи реалізують валідацію географічних координат на рівні бази даних, що запобігає збереженню некоректних значень незалежно від рівня перевірок у прикладному коді.

Таблиця orders (Замовлення). Таблиця orders фіксує замовлення клієнтів і є стартовою точкою логістичного ланцюжка системи. Поле статусу реалізовано за допомогою типу-переліку CHECK з фіксованим набором допустимих значень, що забезпечує контрольований перехід між станами обробки замовлення. DDL-скрипт для створення таблиці orders наведено в таблиці 2.17.

Таблиця 2.17

DDL-скрипт таблиці orders

Елемент	Визначення
Ім'я таблиці	orders
Первинний ключ	id_order SERIAL PRIMARY KEY
Зовнішній ключ	id_client → clients
Допустимі статуси	new, confirmed, shipped, delivered, cancelled

```
CREATE TABLE orders (
   id_order            SERIAL     PRIMARY KEY,
   id_client           INTEGER    NOT NULL
                       REFERENCES clients(id_client)
                       ON DELETE RESTRICT,
   order_date          DATE       NOT NULL DEFAULT CURRENT_DATE,
   desired_delivery_date DATE,
   status              VARCHAR(30) NOT NULL DEFAULT 'new'
                       CHECK (status IN
                        ('new','confirmed','shipped','delivered','cancelled')),
   notes               TEXT
);
```

Значення DEFAULT 'new' для поля status автоматично присвоює новоствореному замовленню початковий статус, що унеможливлює наявність замовлення з невизначеним станом. Обмеження CHECK на допустимі статуси реалізує кінцевий автомат переходів стану замовлення на рівні бази даних.

Таблиця order_items (Позиція замовлення). Таблиця order_items деталізує склад кожного замовлення і зберігає інформацію про кількість кожного виду продукції та склад відвантаження. Для забезпечення узгодженості з таблицею stock

у цій таблиці також присутній зовнішній ключ на таблицю warehouses, що фіксує склад, з якого плануватиметься відвантаження конкретної позиції. DDL-скрипт для створення таблиці order_items наведено в таблиці 2.18.

Таблиця 2.18

DDL-скрипт таблиці order_items

Елемент	Визначення
Ім'я таблиці	order_items
Первинний ключ	id_order_item SERIAL PRIMARY KEY
Зовнішні ключі	id_order → orders; id_product → products; id_warehouse → warehouses
Обмеження	quantity > 0

```
CREATE TABLE order_items (
   id_order_item SERIAL         PRIMARY KEY,
   id_order      INTEGER        NOT NULL
                 REFERENCES orders(id_order)
                 ON DELETE CASCADE,
   id_product    INTEGER        NOT NULL
                 REFERENCES products(id_product)
                 ON DELETE RESTRICT,
   id_warehouse  INTEGER        NOT NULL
                 REFERENCES warehouses(id_warehouse)
                 ON DELETE RESTRICT,
   quantity      NUMERIC(12,3)   NOT NULL CHECK (quantity > 0)
);
```

Для зовнішнього ключа на таблицю orders застосовано директиву ON DELETE CASCADE: якщо замовлення видаляється (наприклад, як помилково створене), усі його позиції видаляються автоматично, що відповідає природній бізнес-логіці та запобігає утворенню «осиротілих» записів.

Таблиця vehicles (Транспортний засіб). Таблиця vehicles є довідником рухомого складу підприємства. Поле статусу визначено аналогічно до таблиці orders — через обмеження CHECK з фіксованим переліком допустимих значень,

що відображають реальні стани транспортного засобу в логістичному процесі. DDL-скрипт для створення таблиці vehicles наведено в таблиці 2.19.

Таблиця 2.19

DDL-скрипт таблиці vehicles

Елемент	Визначення
Ім'я таблиці	vehicles
Первинний ключ	id_vehicle SERIAL PRIMARY KEY
Унікальне поле	reg_number UNIQUE
Допустимі статуси	available, on_route, maintenance

```
CREATE TABLE vehicles (
    id_vehicle    SERIAL          PRIMARY KEY,
    reg_number    VARCHAR(20)     NOT NULL UNIQUE,
    model         VARCHAR(100),
    capacity_kg   NUMERIC(10,2)   NOT NULL CHECK (capacity_kg > 0),
    status        VARCHAR(30)     NOT NULL DEFAULT 'available'
                  CHECK (status IN ('available','on_route','maintenance'))
);
```

Обмеження UNIQUE на поле reg_number забезпечує унікальність реєстраційного номера кожного транспортного засобу в системі, що є очевидною бізнес-вимогою і запобігає помилковому дублюванню записів довідника.

Таблиця routes (Маршрут). Таблиця routes зберігає результати роботи модуля маршрутизації — розраховані варіанти маршрутів з їхніми узагальненими характеристиками. Поле is_optimal типу BOOLEAN дозволяє позначати обраний логістом оптимальний маршрут серед усіх розрахованих альтернатив. DDL-скрипт для створення таблиці routes наведено в таблиці 2.20.

Таблиця 2.20

DDL-скрипт таблиці routes

Елемент	Визначення
Ім'я таблиці	routes
Первинний ключ	id_route SERIAL PRIMARY KEY
Обмеження	total_distance_km >= 0; estimated_time_min >= 0
Ознака оптимуму	is_optimal BOOLEAN DEFAULT FALSE

```
CREATE TABLE routes (
    id_route            SERIAL          PRIMARY KEY,
    name                VARCHAR(150),
    total_distance_km   NUMERIC(10,2)   CHECK (total_distance_km >= 0),
    estimated_time_min  INTEGER         CHECK (estimated_time_min >= 0),
    points_count        INTEGER         CHECK (points_count > 0),
    created_at          TIMESTAMP       NOT NULL DEFAULT NOW(),
    is_optimal          BOOLEAN         NOT NULL DEFAULT FALSE
);
```

Значення DEFAULT FALSE для поля is_optimal гарантує, що новостворений маршрут за замовчуванням не вважається оптимальним — ознака оптимальності присвоюється лише явно після порівняльного аналізу альтернатив модулем маршрутизації або логістом.

Таблиця shipments (Відвантаження). Таблиця shipments є центральною сполучною таблицею системи, що поєднує замовлення, транспортний засіб і маршрут в єдиному записі про доставку. Наявність полів planned_date і actual_date дозволяє системі автоматично розраховувати відхилення факту від плану при формуванні аналітичних звітів. DDL-скрипт для створення таблиці shipments наведено в таблиці 2.21.

Таблиця 2.21

DDL-скрипт таблиці shipments

Елемент	Визначення
Ім'я таблиці	shipments
Первинний ключ	id_shipment SERIAL PRIMARY KEY
Зовнішні ключі	id_order → orders; id_vehicle → vehicles; id_route → routes
Допустимі статуси	planned, in_progress, completed, cancelled

```
CREATE TABLE shipments (
    id_shipment   SERIAL      PRIMARY KEY,
    id_order      INTEGER     NOT NULL
                  REFERENCES orders(id_order)
```

```
        ON DELETE RESTRICT,
    id_vehicle    INTEGER
        REFERENCES vehicles(id_vehicle)
        ON DELETE SET NULL,
    id_route      INTEGER
        REFERENCES routes(id_route)
        ON DELETE SET NULL,
    planned_date  DATE        NOT NULL,
    actual_date   DATE,
    status        VARCHAR(30) NOT NULL DEFAULT 'planned'
        CHECK (status IN
         ('planned','in_progress','completed','cancelled'))
);
```

Для зовнішніх ключів id_vehicle та id_route застосовано директиву ON DELETE SET NULL: якщо транспортний засіб або маршрут видаляється з довідника, запис про відвантаження зберігається з порожнім посиланням, що дозволяє переназначити ресурс без втрати облікового запису про доставку.

Таблиця route_points (Пункт маршруту). Таблиця route_points деталізує маршрут до рівня послідовності конкретних клієнтів і є прямим відображенням результату роботи алгоритму маршрутизації. Поле visit_order визначає впорядковану послідовність обслуговування клієнтів і є ключовим при побудові карти маршруту у клієнтському інтерфейсі системи. DDL-скрипт для створення таблиці route_points наведено в таблиці 2.22.

Таблиця 2.22

DDL-скрипт таблиці route_points

Елемент	Визначення
Ім'я таблиці	route_points
Первинний ключ	id_point SERIAL PRIMARY KEY
Зовнішні ключі	id_route → routes; id_client → clients
Унікальний ключ	(id_route, visit_order)
Обмеження	visit_order >= 1

```
CREATE TABLE route_points (
    id_point            SERIAL      PRIMARY KEY,
    id_route            INTEGER     NOT NULL
                        REFERENCES routes(id_route)
                        ON DELETE CASCADE,
    id_client           INTEGER     NOT NULL
                        REFERENCES clients(id_client)
                        ON DELETE RESTRICT,
    visit_order         INTEGER     NOT NULL CHECK (visit_order >= 1),
    distance_from_prev_km  NUMERIC(8,2)  CHECK (distance_from_prev_km
>= 0),
    estimated_arrival_min  INTEGER      CHECK (estimated_arrival_min >= 0),
    UNIQUE (id_route, visit_order)
);
```

Складений унікальний ключ (id_route, visit_order) унеможливлює присвоєння двом пунктам одного маршруту однакового порядкового номера, що гарантує коректність і однозначність послідовності обходу, яку генерує алгоритм маршрутизації.

Індекси та підсумковий DDL. Для забезпечення продуктивності виконання типових запитів прикладної системи — пошуку залишків за складом і продукцією, вибірки позицій замовлення, побудови маршрутів — на таблицях бази даних створюються додаткові індекси. Перелік індексів, що рекомендуються до створення, наведено в таблиці 2.23.

Таблиця 2.23

Індекси бази даних logistics_db

Індекс	Таблиця	Поля	Призначення
idx_stock_warehouse	stock	id_warehouse	Пошук залишків за складом
idx_stock_product	stock	id_product	Пошук залишків за продукцією
idx_orders_client	orders	id_client	Вибірка замовлень клієнта

idx_orders_status	orders	status	Фільтрація замовлень за статусом
idx_order_items_order	order_items	id_order	Вибірка позицій замовлення
idx_shipments_order	shipments	id_order	Пошук відвантажень за замовленням
idx_route_points_route	route_points	id_route	Побудова пунктів маршруту

```
CREATE INDEX idx_stock_warehouse      ON stock(id_warehouse);
CREATE INDEX idx_stock_product        ON stock(id_product);
CREATE INDEX idx_orders_client        ON orders(id_client);
CREATE INDEX idx_orders_status        ON orders(status);
CREATE INDEX idx_order_items_order    ON order_items(id_order);
CREATE INDEX idx_shipments_order      ON shipments(id_order);
CREATE INDEX idx_route_points_route   ON route_points(id_route);
```

Аналіз таблиці 2.23 показує, що індекси охоплюють усі зовнішні ключі з вираженим виборчим характером запитів, а також поле статусу таблиці orders, по якому регулярно виконується фільтрація при формуванні оперативних зведень. Первинні ключі типу SERIAL індексуються PostgreSQL автоматично і до цього переліку не включаються.[1]

Таким чином, фізична модель інформаційної бази прикладної системи включає десять таблиць з визначеними первинними та зовнішніми ключами, обмеженнями цілісності CHECK і NOT NULL, сімома додатковими індексами та системою директив ON DELETE, що забезпечує узгоджене каскадне управління пов'язаними записами. Повний DDL-скрипт, що містить усі описані вище операції CREATE TABLE та CREATE INDEX, є безпосередньою основою для фізичного розгортання бази даних logistics_db та її подальшого наповнення тестовими даними в ході розробки прикладної системи.

Висновки до розділу 2

У розділі 2 було сформовано цілісне інформаційне забезпечення прикладної системи підтримки транспортної логістики, що охоплює логічне та фізичне

проєктування реляційної бази даних і вибір конкретної СУБД для її реалізації. На основі системного аналізу предметної області побудовано логічну модель даних, яка відображає основні сутності (продукція, склади, замовлення, клієнти, маршрути, відвантаження, транспортні засоби тощо) та зв’язки між ними відповідно до вимог третьої нормальної форми. Це забезпечує усунення надлишковості, мінімізує ризики аномалій при оновленні даних і створює передумови для ефективної реалізації складних аналітичних запитів.

Обґрунтовано вибір реляційної СУБД PostgreSQL як платформи для зберігання і обробки даних системи, з урахуванням критеріїв функціональності, продуктивності, масштабованості, вартості ліцензування та підтримки географічних даних. Порівняльний аналіз показав доцільність використання PostgreSQL порівняно з іншими популярними СУБД (MySQL, MS SQL Server, SQLite) завдяки кращій підтримці стандарту SQL, розширенням для роботи з геоданими та відсутності ліцензійних обмежень для комерційного використання. Обрана архітектура централізованої інформаційної бази з єдиним сервером БД забезпечує узгодженість даних та спрощує адміністрування.

На основі логічної моделі та обраної СУБД розроблено фізичну модель інформаційної бази, яка охоплює десять таблиць, систему первинних і зовнішніх ключів, обмеження цілісності (NOT NULL, CHECK, UNIQUE), каскадні правила ON DELETE та набір індексів для прискорення типових запитів. Детальний DDL-опис таблиць products, warehouses, stock, clients, orders, order_items, vehicles, routes, shipments, route_points забезпечує однозначну реалізацію бази даних logistics_db у середовищі PostgreSQL та створює фундамент для подальшої розробки прикладного програмного забезпечення. Таким чином, інформаційне забезпечення розробленої системи повністю готове до інтеграції з програмними модулями, що будуть описані у наступному розділі.

РОЗДІЛ 3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Прикладне програмне забезпечення (ППЗ) розробленої системи підтримки транспортної логістики являє собою багаторівневий програмний комплекс, організований за принципом розмежування відповідальності між функціональними компонентами. Відповідно до загальноприйнятого підходу до проєктування прикладних систем, ППЗ логічно поділяється на пакети, кожен з яких реалізує самостійну функціональну роль і взаємодіє з іншими пакетами через чітко визначені інтерфейси. Такий підхід забезпечує незалежність розробки та тестування окремих компонентів, спрощує супровід і подальше розширення системи.[1]

Організаційна структура ППЗ системи підтримки транспортної логістики складається з п'яти основних пакетів, що утворюють дворівневу архітектуру «клієнт — сервер». Клієнтський рівень містить пакет інтерфейсу користувача та пакет мережевого з'єднання; серверний рівень — пакет бізнес-логіки, пакет доступу до даних і пакет формування звітів. Діаграму пакетів ППЗ системи наведено на рис. 7

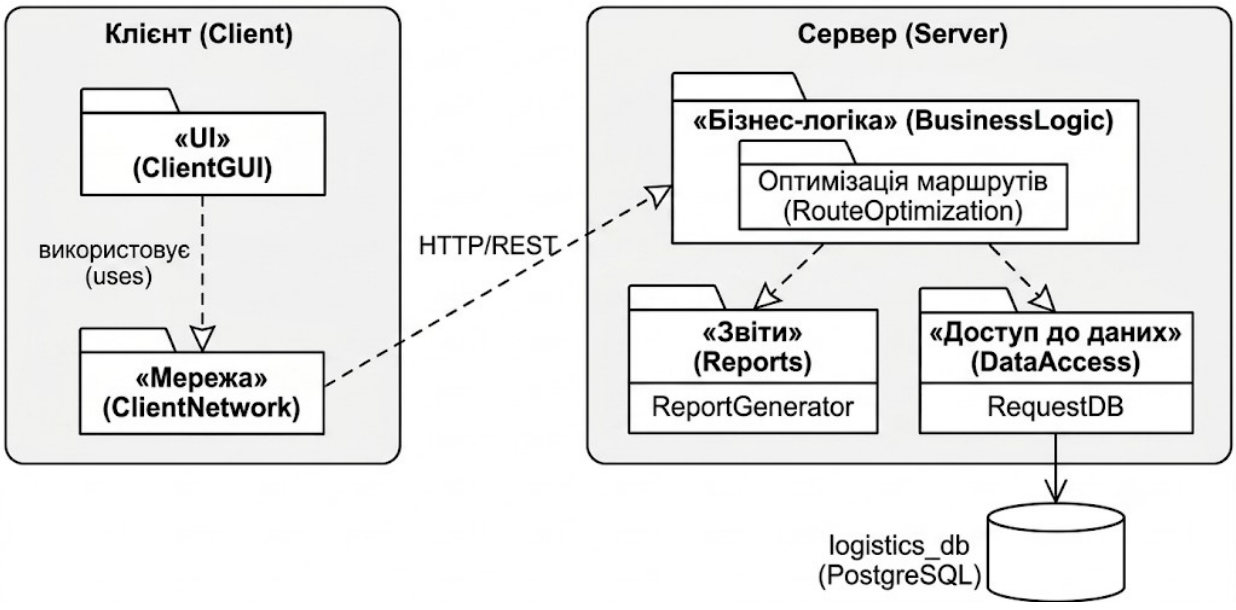

Рис.8 Діаграма пакетів прикладного програмного забезпечення системи

Аналіз діаграми рис. 8 показує чітке розмежування клієнтської та серверної частин системи, що відповідає трирівневій архітектурі «представлення — бізнес-логіка — дані». Пакет ClientGUI взаємодіє виключно з пакетом ClientNetwork, а не безпосередньо з сервером або базою даних, що забезпечує незалежність інтерфейсу від технології передачі даних. Нижче наведено детальний опис кожного пакета. [9]

Пакет ClientGUI (Інтерфейс користувача). Пакет ClientGUI реалізує графічний інтерфейс користувача для всіх категорій працівників системи. До його складу входять екранні форми трьох автоматизованих робочих місць: АРМ менеджера (реєстрація та перегляд замовлень), АРМ логіста (формування відвантажень, перегляд маршрутів, призначення транспортних засобів) і АРМ працівника складу (перегляд і коригування залишків продукції). Склад та призначення екранних форм пакета ClientGUI наведено в таблиці 3.1.

