НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення системи обліку замовлень в кав'ярні»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи: /Василюк-Зайцева С.В/
підпис ПІБ

Виконав: / Коннік Д.П. /
підпис ПІБ

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

" " _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Конніку Данилу Павловичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення системи обліку замовлень в кав'ярні

Затверджена наказом ректора НУБіП України № 682 "С" від 24.04.2025

2. Термін подання завершеної роботи на кафедру 2025 . ____. ____
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи _____ / Василюк-Зайцева С.В/
підпис ініціали та прізвище

Завдання прийняла до виконання _____ / Коннік Д.П. /
підпис ініціали та прізвище

Дата отримання завдання 2025 .____.____
рік, місяць, число

АНОТАЦІЯ

У сучасному суспільстві збільшується кількість інформації, яка потребує обробки. Завдяки автоматизованій системі, касир зможе швидко та ефективно обробляти замовлення. Ручний облік може супроводжуватись людськими помилками, тому система дозволить забезпечити точність та надійність, зменшуючи ризик помилок, що сприятиме ефективній роботі касира.

У дипломній роботі розглянуто процес розробки програмного забезпечення для обліку замовлень у кав'ярні. Основну увагу приділено моделюванню структури даних, побудові архітектури та реалізації функціоналу для управління касирами, товарами, транзакціями й процесом оформлення замовлень. Система передбачає авторизацію користувачів за ролями, ведення історії продажів, взаємодію з базою даних PostgreSQL, а також надає зручний інтерфейс, створений з використанням React. У роботі описано етапи проєктування, вибір технологічного стеку (NestJS, React, Prisma), а також результати тестування програмного продукту. Запропоноване рішення спрямоване на задоволення потреб малих закладів громадського харчування в ефективному й легкому впровадженні облікового інструменту.

ABSTRACT

In modern society, the amount of information that needs to be processed is increasing. Thanks to an automated system, the cashier will be able to process orders quickly and efficiently. Manual accounting can be accompanied by human errors, so the system will ensure accuracy and reliability, reducing the risk of errors, which will contribute to the effective work of the cashier.

The thesis describes the process of developing software for accounting for orders in a coffee shop. The main attention is paid to modelling the data structure, building the architecture and implementing the functionality for managing cashiers, goods, transactions and the ordering process. The system provides for user authorisation by role, sales history, interaction with the PostgreSQL database, and a user-friendly interface created using React. The paper describes the design stages, the choice of a technology stack (NestJS, React, Prisma), and the results of testing the software product. The proposed solution is aimed at meeting the needs of small catering establishments for an effective and easy implementation of an accounting tool.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability;

API – Application Programming Interface;

CRM – Customer relationship management;

ER – Entity-Relationship;

JSON – JavaScript Object Notation;

NestJS – A progressive Node.js framework for building efficient and scalable server-side applications;

ORM – Object-Relational Mapping;

POS – Point of Sale;

SQL – Structured Query Language;

UML – Unified Modeling Language.

ПІБ – Прізвище, ім'я, по батькові;

СУБД – Система управління базами даних.

ВСТУП

У сучасних умовах трансформації сфери обслуговування, коли очікування споживачів дедалі більше орієнтовані на швидкість, персоналізацію та цифрову інтегрованість, питання автоматизації внутрішніх бізнес-процесів постає як одне з ключових. Кав’ярні, як динамічно зростаючий сегмент малого підприємництва, зіштовхуються з постійною необхідністю оперативного обліку замовлень, ведення клієнтської бази та прийняття управлінських рішень на основі актуальної аналітики. Наявність ефективного інформаційного середовища забезпечує конкурентоспроможність навіть невеликих закладів у жорстких ринкових умовах.