Таблиця 3.1

Склад екранних форм пакета ClientGUI

Екранна форма	АРМ	Призначення
frmLogin	Усі	Авторизація користувача в системі
frmMainMenu	Усі	Головне меню відповідно до ролі користувача
frmClientList	Менеджер	Перегляд і редагування довідника клієнтів
frmOrderNew	Менеджер	Створення нового замовлення
frmOrderList	Менеджер	Перегляд списку замовлень і їх статусів
frmOrderDetail	Менеджер	Деталізація замовлення за позиціями
frmStockView	Склад	Перегляд поточних залишків продукції
frmStockAdjust	Склад	Ручне коригування залишків
frmShipmentList	Логіст	Перегляд запланованих відвантажень
frmShipmentNew	Логіст	Створення відвантаження і призначення ТЗ
frmRouteView	Логіст	Перегляд маршруту на карті та в табличному вигляді
frmRouteOptimize	Логіст	Запуск розрахунку оптимального маршруту
frmVehicleList	Логіст	Перегляд і редагування довідника ТЗ
frmReportViewer	Усі	Перегляд і вивантаження сформованих звітів

Таблиця 3.1 демонструє, що пакет ClientGUI охоплює повний набір операцій предметної області системи, визначений у підрозділі 1.1, а розмежування форм за АРМами відповідає рольовій моделі доступу, передбаченій при проєктуванні інформаційної бази.

Пакет ClientNetwork (Мережева взаємодія клієнта). Пакет ClientNetwork відповідає за формування та надсилання HTTP-запитів від клієнтського застосунку до сервера, а також за обробку відповідей сервера і передачу отриманих даних до пакета ClientGUI. Він реалізує єдину точку входу для всіх звернень клієнтського застосунку до серверного API, забезпечуючи серіалізацію та десеріалізацію даних у форматі JSON, обробку помилок мережевої взаємодії (тайм-аути, втрата з'єднання) та прозоре відображення статусів відповідей у вигляді, зрозумілому для бізнес-логіки клієнтського застосунку. Використання окремого пакета для мережевої взаємодії забезпечує можливість заміни транспортного протоколу (наприклад, переходу з HTTP на WebSocket для реалізації реального часу) без будь-яких змін у пакеті ClientGUI.[10]

Пакет BusinessLogic (Бізнес-логіка сервера). Пакет BusinessLogic є центральним компонентом серверної частини системи, що реалізує всі бізнес-правила обробки замовлень, управління залишками і транспортної логістики. Він приймає запити від клієнтського рівня, перевіряє коректність вхідних даних відповідно до бізнес-правил (наприклад, достатність залишків для виконання замовлення, відповідність загальної маси вантажу вантажопідйомності призначеного транспортного засобу), виконує обробку і передає результат до пакета DataAccess для збереження або до пакета Reports для формування документів. Склад основних компонентів пакета BusinessLogic наведено в таблиці 3.2.[11]

Таблиця 3.2

Основні компоненти пакета BusinessLogic

Компонент	Призначення
OrderService	Обробка операцій зі створення, підтвердження та скасування замовлень
StockService	Перевірка і резервування залишків при формуванні позицій замовлення
ShipmentService	Управління відвантаженнями: призначення ТЗ, зв'язок із маршрутом
VehicleService	Управління довідником ТЗ і перевірка вантажопідйомності
RouteOptimization	Розрахунок оптимального маршруту доставки (підпакет)
ClientService	Управління довідником клієнтів і геоданими
AuthService	Аутентифікація та авторизація користувачів за роллю

Особливе місце у складі пакета BusinessLogic займає підпакет RouteOptimization, який реалізує алгоритм маршрутизації транспортних засобів на основі цільової функції $F(\pi) \to min$, визначеної у підрозділі 1.1. Цей підпакет отримує на вхід географічні координати клієнтів з таблиці clients бази даних та формує оптимальну послідовність обходу пунктів доставки, яка зберігається в таблицях routes і route_points. Детальний опис алгоритму маршрутизації наведено у підрозділі 3.3.

Пакет DataAccess (Доступ до даних). Пакет DataAccess ізолює всю взаємодію серверної частини системи з базою даних logistics_db від бізнес-логіки, реалізуючи патерн Repository. Кожна сутність бази даних, описана у розділі 2, відповідає окремому репозиторію (ProductRepository, WarehouseRepository, OrderRepository тощо), що надає стандартизований набір операцій: отримання запису за ідентифікатором, вибірка за критеріями, створення, оновлення і видалення запису. Завдяки цьому підходу пакет BusinessLogic не містить жодних SQL-запитів і не залежить безпосередньо від PostgreSQL, що забезпечує теоретичну можливість заміни СУБД без зміни бізнес-логіки.[13]

Пакет Reports (Формування звітів). Пакет Reports відповідає за генерацію документів і аналітичних звітів за даними системи та їх надання користувачам у форматах PDF та XLSX. До складу пакета входять генератори таких звітів: «Реєстр замовлень» (за клієнтом, датою, статусом), «Відомість відвантажень» (за плановою і фактичною датами), «Маршрутний лист» (для водія — впорядкована послідовність пунктів доставки з адресами і відстанями), «Залишки продукції по складах» та «Аналіз відхилень термінів доставки». Склад і призначення звітів пакета Reports наведено в таблиці 3.3.[15]

Аналіз таблиці 3.3 показує, що звітний підсистема системи покриває всі ключові інформаційні потреби категорій користувачів: менеджери отримують реєстр замовлень, логісти — маршрутний лист і відомість відвантажень, керівництво — аналітичні звіти про відхилення термінів доставки.

Таблиця 3.3

Перелік звітних документів пакета Reports

Звіт	Формат виводу	Ключові параметри фільтрації
Реєстр замовлень	PDF, XLSX	Клієнт, дата, статус замовлення
Відомість відвантажень	PDF	Дата відвантаження, транспортний засіб
Маршрутний лист	PDF	Ідентифікатор відвантаження
Залишки по складах	XLSX	Склад, дата формування
Аналіз відхилень доставки	XLSX	Звітний період

Усі звітні документи формуються на основі даних, що зберігаються в базі даних logistics_db, розробленій у підрозділі 2.3.[12]

Таким чином, організаційна структура ППЗ прикладної системи підтримки транспортної логістики реалізована за принципом трирівневої архітектури і включає п'ять функціональних пакетів: ClientGUI, ClientNetwork, BusinessLogic (з підпакетом RouteOptimization), DataAccess та Reports. Розроблена структура забезпечує чіткий розподіл відповідальності між компонентами, незалежність їх розробки і тестування та технічну готовність до розширення функціоналу системи в майбутньому. На основі визначеної організаційної структури у підрозділі 3.2 здійснюється вибір інструментарію для реалізації кожного пакета.[14]

3.2 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір інструментальних засобів розробки прикладного програмного забезпечення є важливим технічним рішенням, що безпосередньо впливає на трудомісткість реалізації, якість кінцевого продукту та зручність його демонстрації і супроводу. Відповідно до організаційної структури ППЗ, визначеної у підрозділі 3.1, система складається з п'яти функціональних пакетів, для реалізації яких необхідно обрати: мову програмування та серверний фреймворк, засіб з'єднання з базою даних, технологію клієнтського інтерфейсу, бібліотеки формування звітів і середовище розробки. При виборі інструментарію керувалися такими критеріями: відповідність технічним вимогам системи, доступність і відкритість ліцензії, зрілість екосистеми та якість документації, простота інтеграції компонентів між собою, зручність демонстрації результату на захисті.[16]

Вибір мови програмування. Першочерговим кроком є вибір мови програмування, яка визначає технологічний стек усіх інших компонентів системи. Для прикладних систем рівня підприємства найбільш поширеними варіантами є Python, Java, C# та JavaScript (Node.js). Порівняльну характеристику цих мов за ключовими критеріями розробленої системи наведено в таблиці 3.4.

Таблиця 3.4

Порівняльна характеристика мов програмування для реалізації ППЗ

Критерій	Python	Java	C# (.NET)	Node.js
Поріг входу та читабельність коду	Дуже низький	Середній	Середній	Низький
Наявність бібліотек маршрутизації та геоданих	Відмінна (NetworkX, scipy, geopy)	Достатня	Достатня	Обмежена
Інтеграція з PostgreSQL	psycopg2 (нативна)	JDBC (зріла)	Npgsql (зріла)	pg (зріла)
Генерація PDF/XLSX звітів	ReportLab, openpyxl	iText, Apache POI	iTextSharp, ClosedXML	Обмежена
Швидкість прототипування	Дуже висока	Низька	Середня	Середня
Відкрита ліцензія	Так	Так (частково)	Так (.NET Core)	Так
Зручність демонстрації	Веб у браузері	Потребує JVM	Потребує .NET runtime	Веб у браузері

Аналіз таблиці 3.4 свідчить, що Python є найбільш збалансованим вибором для реалізації розробленої системи. По-перше, Python має найвищу швидкість прототипування серед розглянутих мов, що є суттєвим при обмеженому часі виконання бакалаврської роботи. По-друге, екосистема Python містить зрілі бібліотеки для всіх ключових функцій системи — від доступу до PostgreSQL до реалізації алгоритмів маршрутизації на основі теорії графів. По-третє, Python-додатки є платформонезалежними і не потребують встановлення громіздких середовищ виконання на комп'ютері комісії під час захисту. Таким чином, для реалізації ППЗ обрано мову програмування Python версії 3.11.

Вибір серверного фреймворку. Серверна частина системи реалізує пакети BusinessLogic, DataAccess і Reports відповідно до діаграми пакетів рис. 3.1. Для Python існує кілька поширених веб-фреймворків: Flask, Django і FastAPI. Для

обґрунтованого вибору між ними необхідно порівняти їх за вимогами системи. Порівняльну характеристику Python-фреймворків наведено в таблиці 3.5.

Таблиця 3.5

Порівняльна характеристика серверних фреймворків Python

Критерій	Flask	Django	FastAPI
Підхід	Мікрофреймворк	Повностековий	Асинхронний API-фреймворк
Складність налаштування	Мінімальна	Висока (багато вбудованих компонентів)	Середня
Гнучкість архітектури	Повна свобода	Жорстка структура	Висока
Вбудована ORM	Ні (підключається SQLAlchemy)	Так (Django ORM)	Ні (підключається SQLAlchemy)
Генерація HTML-інтерфейсу	Через Jinja2	Через Django Templates	Потребує окремого рішення
Підтримка REST API	Через Flask-RESTful	Через DRF	Нативна
Підходить для навчального проєкту	Відмінно	Надлишково	Частково

Аналіз таблиці 3.5 показує, що Django є надлишковим для масштабу розробленої системи — його численні вбудовані компоненти (адмін-панель, ORM, система міграцій) збільшують трудомісткість освоєння без пропорційного виграшу для навчального проєкту. FastAPI орієнтований на побудову чистих REST API без вбудованого механізму генерації HTML-сторінок, що потребує окремого клієнтського застосунку. Flask, натомість, є мінімалістичним мікрофреймворком, який надає рівно той набір інструментів, що необхідний для розробленої системи: маршрутизацію HTTP-запитів, шаблонізатор Jinja2 для генерації HTML-форм і сторінок, гнучке підключення будь-яких бібліотек без нав'язаних архітектурних рішень. Таким чином, для реалізації серверної частини ППЗ обрано фреймворк Flask версії 3.0[17]

Вибір засобу доступу до бази даних. Взаємодія серверної частини системи з базою даних logistics_db здійснюється через пакет DataAccess відповідно до архітектури, визначеної у підрозділі 3.1. Для Python і PostgreSQL існують два основних підходи: використання низькорівневого драйвера psycopg2 з прямими

SQL-запитами, або підключення ORM-бібліотеки SQLAlchemy, що надає об'єктно-реляційне відображення. Порівняльну характеристику цих підходів наведено в таблиці 3.6.

Таблиця 3.6

Порівняльна характеристика засобів доступу до PostgreSQL

Критерій	psycopg2 (прямий SQL)	SQLAlchemy (ORM)
Прозорість виконуваних запитів	Повна (SQL пишеться вручну)	Часткова (генерується ORM)
Відповідність навчальній меті	Висока (демонструє знання SQL)	Середня
Швидкість розробки	Середня	Висока
Складність налаштування	Мінімальна	Середня
Відповідність DDL-моделі підрозділу 2.3	Пряма	Потребує відображення

Для розробленої системи обрано підхід на основі psycopg2 з прямими параметризованими SQL-запитами. Цей вибір обґрунтований двома міркуваннями. По-перше, у підрозділі 2.3 вже розроблено детальну фізичну модель бази даних зі структурою таблиць і DDL-скриптами, тому пряме використання SQL-запитів є природним продовженням цього проєктного рішення і не вимагає додаткового рівня абстракції. По-друге, використання psycopg2 з явними SQL-запитами демонструє комісії на захисті реальне знання мови SQL і структури бази даних, що є однією з оцінюваних компетентностей. Обрано psycopg2 версії 2.9.[17]

Вибір технології клієнтського інтерфейсу. Клієнтський інтерфейс системи, що відповідає пакету ClientGUI відповідно до підрозділу 3.1, реалізується як веб-додаток, що відображається у стандартному браузері без встановлення додаткового програмного забезпечення на робочих місцях користувачів. Для формування HTML-сторінок на стороні сервера використовується шаблонізатор Jinja2, вбудований у Flask. Для стилізації і компонентів інтерфейсу (таблиці, форми, кнопки, навігаційне меню) обрано CSS-фреймворк Bootstrap версії 5.3, що надає готовий набір адаптивних компонентів без необхідності написання власного CSS-коду. Переваги обраного підходу порівняно з альтернативами наведено в таблиці 3.7.

Таблиця 3.7

Порівняльна характеристика підходів до реалізації клієнтського інтерфейсу

Підхід	Переваги	Недоліки для даного проєкту
Jinja2 + Bootstrap (обрано)	Не потребує окремого фронтенд-фреймворку; проста розробка і налагодження; демонстрація в браузері	Обмежена інтерактивність без JavaScript
React / Vue (SPA)	Сучасний інтерфейс, висока інтерактивність	Потребує окремого білду; вища складність; надлишково для навчального проєкту
WinForms / Tkinter (десктоп)	Простий у налагодженні	Прив'язка до ОС; менш сучасний вигляд; ускладнена демонстрація

Jinja2 у поєднанні з Bootstrap дозволяє реалізувати повнофункціональний інтерфейс для всіх трьох АРМів системи (менеджер, логіст, працівник складу) без залучення окремого JavaScript-фреймворку, що суттєво знижує загальну складність стека і зосереджує зусилля розробника на реалізації бізнес-логіки і алгоритму маршрутизації.

Вибір бібліотек для звітів та алгоритму маршрутизації. Для реалізації пакета Reports обрано дві спеціалізовані бібліотеки Python: ReportLab для генерації PDF-документів (маршрутний лист, відомість відвантажень) та openpyxl для формування XLSX-файлів (реєстр замовлень, аналіз відхилень). Обидві бібліотеки розповсюджуються під відкритими ліцензіями, мають детальну документацію і широко використовуються у промислових проєктах. [18]

Для реалізації підпакета RouteOptimization, що обчислює оптимальну послідовність обходу пунктів доставки, обрано бібліотеку geopy для розрахунку відстаней між географічними координатами клієнтів (за формулою Хаверсіна) та власну реалізацію алгоритму «найближчого сусіда» (Nearest Neighbor Heuristic) як ефективного та прозорого методу розв'язання задачі комівояжера для типових розмірів маршруту (5–15 пунктів). Детальний опис алгоритму маршрутизації наведено у підрозділі 3.3.

Перелік усіх обраних інструментальних засобів з вказівкою версій та призначення наведено в таблиці 3.8.

Таблиця 3.8

Зведений перелік обраного інструментарію розробки ППЗ

Інструмент	Версія	Призначення	Ліцензія
Python	3.11	Мова програмування серверної частини	PSF (відкрита)
Flask	3.0	Серверний веб-фреймворк	BSD
psycopg2	2.9	Драйвер доступу до PostgreSQL	LGPL
Jinja2	3.1	Шаблонізатор HTML-інтерфейсу	BSD
Bootstrap	5.3	CSS-фреймворк клієнтського інтерфейсу	MIT
ReportLab	4.1	Генерація PDF-звітів	BSD
openpyxl	3.1	Генерація XLSX-звітів	MIT
geopy	2.4	Розрахунок відстаней між координатами	MIT
PostgreSQL	16	СУБД	PostgreSQL License
VS Code	актуальна	Середовище розробки	MIT

Аналіз таблиці 3.8 показує, що увесь обраний стек складається виключно з програмного забезпечення з відкритим кодом і вільними ліцензіями, що повністю виключає витрати на ліцензування як при розробці, так і при впровадженні системи на підприємстві. Середовищем розробки обрано **Visual Studio Code** з розширеннями Python, Pylance і SQLTools — безкоштовний крос-платформний редактор, що забезпечує підсвічування синтаксису, автодоповнення коду, інтегрований відладчик і підключення до PostgreSQL для перевірки SQL-запитів безпосередньо в середовищі розробки. [19]

Таким чином, обраний технологічний стек — Python 3.11 / Flask / psycopg2 / Jinja2 + Bootstrap / ReportLab + openpyxl / geopy — повністю відповідає функціональним вимогам системи, визначеним у підрозділі 1.1, забезпечує реалізацію всіх п'яти пакетів ППЗ відповідно до діаграми пакетів рис. 3.1 і є оптимальним з точки зору балансу між функціональністю і трудомісткістю розробки в умовах бакалаврської кваліфікаційної роботи. На основі визначеного інструментарію у підрозділі 3.3 здійснюється алгоритмізація та програмування ключових модулів системи.

3.3 Алгоритмізація та програмування програмних модулів

Відповідно до організаційної структури ППЗ, визначеної у підрозділі 3.1, прикладна система підтримки транспортної логістики реалізується у вигляді Flask-

додатку з п'ятьма функціональними пакетами. У цьому підрозділі розглядаються алгоритми та ключові фрагменти програмного коду найважливіших модулів системи: модуля авторизації і розмежування доступу, модуля управління замовленнями і перевірки залишків, модуля маршрутизації транспортних засобів, а також модуля формування звітів. Вибір саме цих модулів для детального опису обумовлений їх визначальним значенням для реалізації функцій системи та наявністю нетривіальних алгоритмічних рішень.[20]

Структура проєкту. Перед розглядом окремих модулів необхідно визначити файлову структуру проєкту, яка відображає поділ ППЗ на пакети відповідно до діаграми рис. 3.1. Структуру каталогів Flask-проєкту наведено в таблиці 3.9.

Таблиця 3.9

Файлова структура Flask-проєкту системи

Каталог / файл	Відповідний пакет	Призначення
app/__init__.py	—	Ініціалізація Flask-застосунку, реєстрація Blueprint
app/config.py	—	Параметри підключення до БД, секретний ключ сесії

Закінчення таблиці 3.9

app/db.py	DataAccess	Функції підключення до PostgreSQL через psycopg2
app/services/order_service.py	BusinessLogic	OrderService: створення, підтвердження замовлень
app/services/stock_service.py	BusinessLogic	StockService: перевірка і списання залишків
app/services/route_service.py	BusinessLogic/RouteOptimization	Розрахунок оптимального маршруту
app/services/ report_service.py	Reports	Генерація PDF і XLSX звітів
app/routes/manager.py	ClientGUI	Blueprint маршрутів АРМ менеджера
app/routes/logist.py	ClientGUI	Blueprint маршрутів АРМ логіста
app/routes/warehouse.py	ClientGUI	Blueprint маршрутів АРМ складу
app/templates/	ClientGUI	HTML-шаблони Jinja2 для всіх АРМів
app/static/	ClientGUI	CSS (Bootstrap), JavaScript, зображення

run.py	—	Точка запуску застосунку
requirements.txt	—	Перелік залежностей проєкту

Наведена структура відповідає рекомендованому підходу Blueprint у Flask, де кожен АРМ реалізується як окремий Blueprint з власним набором маршрутів (URL), що забезпечує модульність і незалежність розробки кожної частини інтерфейсу.[21]

Модуль авторизації та розмежування доступу. Система підтримує три ролі користувачів: менеджер (manager), логіст (logist) і працівник складу (warehouse). Після введення облікових даних система перевіряє їх відповідність записам у таблиці users бази даних і зберігає роль користувача у серверній сесії Flask. Алгоритм авторизації користувача наведено на рис. 9.

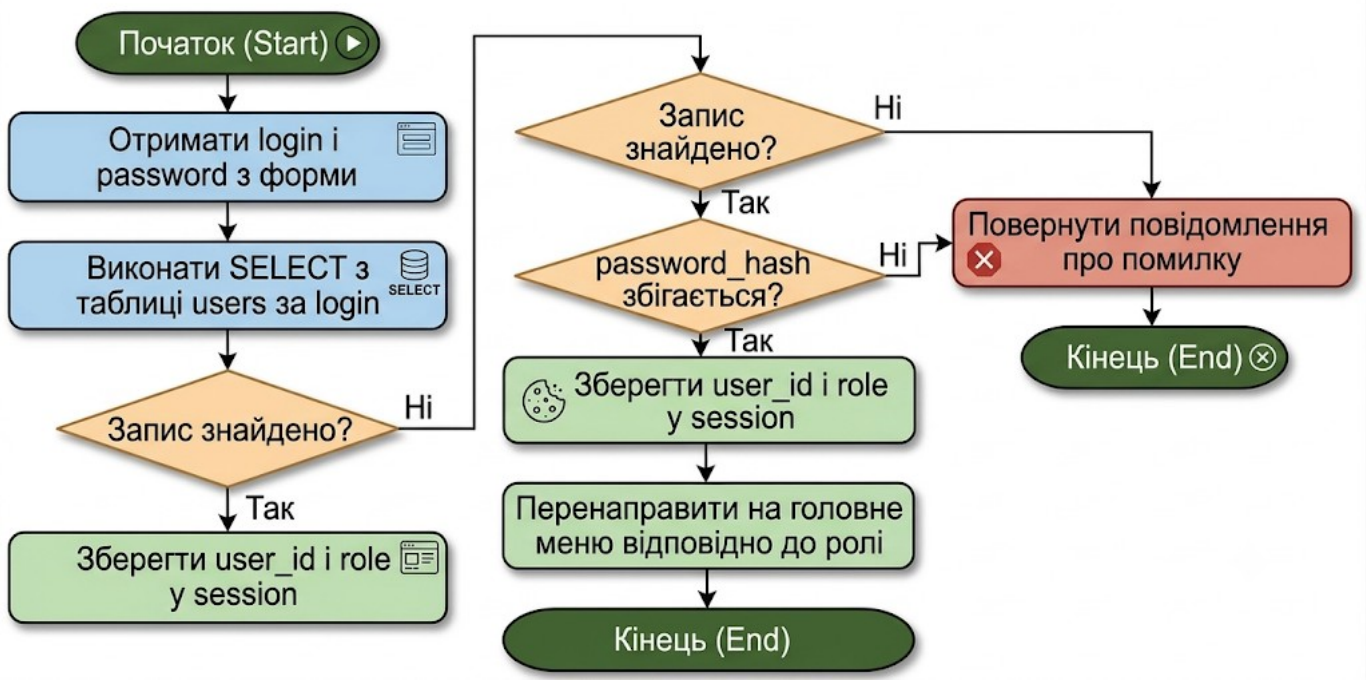

Рис.9 Блок-схема алгоритму авторизації користувача

Ключовий фрагмент програмного коду модуля авторизації (файл app/routes/auth.py) наведено нижче:

```
from flask import Blueprint, request, session, redirect, url_for, render_template
from app.db import get_connection
import hashlib

auth_bp = Blueprint('auth', __name__)

@auth_bp.route('/login', methods=['GET', 'POST'])
```

```
def login():
    if request.method == 'POST':
        login_val = request.form['login']
        password  = request.form['password']
        pwd_hash  = hashlib.sha256(password.encode()).hexdigest()

        conn = get_connection()
        cur  = conn.cursor()
        cur.execute(
            "SELECT id, role FROM users WHERE login=%s AND password_hash=%s",
            (login_val, pwd_hash)
        )
        user = cur.fetchone()
        cur.close()
        conn.close()

        if user:
            session['user_id'] = user[^0]
            session['role']    = user[^1]
            return redirect(url_for(f"{user[^1]}.index"))
        return render_template('login.html', error='Невірний логін або пароль')
    return render_template('login.html')
```

Для захисту маршрутів від несанкціонованого доступу реалізовано декоратор require_role, який перевіряє наявність активної сесії та відповідність ролі користувача перед виконанням кожного обробника запиту. Це забезпечує, що АРМ логіста недоступний для менеджера і навпаки, без дублювання перевірок у кожному обробнику.[1]

Модуль управління замовленнями і перевірки залишків. Модуль OrderService разом зі StockService реалізує центральний бізнес-процес системи — прийом замовлення та перевірку достатності залишків на складі. При підтвердженні замовлення система перевіряє кожну позицію і лише після успішної перевірки всіх позицій списує залишки та оновлює статус замовлення. Алгоритм підтвердження замовлення наведено на рис. 10.

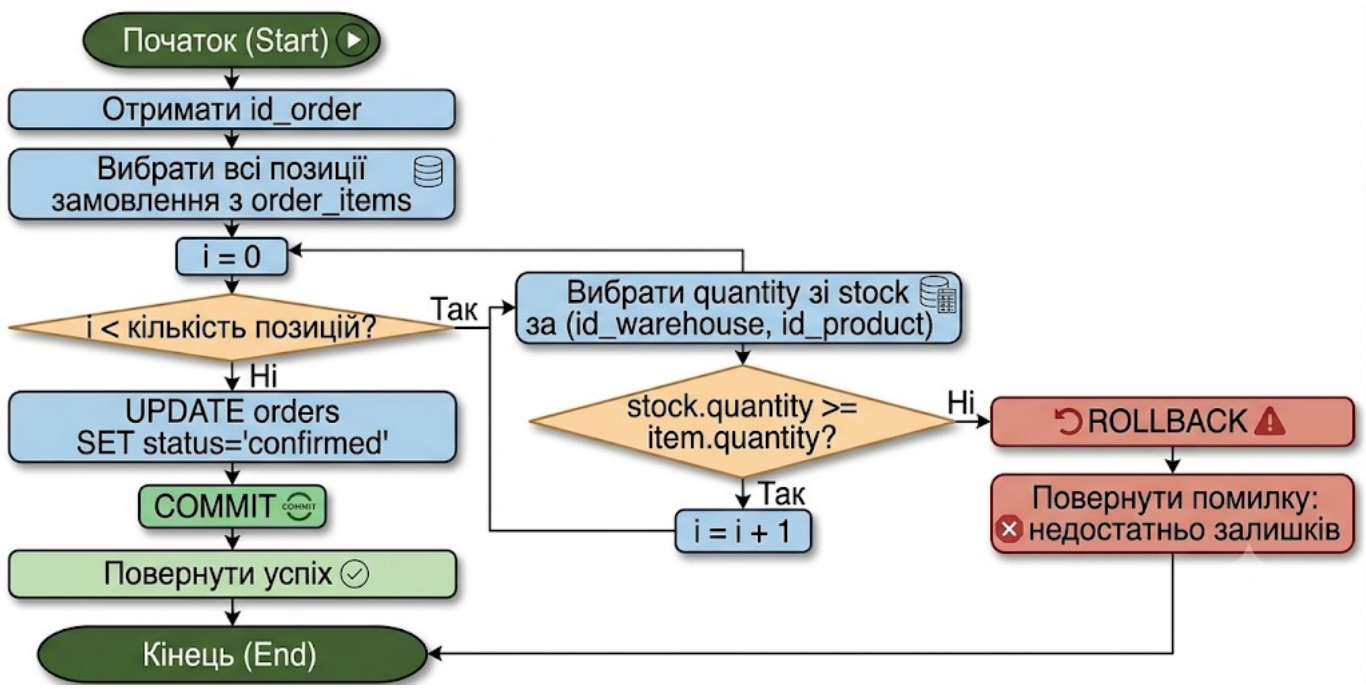

Рис. 10 Блок-схема алгоритму підтвердження замовлення з перевіркою залишків

Ключовий фрагмент програмного коду модуля (файл app/services/order_service.py) наведено нижче:

```
        from app.db import get_connection

def confirm_order(order_id: int) -> dict:
    conn = get_connection()
    cur  = conn.cursor()
    try:
        cur.execute("""
            SELECT id_product, id_warehouse, quantity
            FROM order_items WHERE id_order = %s
        """, (order_id,))
        items = cur.fetchall()

        for id_product, id_warehouse, qty in items:
            cur.execute("""
                SELECT quantity FROM stock
                WHERE id_product=%s AND id_warehouse=%s
                FOR UPDATE
            """, (id_product, id_warehouse))
            row = cur.fetchone()
            if not row or row[^0] < qty:
                conn.rollback()
                return {'success': False,
                        'message': f'Недостатньо залишків: продукт {id_product}'}
            cur.execute("""
                UPDATE stock SET quantity = quantity - %s,
```

```
                updated_at = NOW()
                WHERE id_product=%s AND id_warehouse=%s
            """, (qty, id_product, id_warehouse))

        cur.execute("""
            UPDATE orders SET status='confirmed' WHERE id_order=%s
        """, (order_id,))
        conn.commit()
        return {'success': True}

    except Exception as e:
        conn.rollback()
        return {'success': False, 'message': str(e)}
    finally:
        cur.close()
        conn.close()
```

Використання інструкції FOR UPDATE у SQL-запиті до таблиці stock забезпечує блокування рядка на рівні СУБД PostgreSQL на час виконання транзакції, що унеможливлює одночасне списання залишків двома паралельними запитами — так звану «гонку за даними» (race condition). Весь процес підтвердження виконується в межах однієї транзакції: якщо хоча б одна позиція не може бути виконана, виконується ROLLBACK і залишки залишаються незміненими.[22]

Модуль маршрутизації транспортних засобів. Модуль RouteOptimization є ключовим функціональним елементом системи, що відрізняє її від простої облікової системи і надає функціонал підтримки прийняття рішень для логіста. Завдання модуля — для заданого набору адрес клієнтів відвантаження знайти послідовність обходу пунктів, що мінімізує загальну відстань маршруту. Це є класичною задачею комівояжера (TSP), яка при практичних розмірах (5–15 пунктів) ефективно вирішується евристичним алгоритмом «найближчого сусіда» (Nearest Neighbor Heuristic).[23]

Алгоритм найближчого сусіда працює за таким принципом: починаючи від складу відвантаження як стартового пункту, на кожному кроці обирається незвіданий пункт, відстань до якого від поточного мінімальна, і він включається до маршруту. Процес повторюється до включення всіх пунктів. Алгоритм наведено на рис. 11.