На жаль, більшість рішень, що наразі пропонуються на ринку, зосереджені або на великих франшизах, або мають громіздку архітектуру, що ускладнює адаптацію до реальних потреб малих кав’ярень. Поширені комерційні програмні продукти, як-от iiko, Poster POS чи Square, хоча й пропонують широкий функціонал, часто є або надмірно дорогими, або складними в налаштуванні та інтеграції в локальні облікові системи [1].

Окремо варто відзначити, що більшість існуючих систем орієнтовані переважно на front-end інтерфейс, залишаючи back-end складову закритою, що обмежує можливості розширення, інтеграції зі сторонніми аналітичними сервісами або застосуванням алгоритмів машинного навчання для побудови рекомендацій. Наукова спільнота натомість приділяє дедалі більше уваги темам цифровізації малого бізнесу, модульності архітектури програмних рішень, зменшенню бар’єрів входу для підприємців у сферу цифрових сервісів.

З огляду на викладене, постає науково-технічна задача розроблення програмного забезпечення, яке б поєднувало гнучкість архітектури, відкритість до інтеграцій, простоту розгортання та відповідність сучасним стандартам розробки (зокрема, використання модульного бекенду, REST API, ORM-технологій, а також сумісність із PostgreSQL як однією з найпопулярніших

СУБД з відкритим кодом). Вирішення такої задачі дозволить не лише створити ефективне прикладне рішення для кав’ярень, але й зробить внесок у розвиток методологій модульного проєктування інформаційних систем для малого бізнесу.

Мета роботи полягає у створенні функціонально повноцінного, технологічно гнучкого та адаптованого до потреб кав’ярень програмного забезпечення для автоматизованого обліку замовлень, реалізованого на основі сучасних технологій.

Для досягнення поставленої мети необхідно вирішити такі основні задачі:

- провести системний аналіз предметної області з урахуванням структури та специфіки обліку замовлень у кав’ярні, визначити функціональні вимоги до програмної системи, сформулювати її архітектурні й логічні обмеження;
- розробити інформаційну модель системи обліку замовлень, побудувати логічну структуру бази даних, обґрунтувати вибір СУБД та засобів взаємодії з нею, реалізувати схему бази даних з урахуванням цілісності й розширюваності;
- реалізувати прикладне програмне забезпечення на базі модульної архітектури NestJS із розділенням відповідальності на рівні контролерів, сервісів та модулів; забезпечити інтеграцію з базою даних через Prisma ORM та реалізувати ключові бізнес-функції;
- провести тестування працездатності програмної системи, сформулювати рекомендації щодо її експлуатації та розгортання, визначити вимоги до апаратного забезпечення та сформувати склад інсталяційного пакета для кінцевого користувача.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Кав’ярні стали звичним елементом міської інфраструктури та популярним форматом малого підприємництва, що поєднує елементи гастрономії, сфери обслуговування та соціального простору. Вони функціонують у динамічному середовищі, де швидкість обробки замовлень, зручність обслуговування клієнтів та контроль за витратними ресурсами відіграють визначальну роль для ефективної роботи закладу. Операційна діяльність кав’ярні включає постійне опрацювання замовлень, ведення касових операцій, облік товарів і взаємодію між персоналом, що потребує чітко скоординованої інформаційної підтримки.

Щоденна робота таких закладів передбачає участь кількох ролей – касира, адміністратора, а в окремих випадках і бухгалтера, кожен з яких виконує визначені функції в межах обслуговування та управління [2]. Спільна взаємодія персоналу базується на повторюваних процедурах, які потребують формалізації: прийняття замовлення, оформлення чека, списання товарів, оновлення меню, перегляд фінансових результатів тощо. Ручне виконання цих процесів призводить до накопичення помилок, неузгодженостей, втрати часу та відсутності прозорої аналітики [3]. Тому автоматизація внутрішнього обліку є обов’язковою умовою стабільного функціонування кав’ярні в умовах сучасної конкуренції.