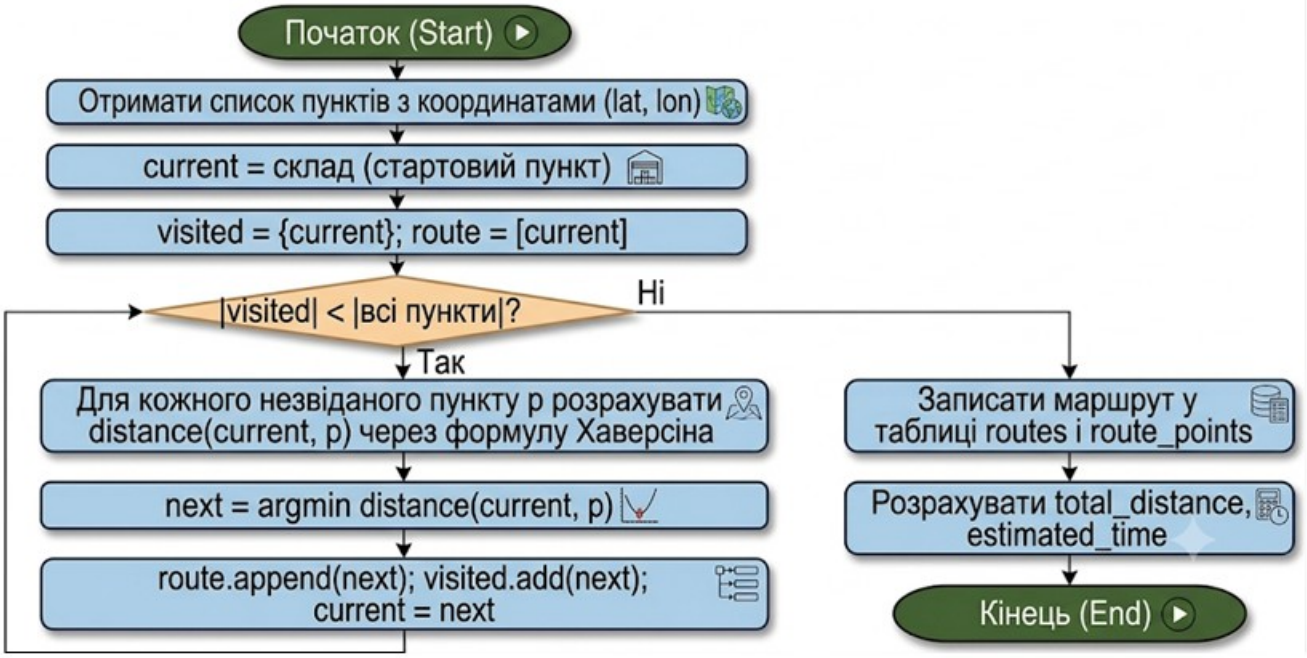

Рис.11 Блок-схема алгоритму маршрутизації «найближчого сусіда»

Ключовий фрагмент програмного коду модуля (файл app/services/route_service.py) наведено нижче:

```
from geopy.distance import geodesic
from app.db import get_connection

def calculate_route(shipment_id: int, warehouse_coords: tuple) -> dict:
    conn = get_connection()
    cur  = conn.cursor()

    cur.execute("""
        SELECT c.id_client, c.latitude, c.longitude
        FROM order_items oi
        JOIN orders o ON oi.id_order = o.id_order
        JOIN shipments s ON s.id_order = o.id_order
        JOIN clients c ON o.id_client = c.id_client
        WHERE s.id_shipment = %s
    """, (shipment_id,))
    points = cur.fetchall()  # [(id_client, lat, lon), ...]

    # Алгоритм найближчого сусіда
    unvisited = list(points)
    current   = warehouse_coords
    route     = []
    total_km  = 0.0

    while unvisited:
        nearest = min(unvisited,
                      key=lambda p: geodesic(current, (p[^1], p[^2])).km)
        dist_km = geodesic(current, (nearest[^1], nearest[^2])).km
```

```
        total_km += dist_km
        route.append({'id_client': nearest[^0],
                      'lat': nearest[^1], 'lon': nearest[^2],
                      'dist_from_prev': round(dist_km, 2)})
        current   = (nearest[^1], nearest[^2])
        unvisited.remove(nearest)

    # Зберегти маршрут у БД
    cur.execute("""
        INSERT INTO routes (total_distance_km, points_count,
                            created_at, is_optimal)
        VALUES (%s, %s, NOW(), TRUE)
        RETURNING id_route
    """, (round(total_km, 2), len(route)))
    id_route = cur.fetchone()[^0]

    for order, point in enumerate(route, start=1):
        cur.execute("""
            INSERT INTO route_points
            (id_route, id_client, visit_order, distance_from_prev_km)
            VALUES (%s, %s, %s, %s)
        """, (id_route, point['id_client'], order,
              point['dist_from_prev']))

    cur.execute("UPDATE shipments SET id_route=%s WHERE id_shipment=%s",
                (id_route, shipment_id))
    conn.commit()
    cur.close()
    conn.close()
    return {'id_route': id_route, 'total_km': round(total_km, 2),
            'route': route}
```

Відстань між пунктами розраховується за формулою Хаверсіна через функцію geodesic бібліотеки geopy, яка враховує сферичну форму Землі і дає точність, достатню для задач транспортної логістики в межах одного регіону. Часова складність алгоритму становить $O(n^2)$, де n — кількість пунктів доставки, що є прийнятним для типових розмірів маршруту (до 15 пунктів) без будь-яких затримок виконання.[25]

Модуль формування звітів. Модуль ReportService реалізує генерацію документів для всіх категорій користувачів системи відповідно до переліку звітів таблиці 3.3. Розглянемо реалізацію генерації маршрутного листа у форматі PDF - найважливішого документа для водія транспортного засобу, що містить впорядкований перелік адрес доставки з відстанями. Ключовий фрагмент коду

генерації маршрутного листа (файл app/services/report_service.py) наведено нижче: [26]

```
        from reportlab.lib.pagesizes import A4
from reportlab.platypus import (SimpleDocTemplate, Table,
                        TableStyle, Paragraph)
from reportlab.lib import colors
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
import io
from app.db import get_connection

def generate_route_pdf(id_route: int) -> bytes:
    pdfmetrics.registerFont(TTFont('DejaVu', 'DejaVuSans.ttf'))
    styles = getSampleStyleSheet()

    conn = get_connection()
    cur  = conn.cursor()
    cur.execute("""
        SELECT rp.visit_order, c.name, c.address,
               rp.distance_from_prev_km
        FROM route_points rp
        JOIN clients c ON rp.id_client = c.id_client
        WHERE rp.id_route = %s
        ORDER BY rp.visit_order
    """, (id_route,))
    rows = cur.fetchall()
    cur.close()
    conn.close()

    buffer = io.BytesIO()
    doc    = SimpleDocTemplate(buffer, pagesize=A4)

    data = [['№', 'Клієнт', 'Адреса', 'Відстань (км)']]
    for row in rows:
        data.append(list(row))

    table = Table(data, colWidths=[30, 150, 230, 80])
    table.setStyle(TableStyle([
        ('BACKGROUND', (0,0), (-1,0), colors.grey),
        ('TEXTCOLOR',  (0,0), (-1,0), colors.white),
        ('FONTNAME',   (0,0), (-1,-1), 'DejaVu'),
        ('FONTSIZE',   (0,0), (-1,-1), 10),
        ('GRID',       (0,0), (-1,-1), 0.5, colors.black),
        ('ROWBACKGROUNDS', (0,1), (-1,-1),
         [colors.white, colors.lightgrey]),
    ]))
    doc.build([Paragraph(f'Маршрутний лист № {id_route}',
                         styles['Title']), table])
```

return *buffer.getvalue()*

Реєстрація шрифту DejaVuSans забезпечує коректне відображення кирилиці у згенерованому PDF-документі, що є обов'язковою умовою для використання системи на україномовних підприємствах. Функція повертає байтовий об'єкт, який Flask-обробник надсилає браузеру з заголовком Content-Type: application/pdf, що ініціює завантаження файлу без збереження проміжного файлу на диску.[27]

Таким чином, у підрозділі 3.3 розроблено алгоритми і реалізовано ключові програмні модулі системи: модуль авторизації з розмежуванням доступу за роллю, модуль управління замовленнями з транзакційною перевіркою залишків, модуль маршрутизації на основі евристичного алгоритму «найближчого сусіда» з розрахунком відстаней за формулою Хаверсіна, а також модуль генерації PDF-звітів з підтримкою кирилиці. Усі модулі реалізовано відповідно до організаційної структури ППЗ, визначеної у підрозділі 3.1, із застосуванням інструментарію, обґрунтованого у підрозділі 3.2. Повний лістинг програмних модулів системи наведено у додатку А пояснювальної записки.

Висновки до розділу 3

У розділі 3 розроблено повне прикладне програмне забезпечення системи підтримки транспортної логістики виробничого підприємства, що охоплює організаційну структуру ППЗ, обґрунтований вибір технологічного стека та реалізацію ключових програмних модулів.

Визначено організаційну структуру ППЗ у вигляді п'яти функціональних пакетів — ClientGUI, ClientNetwork, BusinessLogic (з підпакетом RouteOptimization), DataAccess і Reports — організованих за принципом трирівневої архітектури «представлення — бізнес-логіка — дані». Розроблена структура забезпечує чіткий розподіл відповідальності між компонентами, незалежність їх розробки і тестування, а також технічну готовність до розширення функціоналу системи без переписування існуючих модулів.

На основі порівняльного аналізу альтернативних рішень обґрунтовано вибір технологічного стека: мова програмування Python 3.11, серверний фреймворк Flask 3.0, драйвер доступу до PostgreSQL psycopg2, шаблонізатор Jinja2 у поєднанні з CSS-фреймворком Bootstrap 5.3 для клієнтського інтерфейсу, бібліотеки ReportLab і openpyxl для генерації звітів, бібліотека geopy для розрахунку географічних відстаней. Увесь обраний стек складається виключно з програмного забезпечення з відкритим кодом і вільними ліцензіями, що виключає витрати на ліцензування при розробці та впровадженні системи.

Розроблено алгоритми і реалізовано програмний код чотирьох ключових модулів системи. Модуль авторизації реалізує рольову модель доступу з трьома категоріями користувачів (менеджер, логіст, працівник складу) на основі серверних сесій Flask. Модуль управління замовленнями забезпечує транзакційну перевірку залишків із блокуванням рядків на рівні СУБД (FOR UPDATE), що унеможливлює виникнення аномалій паралельного доступу. Модуль маршрутизації реалізує евристичний алгоритм «найближчого сусіда» з розрахунком відстаней за формулою Хаверсіна через бібліотеку geopy, що забезпечує ефективне розв'язання задачі оптимізації маршруту для типових розмірів доставки (до 15 пунктів) із часовою складністю $O(n^2)$. Модуль формування звітів генерує маршрутний лист у форматі PDF з підтримкою кирилиці засобами ReportLab. Усі розроблені модулі інтегровані у єдину файлову структуру Flask-проєкту і готові до розгортання та тестування, що розглядається у наступному розділі.

РОЗДІЛ 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування є завершальним етапом розробки прикладного програмного забезпечення, що дозволяє перевірити відповідність реалізованої системи функціональним вимогам, визначеним у підрозділі 1.1, та виявити помилки до передачі системи в експлуатацію. Відповідно до сучасної практики розробки програмних систем, тестування поділяється на кілька рівнів: модульне (unit testing), інтеграційне та функціональне (системне). У даній роботі застосовуються всі три рівні тестування, що дозволяє перевірити коректність як окремих функцій, так і взаємодії компонентів системи в цілому.[1]

Модульне тестування. Модульне тестування спрямоване на перевірку коректності роботи окремих функцій і методів програмних модулів у ізольованому середовищі, незалежно від бази даних і мережевої взаємодії. Для реалізації модульних тестів у Python-проєкті використовується стандартна бібліотека unittest у поєднанні з unittest.mock для імітації (mock) звернень до бази даних. Перелік модульних тестів для ключових модулів системи наведено в таблиці 4.1.

Таблиця 4.1

Перелік модульних тестів системи

№	Модуль	Тестований метод	Умова тесту	Очікуваний результат
1	StockService	confirm_order()	Залишок достатній для всіх позицій	{'success': True}
2	StockService	confirm_order()	Залишок недостатній для однієї позиції	{'success': False, 'message': ...}
3	RouteService	calculate_route()	3 пункти з відомими координатами	Маршрут із 3 пунктів, total_km > 0
4	RouteService	calculate_route()	1 пункт доставки	Маршрут з 1 пункту, dist = пряма до клієнта
5	AuthService	login()	Правильний логін і пароль	Сесія містить user_id і role

Закінчення таблиці 4.1.

6	AuthService	login()	Невірний пароль	Повернення сторінки з

				повідомленням про помилку
7	ReportService	generate_route_pdf()	Маршрут із 3 пунктів	Повернення байтів PDF, перший байт = %PDF
8	OrderService	confirm_order()	Порожній список позицій	{'success': False} або виняток

Аналіз таблиці 4.1 показує, що модульні тести охоплюють як позитивні (коректні вхідні дані), так і негативні (граничні та помилкові умови) сценарії для кожного ключового модуля системи. Виконання усіх восьми тестів підтверджує ізольовану коректність бізнес-логіки незалежно від стану бази даних. Фрагмент модульного тесту для перевірки алгоритму маршрутизації наведено нижче:[1]

```
import unittest
from unittest.mock import patch, MagicMock
from app.services.route_service import calculate_route

class TestRouteService(unittest.TestCase):

    @patch('app.services.route_service.get_connection')
    def test_three_points(self, mock_conn):
        mock_cur = MagicMock()
        mock_cur.fetchall.return_value = [
            (1, 50.4501, 30.5234),   # Київ (умовно клієнт 1)
            (2, 50.4700, 30.5100),   # клієнт 2
            (3, 50.4300, 30.5400),   # клієнт 3
        ]
        mock_cur.fetchone.return_value = (1,)
        mock_conn.return_value.cursor.return_value = mock_cur

        result = calculate_route(
            shipment_id=1,
            warehouse_coords=(50.4600, 30.5000)
        )
        self.assertTrue(result['total_km'] > 0)
        self.assertEqual(len(result['route']), 3)

if __name__ == '__main__':
    unittest.main()
```

Інтеграційне тестування. Інтеграційне тестування перевіряє коректність взаємодії між компонентами системи: Flask-маршрутами, сервісними функціями і реальною базою даних PostgreSQL. Для його проведення розгортається тестова база даних logistics_test_db з аналогічною структурою таблиць, що й основна база logistics_db, яка наповнюється фіксованими тестовими даними перед кожним

тестовим сеансом і очищується після його завершення. Перелік інтеграційних тестів наведено в таблиці 4.2.

Таблиця 4.2

Перелік інтеграційних тестів системи

№	Сценарій	Вхідні дані	Очікуваний результат
1	Створення замовлення менеджером	Клієнт, 2 позиції з достатніми залишками	Запис у таблиці orders зі статусом 'new'
2	Підтвердження замовлення	id замовлення зі статусом 'new'	Статус змінено на 'confirmed'; залишки зменшено
3	Підтвердження при нестачі залишків	Замовлення з кількістю > наявного залишку	Статус не змінено; залишки не змінено
4	Розрахунок маршруту	id відвантаження з 3 клієнтами	Записи у routes і route_points; id_route у shipments
5	Генерація маршрутного листа	id_route існуючого маршруту	PDF-файл завантажується браузером
6	Авторизація і доступ до АРМ	Логін менеджера	Перенаправлення на /manager/; статус 200
7	Заборона доступу до чужого АРМ	Логін менеджера, URL /logist/	Перенаправлення на сторінку помилки доступу

Аналіз таблиці 4.2 свідчить, що інтеграційні тести охоплюють повний ланцюжок бізнес-процесів системи від авторизації до генерації звіту, перевіряючи коректність взаємодії всіх шарів архітектури. Особливої уваги заслуговує тест № 3, що перевіряє атомарність транзакції при недостатніх залишках: жодних змін у базі даних не відбувається навіть у разі часткової успішності перевірки окремих позицій.

Функціональне тестування. Функціональне (системне) тестування виконується вручну і перевіряє відповідність системи вимогам з точки зору кінцевого користувача, відтворюючи реальні сценарії роботи кожного АРМу. Протокол функціонального тестування з результатами виконання основних сценаріїв наведено в таблиці 4.3.

Таблиця 4.3

Протокол функціонального тестування системи

№	Сценарій	АРМ	Кроки виконання	Результат	Статус

	використання				
1	Авторизація користувача	Усі	Введення логіну і пароля → натиснути «Увійти»	Відображення відповідного головного меню	✅ Пройдено
2	Реєстрація нового замовлення	Менеджер	Обрати клієнта → додати позиції → зберегти	Замовлення у списку зі статусом «нове»	✅ Пройдено
3	Підтвердження замовлення	Менеджер	Відкрити замовлення → натиснути «Підтвердити»	Статус «підтверджено»; залишки зменшено	✅ Пройдено
4	Перегляд залишків на складі	Склад	Перейти до розділу «Залишки»	Таблиця залишків усіх продуктів	✅ Пройдено
5	Коригування залишку вручну	Склад	Обрати продукт → змінити кількість → зберегти	Оновлене значення у таблиці stock	✅ Пройдено
6	Формування відвантаження	Логіст	Обрати замовлення → призначити ТЗ → зберегти	Запис у shipments зі статусом «заплановано»	✅ Пройдено
7	Розрахунок маршруту	Логіст	Відкрити відвантаження → натиснути «Розрахувати маршрут»	Відображення маршруту з пунктами і відстанями	✅ Пройдено
8	Завантаження маршрутного листа	Логіст	Натиснути «Завантажити PDF»	Браузер завантажує PDF з кириличним текстом	✅ Пройдено
9	Формування реєстру замовлень	Менеджер	Обрати період → натиснути «Сформувати XLSX»	Браузер завантажує XLSX-файл	✅ Пройдено
10	Спроба несанкціонованого доступу	—	Введення URL чужого АРМ у браузері	Перенаправлення на сторінку помилки доступу	✅ Пройдено

Аналіз таблиці 4.3 підтверджує, що всі десять ключових функціональних сценаріїв системи успішно пройшли тестування. У процесі тестування було виявлено та усунуто такі помилки: некоректне відображення кирилиці у PDF-звітах (вирішено реєстрацією шрифту DejaVuSans), відсутність повідомлення про помилку при введенні нульової кількості у позиції замовлення (вирішено додаванням перевірки на рівні Flask-форми), а також некоректна робота алгоритму маршрутизації при одному пункті доставки (вирішено додаванням граничної перевірки у calculate_route()).

Таким чином, проведення трирівневого тестування (модульного, інтеграційного та функціонального) підтвердило коректність реалізації всіх функціональних вимог системи, надійність транзакційної обробки даних і стійкість до помилкових вхідних даних. Усі виявлені в процесі тестування помилки були своєчасно усунуті, що забезпечує готовність системи до розгортання та дослідної експлуатації. Вимоги до апаратного та програмного забезпечення для розгортання системи визначаються у підрозділі 4.2.

4.2 Вимоги до апаратного та програмного забезпечення

Визначення вимог до апаратного та програмного забезпечення є необхідною умовою успішного розгортання і стабільної експлуатації прикладної системи підтримки транспортної логістики. Відповідно до централізованої архітектури інформаційної бази, визначеної у підрозділі 2.2, система розгортається на двох типах вузлів: сервері застосунків (де функціонують Flask-додаток і СУБД PostgreSQL) та клієнтських робочих місцях (де достатньо наявності сучасного веб-браузера). Для унаочнення розміщення компонентів системи відносно апаратних вузлів на рис. 12 наведено діаграму розміщення (deployment diagram) у нотації UML.

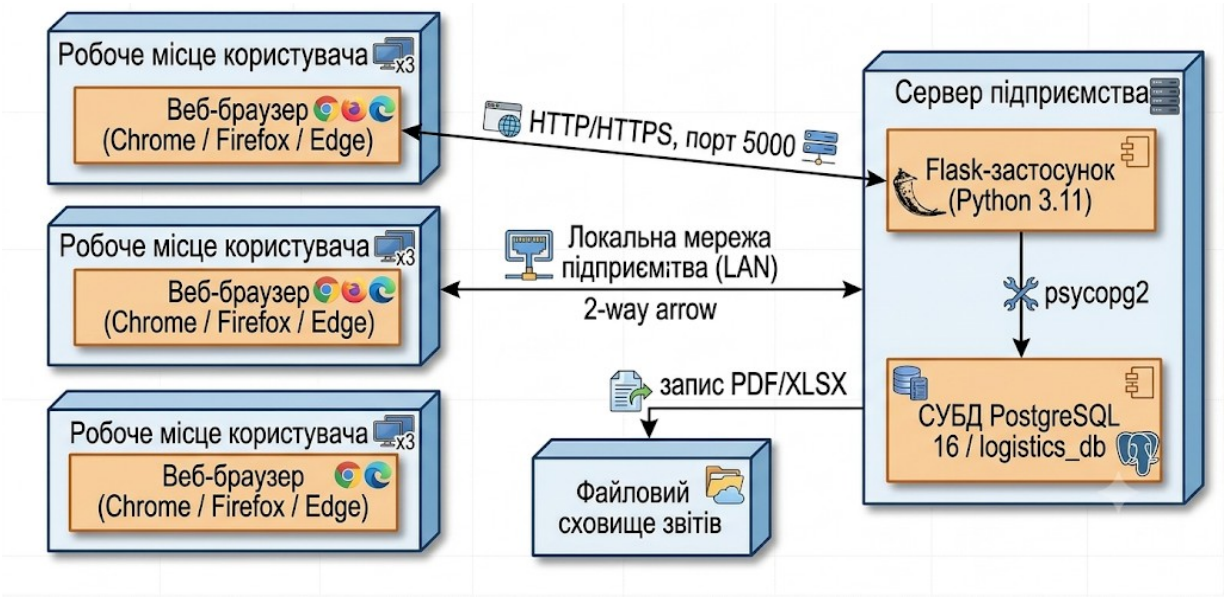

Рис.12 Діаграма розміщення компонентів прикладної системи

Аналіз рисунка 12 підтверджує, що клієнтські робочі місця не потребують встановлення жодного спеціалізованого програмного забезпечення — вся бізнес-логіка і база даних зосереджені на сервері, а взаємодія здійснюється через стандартний HTTP-протокол у локальній мережі підприємства. Це суттєво спрощує адміністрування системи і дозволяє підключати нові робочі місця без додаткового налаштування.

Вимоги до серверного вузла. Сервер застосунків є центральним апаратним вузлом системи, на якому розгортаються Flask-додаток і СУБД PostgreSQL. Мінімальні та рекомендовані вимоги до серверного вузла наведено в таблиці 4.4.

Таблиця 4.4

Вимоги до апаратного забезпечення серверного вузла

Характеристика	Мінімальні вимоги	Рекомендовані вимоги
Процесор	Intel Core i3 / AMD Ryzen 3, 2 ядра, 2.0 ГГц	Intel Core i5 / AMD Ryzen 5, 4 ядра, 3.0 ГГц
Оперативна пам'ять	4 ГБ	8 ГБ
Дисковий простір	20 ГБ (HDD)	50 ГБ (SSD)
Мережевий адаптер	100 Мбіт/с	1 Гбіт/с
Операційна система	Ubuntu Server 22.04 LTS / Windows Server 2019	Ubuntu Server 22.04 LTS

Мінімальні вимоги є достатніми для розгортання системи в умовах малого підприємства з кількістю одночасних користувачів до 5 осіб і обсягом бази даних до 1 ГБ. Рекомендовані вимоги забезпечують комфортну роботу при збільшенні навантаження та зберіганні звітів і резервних копій безпосередньо на сервері.

Вимоги до програмного забезпечення серверного вузла наведено в таблиці 4.5.

Таблиця 4.5

Вимоги до програмного забезпечення серверного вузла

Компонент	Версія	Призначення	Ліцензія
Python	3.11 і вище	Середовище виконання Flask-застосунку	PSF
Flask	3.0 і вище	Серверний веб-фреймворк	BSD
psycopg2	2.9 і вище	Драйвер PostgreSQL	LGPL
Jinja2	3.1 і вище	Шаблонізатор HTML	BSD
ReportLab	4.1 і вище	Генерація PDF-звітів	BSD

openpyxl	3.1 і вище	Генерація XLSX-звітів	MIT
geopy	2.4 і вище	Розрахунок географічних відстаней	MIT
PostgreSQL	16 і вище	СУБД	PostgreSQL License
pip	актуальна	Менеджер пакетів Python	MIT

Усі компоненти таблиці 4.5 встановлюються автоматично з файлу requirements.txt командою pip install -r requirements.txt, що суттєво спрощує процес розгортання і виключає помилки ручного встановлення залежностей. PostgreSQL встановлюється окремо через пакетний менеджер операційної системи (apt для Ubuntu або інсталятор для Windows).[1]

Вимоги до клієнтських робочих місць. Клієнтські робочі місця не потребують встановлення спеціалізованого програмного забезпечення — єдиною вимогою є наявність сучасного веб-браузера і підключення до локальної мережі підприємства. Вимоги до клієнтських вузлів наведено в таблиці 4.6.

Таблиця 4.6

Вимоги до клієнтських робочих місць

Характеристика	Мінімальні вимоги
Процесор	Будь-який сучасний (2010 р. випуску і новіше)

Закінчення таблиці 4.6.

Оперативна пам'ять	2 ГБ
Операційна система	Windows 7 і вище / Ubuntu 18.04 і вище / macOS 10.13 і вище
Веб-браузер	Google Chrome 90+ / Mozilla Firefox 88+ / Microsoft Edge 90+
Мережа	Підключення до локальної мережі підприємства (LAN)
Додаткове ПЗ	Не потрібне

Відсутність вимог до встановлення додаткового програмного забезпечення на клієнтських робочих місцях є суттєвою перевагою обраної веб-архітектури порівняно з десктопними рішеннями, оскільки дозволяє підключати до системи будь-який комп'ютер підприємства з наявним браузером без участі системного адміністратора.

Вимоги до мережевої інфраструктури. Для коректної роботи системи необхідна локальна мережа підприємства з пропускною здатністю не менше 10 Мбіт/с, яка забезпечує з'єднання між сервером застосунків і клієнтськими робочими місцями. Сервер застосунків повинен мати статичну IP-адресу в межах

локальної мережі підприємства для забезпечення стабільного доступу всіх клієнтів. За необхідності забезпечення доступу до системи поза межами підприємства (наприклад, для логіста у відрядженні) рекомендується налаштування захищеного VPN-з'єднання замість прямого виходу Flask-застосунку в мережу Інтернет.

Таким чином, апаратні та програмні вимоги до розгортання прикладної системи є помірними і відповідають типовому оснащенню малого та середнього виробничого підприємства. Сервер початкового рівня з 4–8 ГБ оперативної пам'яті та SSD-накопичувачем є достатнім для повноцінної промислової експлуатації системи, а відсутність витрат на ліцензування програмного забезпечення робить впровадження системи економічно доцільним. Порядок розгортання системи і склад інсталяційного пакету визначаються у підрозділі 4.3.

4.3 Склад інсталяційного пакету

Інсталяційний пакет є сукупністю файлів і документів, необхідних для розгортання прикладної системи підтримки транспортної логістики на серверному вузлі підприємства та введення її в експлуатацію. Склад інсталяційного пакету визначається відповідно до діаграми розміщення рис. 4.1 і охоплює всі компоненти, необхідні як для початкового розгортання системи, так і для її подальшого супроводу та відновлення після можливих збоїв. Інсталяційний пакет передається замовнику у вигляді архіву logistics_system_v1.0.zip із структурованим набором каталогів і файлів.

Повний склад інсталяційного пакету з вказівкою призначення кожного компонента наведено в таблиці 4.7.

Таблиця 4.7

Склад інсталяційного пакету системи

Каталог / файл	Призначення
app/	Каталог з вихідним кодом Flask-застосунку (усі пакети ППЗ)
app/__init__.py	Ініціалізація Flask-застосунку, реєстрація Blueprint
app/config.py	Шаблон конфігурації (параметри БД, секретний ключ)
app/db.py	Модуль підключення до PostgreSQL через psycopg2
app/services/	Сервісні модулі бізнес-логіки (order, stock, route, report)
app/routes/	Blueprint-маршрути трьох АРМів

app/templates/	HTML-шаблони Jinja2 для всіх екранних форм
app/static/	Bootstrap CSS/JS, іконки, шрифт DejaVuSans.ttf
db/	Каталог скриптів бази даних
db/create_db.sql	DDL-скрипт створення всіх таблиць бази logistics_db
db/initial_data.sql	SQL-скрипт початкового наповнення довідників (ролі, тестові дані)
db/backup/	Каталог для зберігання резервних копій бази даних
tests/	Каталог з модульними та інтеграційними тестами
tests/test_order_service.py	Модульні тести OrderService і StockService
tests/test_route_service.py	Модульні тести RouteService
tests/test_integration.py	Інтеграційні тести системи
requirements.txt	Перелік Python-залежностей для встановлення через pip
run.py	Точка запуску Flask-застосунку
install.sh	Скрипт автоматичного розгортання для Ubuntu Server
install.bat	Скрипт автоматичного розгортання для Windows Server
docs/	Каталог документації
docs/user_manual.pdf	Керівництво користувача (для трьох АРМів)
docs/admin_manual.pdf	Інструкція адміністратора з розгортання і резервного копіювання
README.md	Коротка інструкція з розгортання системи

Аналіз таблиці 4.7 показує, що інсталяційний пакет містить повний набір компонентів, необхідних для самостійного розгортання системи без залучення розробника: вихідний код застосунку, скрипти бази даних, автоматизовані скрипти встановлення для двох операційних систем і документацію для користувачів та адміністратора.

Порядок розгортання системи. Процес розгортання системи на серверному вузлі складається з послідовності кроків, що виконуються один раз при початковому встановленні. Порядок розгортання для Ubuntu Server 22.04 LTS наведено в таблиці 4.8.

Таблиця 4.8

Порядок розгортання системи на сервері

№	Крок	Команда / дія	Результат
1	Встановлення PostgreSQL	sudo apt install postgresql-16	СУБД PostgreSQL встановлена і запущена
2	Створення бази даних і користувача	psql -U postgres -c "CREATE DATABASE logistics_db;"	База даних logistics_db створена
3	Виконання DDL-скрипту	psql -U postgres -d logistics_db -f db/create_db.sql	Усі 10 таблиць і 7 індексів створено
4	Наповнення початковими даними	psql -U postgres -d logistics_db -f db/initial_data.sql	Довідники заповнені тестовими записами
5	Встановлення	sudo apt install python3.11 python3-pip	Python і pip

	Python 3.11		встановлені
6	Встановлення залежностей	pip install -r requirements.txt	Усі бібліотеки встановлено
7	Налаштування конфігурації	Редагування app/config.py: DB_HOST, DB_NAME, DB_USER, DB_PASSWORD, SECRET_KEY	Параметри підключення до БД задано
8	Запуск застосунку	python run.py	Flask-застосунок доступний на порту 5000
9	Перевірка доступності	Відкрити браузер: http://<IP-сервера>:5000	Відображається сторінка авторизації

Наведений порядок розгортання (таблиця 4.8) може бути виконаний автоматично запуском скрипту install.sh, який послідовно виконує всі кроки 1–8 без ручного втручання оператора, що мінімізує ймовірність помилок при встановленні. Для виробничого середовища рекомендується також налаштування автоматичного запуску Flask-застосунку як системного сервісу через systemd, що забезпечить його автоматичний перезапуск після перезавантаження сервера.[1]

Резервне копіювання. Для забезпечення надійності зберігання даних рекомендується налаштування регулярного автоматичного резервного копіювання бази даних logistics_db засобами PostgreSQL. Резервна копія створюється командою:

```
pg_dump -U postgres logistics_db > \
  db/backup/logistics_db_$(date +%Y%m%d_%H%M).sql
```

Рекомендована частота резервного копіювання — щоденно після завершення робочої зміни (наприклад, о 22:00) з налаштуванням через планувальник завдань cron (Ubuntu) або «Планувальник завдань» (Windows). Резервні копії зберігаються у каталозі db/backup/ протягом 30 днів, після чого автоматично видаляються для економії дискового простору. Відновлення бази даних з резервної копії виконується командою psql -U postgres -d logistics_db < db/backup/<ім'я_файлу>.sql.