Підприємець, який організовує кав’ярню, прагне мінімізувати кількість рутинних операцій і водночас мати контроль над даними про продажі, залишки товарів та зміною складу працівників. У впровадженні інформаційної системи основна увага зосереджується на створенні зручного інтерфейсу для касира,

швидкому оформленні замовлень, формуванні меню і переглядати аналітичну звітність. Важливо, щоб взаємодія між компонентами системи була простою й передбачуваною, а зміна одного об'єкта (наприклад, замовлення) автоматично відображалась в інших складових – звіті.

До основних процесів, що реалізуються в межах інформаційної системи кав’ярні, належать:

– прийом замовлення касиром;

– формування замовлення із зазначенням обраних товарів та кількості;

– реєстрація способу оплати та проведення транзакції;

– додавання нових позицій у меню або оновлення характеристик товарів;

– ведення довідника персоналу;

– формування і перегляд звітності за визначеними періодами (щоденної, щомісячної, по товарам тощо);

– керування правами доступу до функціоналу відповідно до ролей користувачів.

Кожен із цих процесів вимагає чітко визначеної логіки, взаємопов’язаної з іншими складовими системи. Саме узгоджена реалізація зазначених функцій дозволяє забезпечити ефективність, оперативність і прозорість роботи кав’ярні без надмірного навантаження на персонал.

1.2 Аналіз вимог до програмної системи

Проєктування програмного забезпечення для автоматизованого обліку замовлень у кав’ярні передбачає визначення вимог до системи як з боку її функціонального наповнення, так і з позицій структурної, технологічної та користувацької придатності. Вимоги до системи формуються на основі аналізу специфіки предметної області, очікуваного сценарію взаємодії з користувачем, ролей персоналу, а також загальних принципів побудови надійних, масштабованих інформаційних рішень.

Згідно з логікою функціонування кав’ярні, система повинна забезпечувати взаємодію між двома основними ролями користувачів: адміністратором і касиром, кожен з яких має власну сферу відповідальності та доступ до відповідного підмножини функціоналу.

Адміністратор є вищим рівнем управління в системі. Йому надаються повноваження з управління кадровим складом, адміністрування товарного довідника (додавання нових товарів, редагування існуючих позицій, оновлення цін), а також перегляду інформації про проведені транзакції. Таким чином, адміністратор виконує функції стратегічного управління та аналітичного спостереження.

У межах програмного інтерфейсу адміністратора повинні бути реалізовані такі розділи:

- касир – додавання нового касира, перегляд наявного переліку, доступ до профільної інформації;
- товар – можливість додавання нових позицій у меню, редагування товарів, перегляд актуального меню;
- транзакції – перегляд історії операцій, контроль за фінансовими потоками;
- вихід – завершення сесії з облікового запису.

Касир як роль орієнтована переважно на оперативну діяльність. Його основна взаємодія із системою полягає у веденні касових операцій: реєстрація замовлень, додавання товарів у процесі продажу, взаємодія з клієнтами. Додатково, як і адміністратор, касир має доступ до довідника товарів, що дозволяє йому оперативно доповнювати асортимент (наприклад, при появі нової позиції на зміні).

Для касира передбачено такі основні функціональні блоки:

- товар – доступ до перегляду й додавання нових товарів у продаж;
- робота – касовий модуль, через який здійснюється реєстрація замовлень;
- вихід – завершення роботи з обліковим записом.

Обов’язковим елементом системи є механізм авторизації, що забезпечує доступ до функціоналу відповідно до ролі користувача. Верифікація здійснюється за логіном та паролем, після чого інтерфейс адаптується відповідно до доступних можливостей конкретного облікового запису. Це дозволяє дотримуватися принципів безпеки та ролевої ізоляції.

Крім функціональних вимог, до системи висуваються нефункціональні вимоги, серед яких:

- зручність користування – інтерфейс має бути інтуїтивно зрозумілим для користувачів без спеціальної підготовки;
- надійність – усі операції повинні фіксуватись коректно з гарантією збереження даних у разі нештатних ситуацій;
- масштабованість – у подальшому система повинна підтримувати розширення (наприклад, додавання нових ролей, інтеграцію з аналітичними сервісами);
- безпека – захист персональних облікових даних, обмеження доступу до конфіденційної інформації;
- продуктивність – забезпечення стабільної роботи навіть при інтенсивному обслуговуванні.

Таким чином, аналіз вимог до програмної системи дозволяє сформулювати чітке технічне завдання на реалізацію програмного продукту, що задовольняє ключові потреби кав’ярні в контексті обліку замовлень, обробки фінансових операцій та адміністрування персоналу.

1.3 Огляд інформаційних джерел та існуючих рішень

Питання автоматизації процесів обліку замовлень у закладах громадського харчування, зокрема в кав’ярнях, вже тривалий час є предметом практичного інтересу як для бізнесу, так і для розробників прикладного програмного забезпечення. Ця потреба зумовлена як зростанням кількості малих підприємств у сфері харчування, так і необхідністю забезпечити стабільність, швидкість та прозорість операцій обслуговування клієнтів. У сучасних умовах цифровізації, коли оперативний доступ до облікових даних стає запорукою конкурентоспроможності, автоматизовані системи набувають особливої ваги. На ринку представлено низку рішень, що позиціонуються як POS-системи для ресторанного бізнесу або універсальні облікові системи з можливістю адаптації під специфіку кав’ярні. Найбільш поширені серед них – це такі продукти, як Poster POS, Loyverse POS, SambaPOS, а також сервіси типу Square, які забезпечують хмарну інфраструктуру для ведення обліку продажів, складу, клієнтів і звітності. У багатьох випадках такі системи виступають базовим інструментом організації бізнесу для новостворених закладів або франшиз. Водночас попри широку присутність на ринку, наявні рішення не завжди відповідають усім вимогам простоти, доступності та адаптивності, що особливо важливо для невеликих закладів громадського харчування в умовах обмежених ресурсів. Серед найбільш актуальних проблем користувачі відзначають складність налаштування, обмеження в локалізації, залежність від платної підписки або недостатню гнучкість у налаштуванні бізнес-логіки. Усе це формує запит на створення спеціалізованого, адаптованого та водночас

нескладного у впровадженні рішення, орієнтованого на реальні потреби українських кав'ярень.

Poster POS – це хмарна POS-система, призначена для автоматизації бізнес-процесів у закладах громадського харчування, таких як кав'ярні, бари та ресторани (рис. 1.3). Основною метою застосунку є забезпечення зручного і швидкого обслуговування клієнтів. Poster POS підтримує роботу з планшетами, POS-терміналами, мобільними принтерами і банківськими терміналами, що дозволяє запускати систему в межах мінімального обладнання [4].