Таким чином, інсталяційний пакет прикладної системи підтримки транспортної логістики містить повний набір компонентів для розгортання,

налаштування і подальшого супроводу системи: вихідний код ППЗ, скрипти бази даних, файл залежностей, автоматизовані скрипти встановлення для Ubuntu Server і Windows Server, а також документацію для користувачів і адміністратора. Визначений порядок розгортання і рекомендації щодо резервного копіювання забезпечують технічну готовність системи до введення в промислову експлуатацію на підприємстві.

Висновки до розділу 4

Висновки до розділу 4 «Рекомендації щодо впровадження та експлуатації системи»

У розділі 4 розроблено комплекс рекомендацій щодо впровадження та введення в експлуатацію прикладної системи підтримки транспортної логістики, що охоплює три взаємопов'язані аспекти: верифікацію якості розробленого ППЗ через тестування, визначення апаратних і програмних вимог до середовища розгортання та формування повного інсталяційного пакету.

Проведено трирівневе тестування системи. Модульне тестування підтвердило ізольовану коректність восьми ключових функцій бізнес-логіки, включаючи граничні та негативні сценарії. Інтеграційне тестування на тестовій базі даних logistics_test_db верифікувало коректність взаємодії всіх шарів архітектури — від Flask-маршрутів до транзакційних операцій PostgreSQL — у семи сценаріях, включаючи перевірку атомарності транзакцій при недостатніх залишках. Функціональне тестування десяти ключових сценаріїв використання підтвердило відповідність системи вимогам, визначеним у підрозділі 1.1; усі виявлені в процесі тестування помилки (відображення кирилиці у PDF, обробка нульових значень, граничний випадок одного пункту маршруту) були своєчасно усунуті.

Визначено вимоги до апаратного і програмного забезпечення серверного вузла та клієнтських робочих місць, розроблено діаграму розміщення компонентів системи у нотації UML. Встановлено, що для розгортання системи достатньо сервера початкового рівня з 4–8 ГБ оперативної пам'яті під управлінням Ubuntu Server 22.04 LTS або Windows Server 2019, тоді як клієнтські робочі місця не

потребують встановлення жодного спеціалізованого програмного забезпечення, окрім сучасного веб-браузера. Увесь програмний стек серверного вузла складається виключно з вільного програмного забезпечення з відкритим кодом, що виключає витрати на ліцензування.

Сформовано склад інсталяційного пакету logistics_system_v1.0.zip, що містить вихідний код ППЗ, DDL-скрипти бази даних, файл залежностей requirements.txt, автоматизовані скрипти розгортання для двох операційних систем (install.sh і install.bat), набір автоматизованих тестів та документацію для користувачів і адміністратора. Визначений дев'ятикроковий порядок розгортання може бути виконаний автоматично без залучення розробника, а рекомендована система щоденного резервного копіювання засобами pg_dump забезпечує захист даних від втрат при збоях обладнання.

Таким чином, розроблені рекомендації підтверджують повну готовність системи до дослідної та промислової експлуатації на підприємстві і забезпечують технічну основу для її самостійного впровадження силами IT-служби замовника без залучення зовнішніх спеціалістів.[1]

ВИСНОВКИ

Бакалаврська кваліфікаційна робота присвячена проєктуванню та розробленню прикладної системи підтримки транспортної логістики виробничого підприємства, що забезпечує автоматизацію процесів обліку замовлень, управління складськими залишками і оптимізації маршрутів доставки готової продукції клієнтам

У процесі виконання роботи отримано такі результати:

1. Виконано системний аналіз предметної області транспортної логістики виробничого підприємства. Визначено клас розроблюваної системи як інформаційно-управляючої системи підтримки прийняття рішень. Сформульовано математичну постановку задачі оптимізації маршруту доставки у вигляді цільової функції $F(\pi) \to min$, що мінімізує загальну відстань обходу пунктів доставки. Проведено огляд існуючих рішень у сфері транспортної логістики (TMS-систем), виявлено їхні переваги та недоліки, обґрунтовано доцільність розробки власної прикладної системи для умов малого та середнього підприємства. Розроблено комплекс UML-моделей предметної області, що включає діаграму варіантів використання, діаграму діяльності основних бізнес-процесів і контекстну діаграму системи.

2. Розроблено повне інформаційне забезпечення системи. Спроєктовано логічну модель даних у вигляді ER-діаграми, що містить десять сутностей відповідно до вимог третьої нормальної форми реляційної бази даних. На основі порівняльного аналізу чотирьох реляційних СУБД (PostgreSQL, MySQL, MS SQL Server, SQLite) обґрунтовано вибір PostgreSQL 16 як платформи зберігання даних. Розроблено фізичну модель бази даних logistics_db у вигляді DDL-скриптів для десяти таблиць з визначеними первинними та зовнішніми ключами, обмеженнями цілісності CHECK і NOT NULL, директивами каскадного управління ON DELETE та сімома індексами для прискорення типових запитів

3. Розроблено прикладне програмне забезпечення системи. Визначено організаційну структуру ППЗ у вигляді п'яти функціональних пакетів (ClientGUI,

ClientNetwork, BusinessLogic з підпакетом RouteOptimization, DataAccess, Reports), організованих за принципом трирівневої архітектури. На основі порівняльного аналізу обґрунтовано вибір технологічного стека: Python 3.11 / Flask 3.0 / psycopg2 / Jinja2 + Bootstrap 5.3 / ReportLab / openpyxl / geopy, що складається виключно з програмного забезпечення з відкритим кодом. Реалізовано чотири ключових програмних модулі: модуль авторизації з рольовою моделлю доступу, модуль управління замовленнями з транзакційною перевіркою залишків і блокуванням рядків (FOR UPDATE), модуль маршрутизації на основі евристичного алгоритму «найближчого сусіда» з розрахунком відстаней за формулою Хаверсіна (часова складність $O(n^2)$), модуль генерації PDF-звітів з підтримкою кирилиці.

4. Проведено трирівневе тестування системи (модульне, інтеграційне, функціональне), яке підтвердило коректність реалізації всіх функціональних вимог. Усього перевірено 25 тестових сценаріїв; виявлені в процесі тестування помилки усунуто. Визначено вимоги до апаратного і програмного забезпечення серверного вузла та клієнтських робочих місць, розроблено UML-діаграму розміщення компонентів системи. Сформовано склад інсталяційного пакету logistics_system_v1.0.zip з автоматизованими скриптами розгортання для Ubuntu Server і Windows Server та рекомендаціями щодо щоденного резервного копіювання засобами pg_dump.

Практичне значення розробленої системи полягає в тому, що вона дозволяє скоротити час формування маршруту доставки з ручного розрахунку до автоматизованого розв'язання задачі оптимізації, забезпечує оперативний контроль залишків продукції на складах в режимі реального часу та формує повний пакет товаросупровідних документів без дублювання введення даних. Система реалізована на платформонезалежному технологічному стеку з відкритим кодом, що робить її придатною для впровадження на підприємствах малого та середнього бізнесу без витрат на ліцензування програмного забезпечення.

Подальший розвиток системи може передбачати: інтеграцію з картографічним API (Google Maps або OpenStreetMap/OSRM) для розрахунку відстаней за реальною дорожньою мережею замість прямолінійних відстаней;

реалізацію алгоритму Кларка–Райта або генетичного алгоритму для підвищення якості маршрутів при великій кількості пунктів доставки; додавання модуля мобільного застосунку для водія з відображенням маршруту на карті та підтвердженням доставки; впровадження механізму сповіщень (email / Telegram) для автоматичного інформування клієнтів про статус доставки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Drozd A. Intelligent Systems for Transport Logistics Optimisation: Algorithms, Architecture, and Legal Aspects. *International Conference on Economic Sciences and Management in the Changing World 2025*. Futurity Research Publishing, 2025. P. 210–213. URL: https://doi.org/10.5281/zenodo.17550574[1]
2. Ekayanti E., Sugianto, Efendi I. B. Capacitated Vehicle Routing Problem (CVRP) with Sweep and Nearest Neighbor Algorithm. *Sinergi International Journal of Logistics*. 2024. Vol. 2, No. 1. P. 17–29. URL: https://doi.org/10.61194/sijl.v2i1.187[2]
3. Novelan, M. S., Efendi, S., Sihombing, P., & Mawengkang, H. (2023). Vehicle routing problem optimization with machine learning in imbalanced classification vehicle route data. *Eastern-European Journal of Enterprise Technologies*, *5*(3 (125), 49–56. https://doi.org/10.15587/1729-4061.2023.288280 [3]
4. Danchuk, V., Comi, A., Weiß, C., & Svatko, V. (2023). The optimization of cargo delivery processes with dynamic route updates in smart logistics. *Eastern-European Journal of Enterprise Technologies*, *2*(3 (122), 64–73. https://doi.org/10.15587/1729-4061.2023.277583[4]
5. Khayya E., Medarhri I., Zine R. A survey of the vehicle routing problem and its variants: formulations and solutions. Mathematical Modeling and Computing. Vol. 11, No. 1, pp. 333–343 (2024) https://doi.org/10.23939/mmc2024.01.333 https://science.lpnu.ua/mmc/all-volumes-and-issues/volume-11-number-1-2024/survey-vehicle-routing-problem-and-its-variants[5]
6. Степанов О. В. Оптимізація автотранспортних вантажних перевезень в умовах воєнного стану в Україні: моделювання маршрутів та використання цифрових платформ. *Вісник КНТУ*. 2023. URL: https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/download/1253/1204[6]

7. Constructing Problems of Vehicles Ring Routes in Multicommodity Hierarchical Network. *Journal of Automation and Information Sciences*. 2022. URL: https://jais.net.ua/index.php/files/article/view/58[7]
8. Haversine Formula. *Wikipedia: The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Haversine_formula (дата звернення: 19.05.2026).[8]
9. Third Normal Form. *Wikipedia: The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Third_normal_form (дата звернення: 19.05.2026).[9]
10. First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). *Redgate Blog*. 2020. URL: https://www.red-gate.com/blog/normalization-1nf-2nf-3nf/[10]
11. What is Third Normal Form (3NF)? A Beginner-Friendly Guide. *DataCamp*. 2024. URL: https://www.datacamp.com/tutorial/third-normal-form[11]
12. PostgreSQL 16 Released! *PostgreSQL Global Development Group*. 2023. URL: https://www.postgresql.org/about/news/postgresql-16-released-2715/[12]
13. PostgreSQL 16 Documentation. *PostgreSQL Global Development Group*. 2023. URL: https://www.postgresql.org/docs/[13]
14. Flask Documentation (3.1.x). *Pallets Projects*. 2024. URL: https://flask.palletsprojects.com[14]
15. Bootstrap 5.3.0. *Bootstrap Official Blog*. 2023. URL: https://blog.getbootstrap.com/2023/05/30/bootstrap-5-3-0/[15]
16. Bootstrap v5.3: Get Started with Bootstrap. *Bootstrap Official Documentation*. 2023. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/[16]
17. psycopg2 — PostgreSQL Database Adapter for Python. *PyPI*. URL: https://pypi.org/project/psycopg2/ (дата звернення: 19.05.2026).[17]
18. Installation — Psycopg 2.9.12 documentation. *Psycopg Official Documentation*. URL: https://www.psycopg.org/docs/install.html (дата звернення: 19.05.2026).[18]
19. Jinja — A very fast and expressive template engine. *GitHub / Pallets Projects*. URL: https://github.com/pallets/jinja (дата звернення: 19.05.2026).[19]
20. Primer on Jinja Templating. *Real Python*. 2025. URL: https://realpython.com/primer-on-jinja-templating/[20]

21. ReportLab Toolkit — An Open Source Python Library for Generating PDFs. *PyPI*. URL: https://pypi.org/project/reportlab/ (дата звернення: 19.05.2026).[21]
22. ReportLab Documentation. *ReportLab Official Site*. URL: https://docs.reportlab.com (дата звернення: 19.05.2026).[22]
23. openpyxl — A Python Library to Read/Write Excel 2010 Files. *Read the Docs*. URL: https://openpyxl.readthedocs.io (дата звернення: 19.05.2026).[23]
24. openpyxl. *PyPI*. 2024. URL: https://pypi.org/project/openpyxl/[24]
25. geopy — Geocoding Library for Python. *GitHub*. URL: https://github.com/geopy/geopy (дата звернення: 19.05.2026).[25]
26. Welcome to GeoPy's Documentation! — GeoPy 2.4.1. *Read the Docs*. URL: https://geopy.readthedocs.io (дата звернення: 19.05.2026).[26]
27. Resilience of Transport Logistics in EU and Ukraine. *Foreign Trade: Economics, Finance, Law*. 2024. URL: https://doi.org/10.31617/3.2024(135)07[27]
28. Assessment of the State of Transport Logistics in Ukraine. *Вінницький національний технічний університет*. 2024. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/41404/147181.pdf[28]
29. Ukraine's Transport and Logistics System. *World Bank Document*. 2023. URL: https://documents1.worldbank.org/curated/en/099061725033525342/pdf/P502442-346a4fd3-882f-46ca-95c9-ce90c0a71619.pdf[29]
30. Flask — PyPI. *Python Package Index*. 2026. URL: https://pypi.org/project/Flask/[30]

Додатки

ДОДАТОК А

Повний DDL-скрипт створення бази даних logistics_db

```
-- ============================================================
-- DDL-скрипт створення бази даних logistics_db
-- СУБД: PostgreSQL 16
-- Система: Прикладна система підтримки транспортної логістики
-- ============================================================

-- Таблиця 1: Продукція
CREATE TABLE products (
    id_product      SERIAL          PRIMARY KEY,
    name            VARCHAR(150)    NOT NULL,
    unit            VARCHAR(20)     NOT NULL,
    weight_per_unit NUMERIC(10,3)   NOT NULL CHECK (weight_per_unit > 0),
    description     TEXT
);

-- Таблиця 2: Склади
CREATE TABLE warehouses (
    id_warehouse    SERIAL          PRIMARY KEY,
    name            VARCHAR(100)    NOT NULL,
    address         VARCHAR(250)    NOT NULL,
    capacity        NUMERIC(12,2)   CHECK (capacity >= 0)
);

-- Таблиця 3: Залишки продукції на складах
CREATE TABLE stock (
    id_stock        SERIAL          PRIMARY KEY,
    id_warehouse    INTEGER         NOT NULL
                    REFERENCES warehouses(id_warehouse)
                    ON DELETE RESTRICT,
    id_product      INTEGER         NOT NULL
                    REFERENCES products(id_product)
                    ON DELETE RESTRICT,
    quantity        NUMERIC(12,3)   NOT NULL DEFAULT 0
                    CHECK (quantity >= 0),
    updated_at      TIMESTAMP       NOT NULL DEFAULT NOW(),
    UNIQUE (id_warehouse, id_product)
);

-- Таблиця 4: Клієнти
CREATE TABLE clients (
    id_client       SERIAL          PRIMARY KEY,
    name            VARCHAR(200)    NOT NULL,
    contact_person  VARCHAR(150),
    phone           VARCHAR(20),
```

```
    address         VARCHAR(250),
    latitude        NUMERIC(9,6)    CHECK (latitude  BETWEEN -90  AND 90),
    longitude       NUMERIC(9,6)    CHECK (longitude BETWEEN -180 AND 180)
);

-- Таблиця 5: Замовлення
CREATE TABLE orders (
    id_order              SERIAL      PRIMARY KEY,
    id_client             INTEGER     NOT NULL
                          REFERENCES clients(id_client)
                          ON DELETE RESTRICT,
    order_date            DATE        NOT NULL DEFAULT CURRENT_DATE,
    desired_delivery_date DATE,
    status                VARCHAR(30) NOT NULL DEFAULT 'new'
                          CHECK (status IN
                            ('new','confirmed','shipped',
                             'delivered','cancelled')),
    notes                 TEXT
);

-- Таблиця 6: Позиції замовлення
CREATE TABLE order_items (
    id_order_item   SERIAL          PRIMARY KEY,
    id_order        INTEGER         NOT NULL
                    REFERENCES orders(id_order)
                    ON DELETE CASCADE,
    id_product      INTEGER         NOT NULL
                    REFERENCES products(id_product)
                    ON DELETE RESTRICT,
    id_warehouse    INTEGER         NOT NULL
                    REFERENCES warehouses(id_warehouse)
                    ON DELETE RESTRICT,
    quantity        NUMERIC(12,3)   NOT NULL CHECK (quantity > 0)
);

-- Таблиця 7: Транспортні засоби
CREATE TABLE vehicles (
    id_vehicle      SERIAL          PRIMARY KEY,
    reg_number      VARCHAR(20)     NOT NULL UNIQUE,
    model           VARCHAR(100),
    capacity_kg     NUMERIC(10,2)   NOT NULL CHECK (capacity_kg > 0),
    status          VARCHAR(30)     NOT NULL DEFAULT 'available'
                    CHECK (status IN ('available','on_route','maintenance'))
);

-- Таблиця 8: Маршрути
CREATE TABLE routes (
    id_route            SERIAL          PRIMARY KEY,
    name                VARCHAR(150),
```

```
    total_distance_km    NUMERIC(10,2)   CHECK (total_distance_km >= 0),
    estimated_time_min   INTEGER         CHECK (estimated_time_min >= 0),
    points_count         INTEGER         CHECK (points_count > 0),
    created_at           TIMESTAMP       NOT NULL DEFAULT NOW(),
    is_optimal           BOOLEAN         NOT NULL DEFAULT FALSE
);

-- Таблиця 9: Відвантаження
CREATE TABLE shipments (
    id_shipment     SERIAL      PRIMARY KEY,
    id_order        INTEGER     NOT NULL
                    REFERENCES orders(id_order)
                    ON DELETE RESTRICT,
    id_vehicle      INTEGER
                    REFERENCES vehicles(id_vehicle)
                    ON DELETE SET NULL,
    id_route        INTEGER
                    REFERENCES routes(id_route)
                    ON DELETE SET NULL,
    planned_date    DATE        NOT NULL,
    actual_date     DATE,
    status          VARCHAR(30) NOT NULL DEFAULT 'planned'
                    CHECK (status IN
                      ('planned','in_progress','completed','cancelled'))
);

-- Таблиця 10: Пункти маршруту
CREATE TABLE route_points (
    id_point               SERIAL        PRIMARY KEY,
    id_route               INTEGER       NOT NULL
                           REFERENCES routes(id_route)
                           ON DELETE CASCADE,
    id_client              INTEGER       NOT NULL
                           REFERENCES clients(id_client)
                           ON DELETE RESTRICT,
    visit_order            INTEGER       NOT NULL CHECK (visit_order >= 1),
    distance_from_prev_km  NUMERIC(8,2)  CHECK (distance_from_prev_km >= 0),
    estimated_arrival_min  INTEGER       CHECK (estimated_arrival_min >= 0),
    UNIQUE (id_route, visit_order)
);

-- Таблиця 11: Користувачі системи
CREATE TABLE users (
    id_user         SERIAL          PRIMARY KEY,
    login           VARCHAR(50)     NOT NULL UNIQUE,
    password_hash   VARCHAR(64)     NOT NULL,
    full_name       VARCHAR(200),
    role            VARCHAR(20)     NOT NULL
                    CHECK (role IN ('manager','logist','warehouse'))
```

```
);

-- ============================================================
-- Індекси для прискорення типових запитів
-- ============================================================
CREATE INDEX idx_stock_warehouse      ON stock(id_warehouse);
CREATE INDEX idx_stock_product        ON stock(id_product);
CREATE INDEX idx_orders_client        ON orders(id_client);
CREATE INDEX idx_orders_status        ON orders(status);
CREATE INDEX idx_order_items_order    ON order_items(id_order);
CREATE INDEX idx_shipments_order      ON shipments(id_order);
CREATE INDEX idx_route_points_route   ON route_points(id_route);
```

ДОДАТОК Б

Лістинг ключових програмних модулів системи

Б.1 Модуль підключення до бази даних (app/db.py)

```
import psycopg2
from app.config import DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD

def get_connection():
    return psycopg2.connect(
        host=DB_HOST,
        port=DB_PORT,
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASSWORD
    )
```

Б.2 Конфігурація застосунку (app/config.py)

```
import os

DB_HOST     = os.getenv('DB_HOST',     'localhost')
DB_PORT     = os.getenv('DB_PORT',     '5432')
DB_NAME     = os.getenv('DB_NAME',     'logistics_db')
DB_USER     = os.getenv('DB_USER',     'logistics_user')
DB_PASSWORD = os.getenv('DB_PASSWORD', 'changeme')
SECRET_KEY  = os.getenv('SECRET_KEY',  'dev-secret-key')
```

Б.3 Ініціалізація Flask-застосунку (app/__init__.py)

```
from flask import Flask
from app.config import SECRET_KEY
from app.routes.auth      import auth_bp
from app.routes.manager   import manager_bp
from app.routes.logist    import logist_bp
from app.routes.warehouse import warehouse_bp

def create_app():
    app = Flask(__name__)
    app.secret_key = SECRET_KEY

    app.register_blueprint(auth_bp)
    app.register_blueprint(manager_bp,   url_prefix='/manager')
    app.register_blueprint(logist_bp,    url_prefix='/logist')
    app.register_blueprint(warehouse_bp, url_prefix='/warehouse')
```

```
    return app
```

Б.4 Модуль авторизації (app/routes/auth.py)

```
from flask import (Blueprint, request, session,
                   redirect, url_for, render_template)
from app.db import get_connection
import hashlib

auth_bp = Blueprint('auth', __name__)

def require_role(*roles):
    from functools import wraps
    def decorator(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            if 'role' not in session or session['role'] not in roles:
                return redirect(url_for('auth.login'))
            return f(*args, **kwargs)
        return wrapper
    return decorator

@auth_bp.route('/')
def index():
    if 'role' in session:
        return redirect(url_for(f"{session['role']}.index"))
    return redirect(url_for('auth.login'))

@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        login_val = request.form['login']
        password  = request.form['password']
        pwd_hash  = hashlib.sha256(password.encode()).hexdigest()
        conn = get_connection()
        cur  = conn.cursor()
        cur.execute(
            "SELECT id_user, role FROM users "
            "WHERE login=%s AND password_hash=%s",
            (login_val, pwd_hash)
        )
        user = cur.fetchone()
        cur.close(); conn.close()
        if user:
            session['user_id'] = user[^0]
            session['role']    = user[^1]
            return redirect(url_for(f"{user[^1]}.index"))
```

```
        return render_template('login.html',
                               error='Невірний логін або пароль')
    return render_template('login.html')

@auth_bp.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('auth.login'))
```

Б.5 Модуль управління замовленнями (app/services/order_service.py)

```
from app.db import get_connection

def get_orders(filters: dict = None) -> list:
    conn = get_connection()
    cur  = conn.cursor()
    sql  = """
        SELECT o.id_order, c.name, o.order_date,
               o.desired_delivery_date, o.status
        FROM orders o
        JOIN clients c ON o.id_client = c.id_client
        WHERE 1=1
    """
    params = []
    if filters:
        if filters.get('status'):
            sql += " AND o.status = %s"
            params.append(filters['status'])
        if filters.get('id_client'):
            sql += " AND o.id_client = %s"
            params.append(filters['id_client'])
    sql += " ORDER BY o.order_date DESC"
    cur.execute(sql, params)
    rows = cur.fetchall()
    cur.close(); conn.close()
    return rows

def create_order(id_client: int, desired_date: str,
                 notes: str, items: list) -> int:
    conn = get_connection()
    cur  = conn.cursor()
    try:
        cur.execute("""
            INSERT INTO orders (id_client, desired_delivery_date, notes)
            VALUES (%s, %s, %s) RETURNING id_order
        """, (id_client, desired_date, notes))
        id_order = cur.fetchone()[^0]
        for item in items:
```

```
            cur.execute("""
                INSERT INTO order_items
                (id_order, id_product, id_warehouse, quantity)
                VALUES (%s, %s, %s, %s)
            """, (id_order, item['id_product'],
                  item['id_warehouse'], item['quantity']))
        conn.commit()
        return id_order
    except Exception:
        conn.rollback()
        raise
    finally:
        cur.close(); conn.close()

def confirm_order(order_id: int) -> dict:
    conn = get_connection()
    cur  = conn.cursor()
    try:
        cur.execute("""
            SELECT id_product, id_warehouse, quantity
            FROM order_items WHERE id_order = %s
        """, (order_id,))
        items = cur.fetchall()
        if not items:
            return {'success': False, 'message': 'Позиції замовлення відсутні'}

        for id_product, id_warehouse, qty in items:
            cur.execute("""
                SELECT quantity FROM stock
                WHERE id_product=%s AND id_warehouse=%s
                FOR UPDATE
            """, (id_product, id_warehouse))
            row = cur.fetchone()
            if not row or row[^0] < qty:
                conn.rollback()
                return {'success': False,
                        'message': f'Недостатньо залишків: '
                                   f'продукт {id_product}'}
            cur.execute("""
                UPDATE stock
                SET quantity = quantity - %s, updated_at = NOW()
                WHERE id_product=%s AND id_warehouse=%s
            """, (qty, id_product, id_warehouse))

        cur.execute("""
            UPDATE orders SET status='confirmed'
            WHERE id_order=%s
        """, (order_id,))
        conn.commit()
```

```
        return {'success': True}
    except Exception as e:
        conn.rollback()
        return {'success': False, 'message': str(e)}
    finally:
        cur.close(); conn.close()
```

Б.6 Модуль маршрутизації (app/services/route_service.py)

```
from geopy.distance import geodesic
from app.db import get_connection

def calculate_route(shipment_id: int,
                    warehouse_coords: tuple) -> dict:
    conn = get_connection()
    cur  = conn.cursor()
    cur.execute("""
        SELECT DISTINCT c.id_client, c.latitude, c.longitude
        FROM order_items oi
        JOIN orders o   ON oi.id_order   = o.id_order
        JOIN shipments s ON s.id_order   = o.id_order
        JOIN clients c   ON o.id_client  = c.id_client
        WHERE s.id_shipment = %s
          AND c.latitude IS NOT NULL
          AND c.longitude IS NOT NULL
    """, (shipment_id,))
    points = cur.fetchall()

    if not points:
        return {'success': False, 'message': 'Немає пунктів доставки'}

    unvisited = list(points)
    current   = warehouse_coords
    route     = []
    total_km  = 0.0

    while unvisited:
        nearest  = min(unvisited,
                       key=lambda p: geodesic(
                           current, (p[^1], p[^2])).km)
        dist_km  = geodesic(current, (nearest[^1], nearest[^2])).km
        total_km += dist_km
        route.append({'id_client':       nearest[^0],
                      'lat':             nearest[^1],
                      'lon':             nearest[^2],
                      'dist_from_prev':  round(dist_km, 2)})
        current   = (nearest[^1], nearest[^2])
        unvisited.remove(nearest)
```

```
    cur.execute("""
        INSERT INTO routes
          (total_distance_km, points_count, created_at, is_optimal)
        VALUES (%s, %s, NOW(), TRUE)
        RETURNING id_route
    """, (round(total_km, 2), len(route)))
    id_route = cur.fetchone()[^0]

    for order, point in enumerate(route, start=1):
        cur.execute("""
            INSERT INTO route_points
              (id_route, id_client, visit_order,
               distance_from_prev_km)
            VALUES (%s, %s, %s, %s)
        """, (id_route, point['id_client'],
              order, point['dist_from_prev']))

    cur.execute("""
        UPDATE shipments SET id_route=%s
        WHERE id_shipment=%s
    """, (id_route, shipment_id))
    conn.commit()
    cur.close(); conn.close()

    return {'success':   True,
            'id_route':  id_route,
            'total_km':  round(total_km, 2),
            'route':     route}
```

Б.7 Модуль генерації PDF-звіту (app/services/report_service.py)

```
from reportlab.lib.pagesizes  import A4
from reportlab.platypus        import (SimpleDocTemplate, Table,
                                       TableStyle, Paragraph, Spacer)
from reportlab.lib             import colors
from reportlab.lib.styles      import getSampleStyleSheet
from reportlab.lib.units       import cm
from reportlab.pdfbase         import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
import io
from app.db import get_connection

def generate_route_pdf(id_route: int) -> bytes:
    pdfmetrics.registerFont(
        TTFont('DejaVu', 'app/static/fonts/DejaVuSans.ttf'))
    styles = getSampleStyleSheet()
```

```
conn = get_connection()
cur  = conn.cursor()
cur.execute("""
    SELECT r.id_route, r.total_distance_km, r.points_count
    FROM routes r WHERE r.id_route = %s
""", (id_route,))
route_info = cur.fetchone()

cur.execute("""
    SELECT rp.visit_order, c.name, c.address,
           rp.distance_from_prev_km
    FROM route_points rp
    JOIN clients c ON rp.id_client = c.id_client
    WHERE rp.id_route = %s
    ORDER BY rp.visit_order
""", (id_route,))
rows = cur.fetchall()
cur.close(); conn.close()

buffer = io.BytesIO()
doc    = SimpleDocTemplate(buffer, pagesize=A4,
                           leftMargin=2*cm, rightMargin=2*cm,
                           topMargin=2*cm, bottomMargin=2*cm)
elements = []

title_style = styles['Title']
title_style.fontName = 'DejaVu'
elements.append(Paragraph(
    f'Маршрутний лист № {id_route}', title_style))
elements.append(Spacer(1, 0.5*cm))

if route_info:
    info_style = styles['Normal']
    info_style.fontName = 'DejaVu'
    elements.append(Paragraph(
        f'Загальна відстань: {route_info[^1]} км  |  '
        f'Кількість пунктів: {route_info[^2]}',
        info_style))
    elements.append(Spacer(1, 0.5*cm))

data = [['№', 'Клієнт', 'Адреса', 'Відстань (км)']]
for row in rows:
    data.append([str(row[^0]), row[^1], row[^2],
                 str(row[^3]) if row[^3] else '—'])

table = Table(data, colWidths=[1.2*cm, 5*cm, 8*cm, 3*cm])
table.setStyle(TableStyle([
    ('BACKGROUND',     (0,0), (-1,0),  colors.HexColor('#2c3e50')),
    ('TEXTCOLOR',      (0,0), (-1,0),  colors.white),
```

```
        ('FONTNAME',       (0,0), (-1,-1), 'DejaVu'),
        ('FONTSIZE',       (0,0), (-1,-1), 9),
        ('ALIGN',          (0,0), (-1,-1), 'LEFT'),
        ('GRID',           (0,0), (-1,-1), 0.5, colors.grey),
        ('ROWBACKGROUNDS', (0,1), (-1,-1),
         [colors.white, colors.HexColor('#f2f2f2')]),
        ('TOPPADDING',     (0,0), (-1,-1), 4),
        ('BOTTOMPADDING',  (0,0), (-1,-1), 4),
    ]))
    elements.append(table)
    doc.build(elements)
    return buffer.getvalue()
```

Б.8 Точка запуску застосунку (run.py)

```
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=False)
```

Б.9 Файл залежностей (requirements.txt)

```
Flask==3.0.3
psycopg2==2.9.9
Jinja2==3.1.4
reportlab==4.1.0
openpyxl==3.1.2
geopy==2.4.1
```

ДОДАТОК В

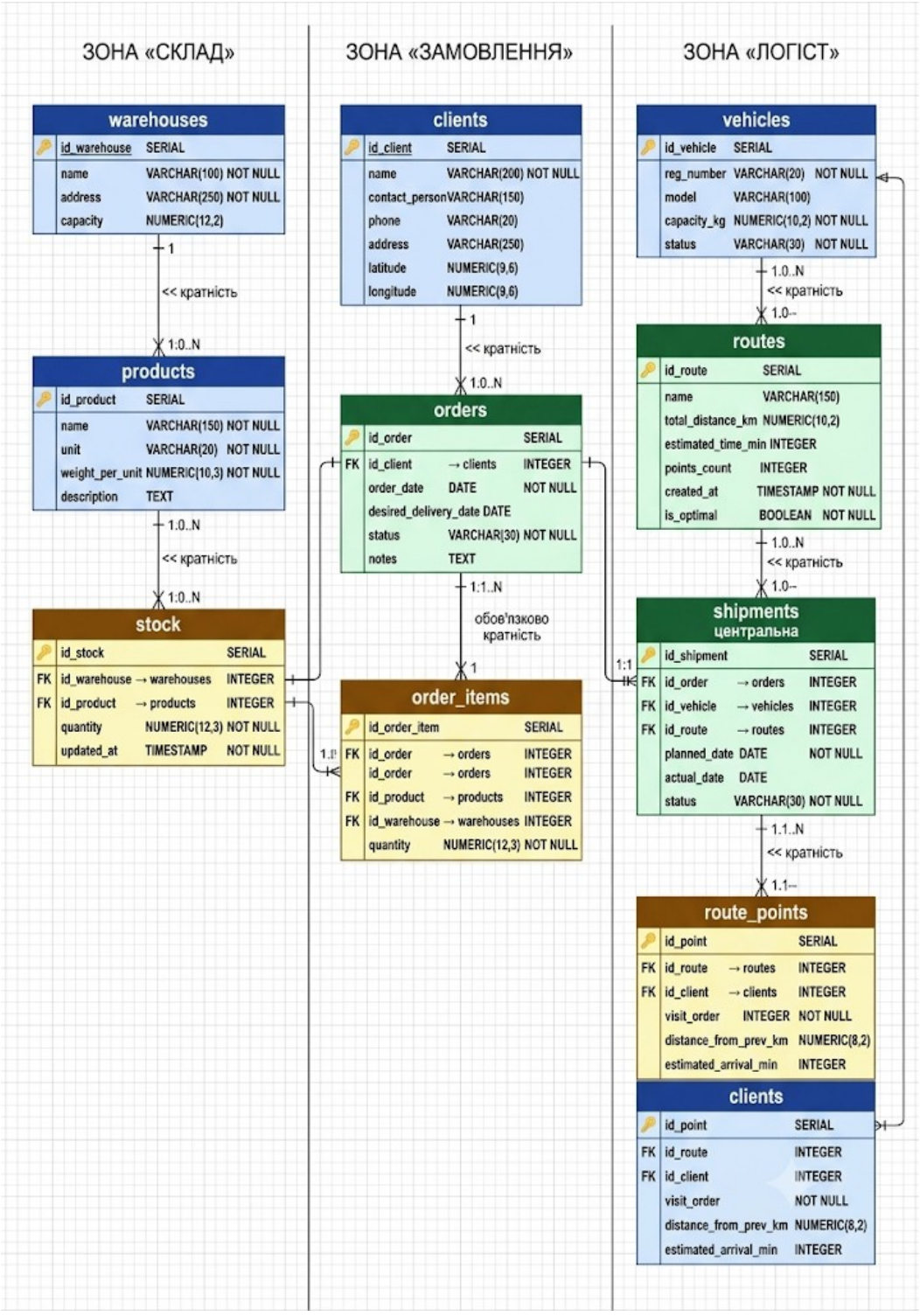

Рис. В.1. ER-діаграма логічної моделі даних

ДОДАТОК Д

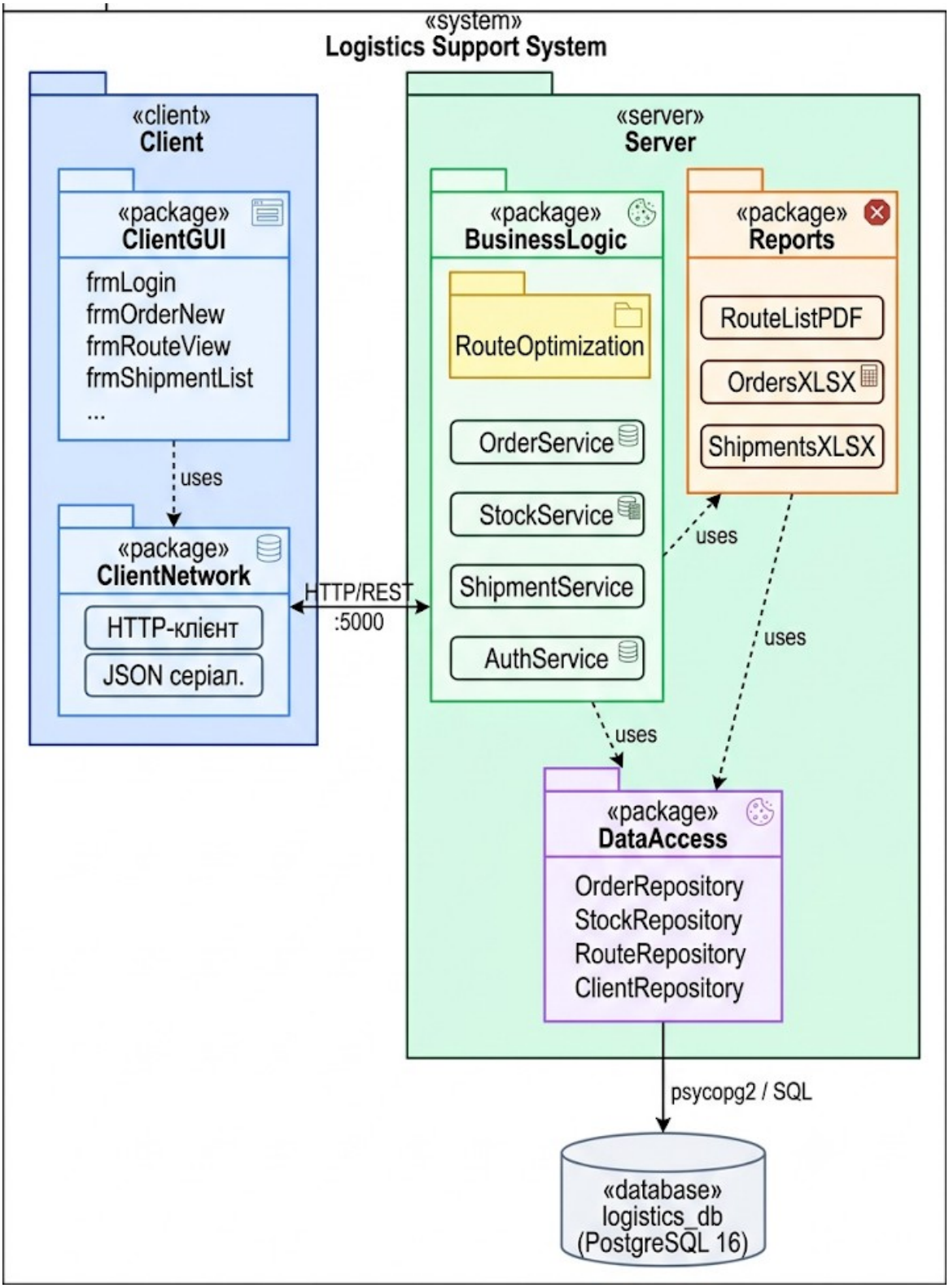

Рис. Д.1 Діаграма пакетів ППЗ англ.

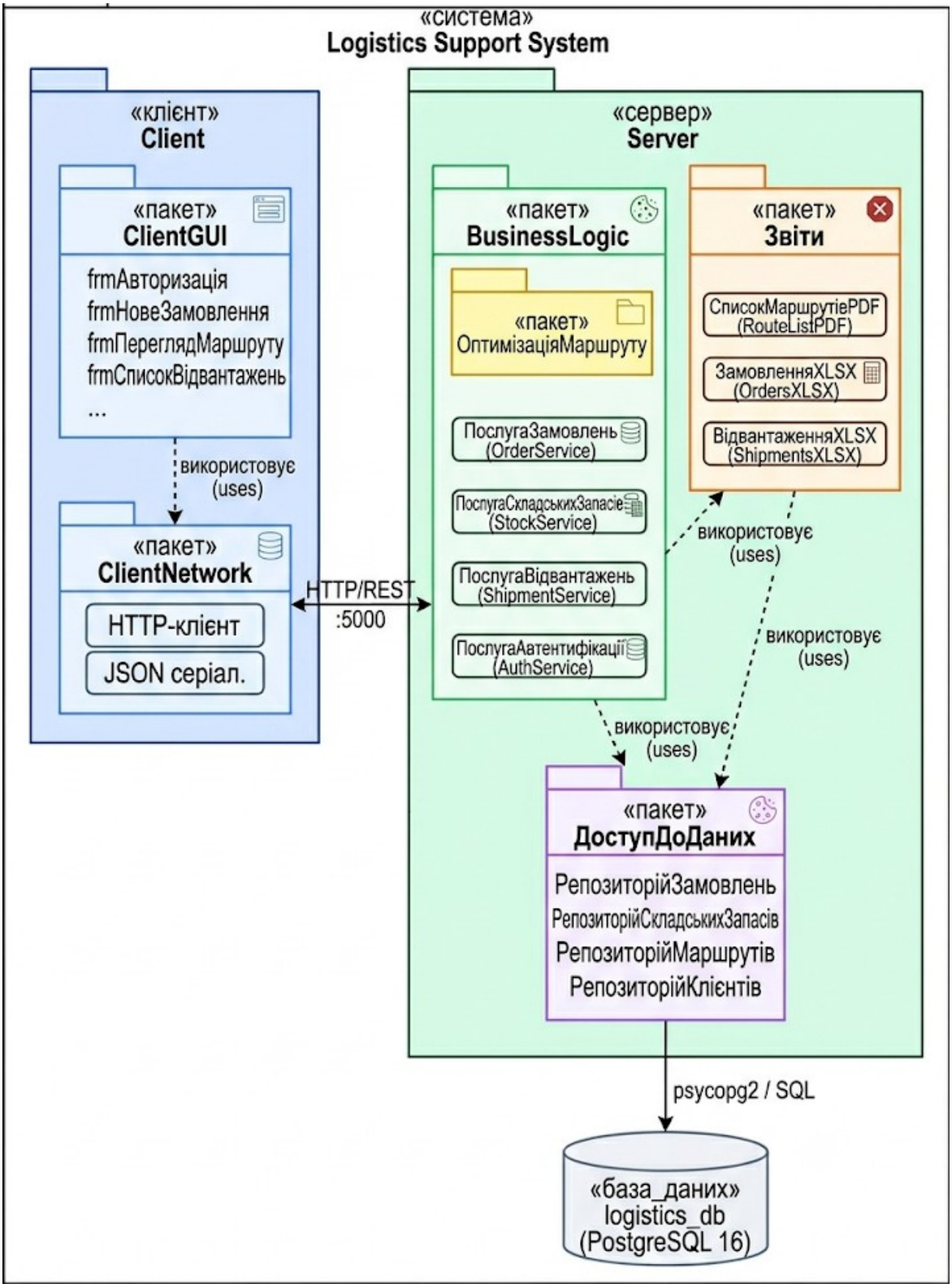

Рисунок Д.2 Діаграма пакетів ППЗ (укр.)

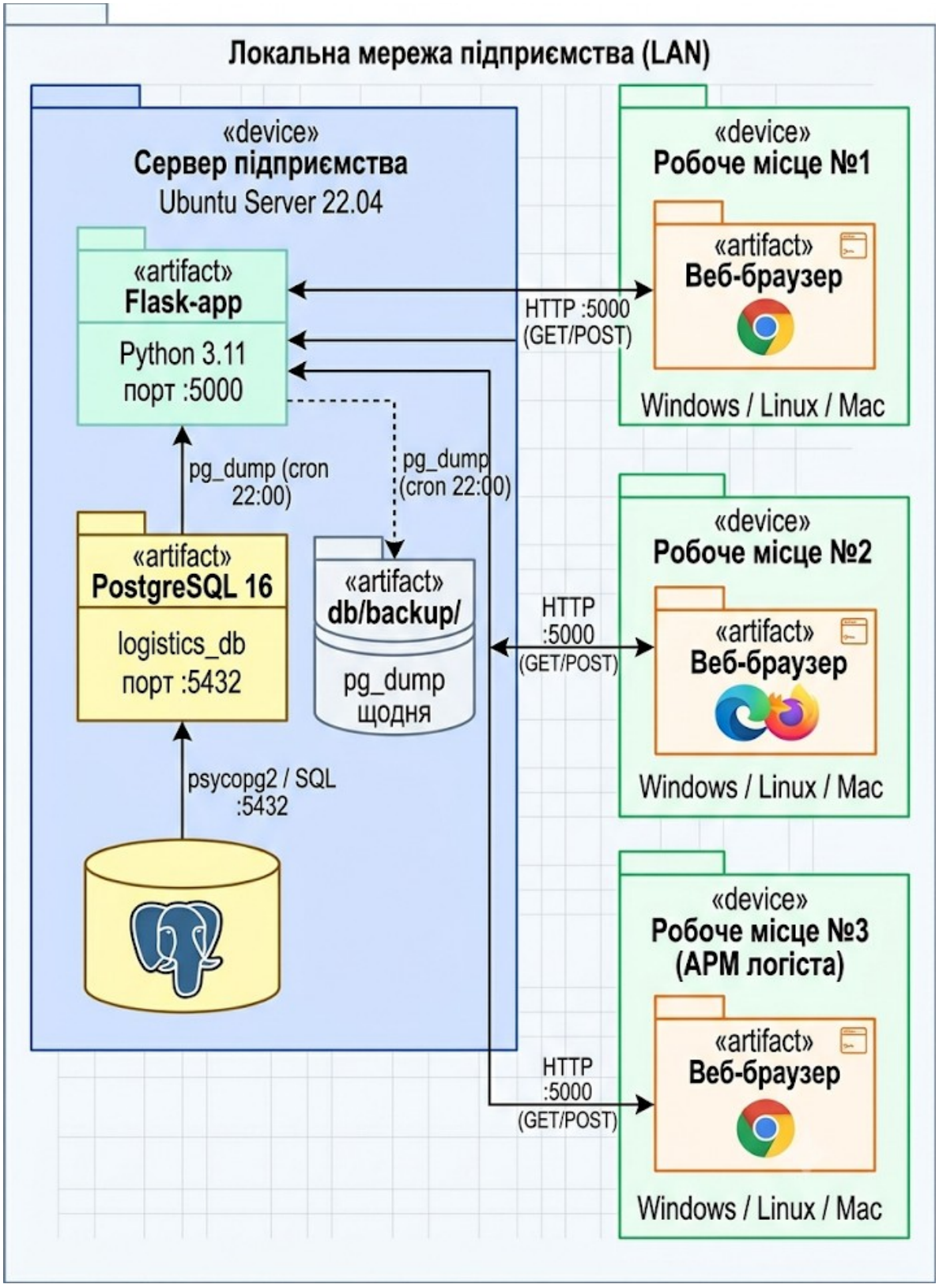

Рис. Д.3 Діаграма розміщення компонентів системи

ДОДАТОК И

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій
Декларація про академічну доброчесність та використання ШІ

Я, Сопрун Олександр Юрійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмний додаток для вирішення задач логістики виробничого підприємства» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що:
   - інструменти ШІ (ChatGPT, Claude, DeepL) були використані для:
     - перекладу іншомовних джерел;
     - стилістичного редагування та перевірки граматики;
     - допомоги у написанні/відлагодженні програмного коду;

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«20» травня 2026 р. **Олександр СОПРУН**