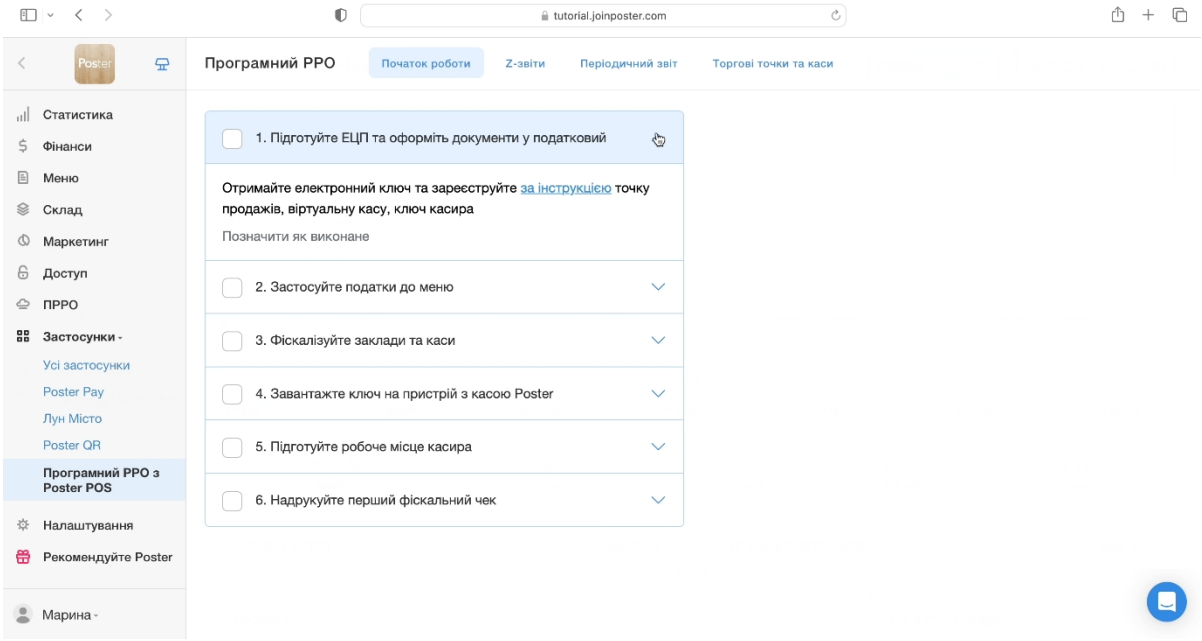

Рис. 1.3 Приклад інтерфейсу системи «Poster POS» [5]

Архітектура Poster POS побудована за принципом клієнт–серверної взаємодії з використанням хмарної інфраструктури. Серверна частина зберігає дані про заклад, меню, замовлення, транзакції, користувачів та статистику у централізованій базі, до якої здійснюється доступ через інтернет. Клієнтська частина – це веб-застосунок або мобільний застосунок, який працює у браузері або як окремий додаток на планшеті. Вона взаємодіє із сервером через API у режимі реального часу або офлайн (із подальшою синхронізацією). Такий підхід забезпечує гнучкість у розгортанні, централізоване оновлення функціоналу, резервне копіювання даних і можливість доступу до статистики

та управління закладом з будь-якого пристрою. Архітектура системи також підтримує багатоточкову модель, коли кілька точок продажу можуть бути пов’язані в єдину управлінську панель для адміністрування мережі закладів.

Переваги Poster POS:

- простий та інтуїтивно зрозумілий інтерфейс, адаптований для планшетів та сенсорних пристроїв;
- хмарна архітектура забезпечує доступ до системи з будь-якої точки світу через інтернет [6];
- вбудовані функції для обліку складу, управління персоналом, CRM та аналітики;
- підтримка офлайн-режиму з автоматичною синхронізацією після відновлення підключення [7].

Недоліки Poster POS:

- система працює за моделлю підписки, що може бути дорогою для малого бізнесу [8];
- відсутність повного доступу до серверного коду та обмежені можливості кастомізації бізнес-логіки;
- залежність від стабільного інтернет-з’єднання при роботі в онлайн-режимі;
- частина функціоналу доступна лише в розширених тарифних планах.

Отже, Poster POS – це ефективне і сучасне рішення для автоматизації кав’ярень і закладів харчування, яке підходить для швидкого запуску бізнесу без значних технічних знань. Завдяки своїй зручності та хмарній моделі воно

забезпечує високий рівень мобільності та зручності, однак обмеження щодо адаптації під специфічні вимоги та модель підписки можуть виявитися стримуючими факторами для закладів із нестандартними потребами або обмеженим бюджетом.

Loyverse POS – це безкоштовна POS-система, орієнтована на малі та середні підприємства, зокрема кав'ярні, магазини, бари й кіоски, яка забезпечує базову автоматизацію продажів, облік товарів, управління клієнтами та аналітику (рис. 1.4). Основне призначення системи полягає в оперативному обслуговуванні замовлень, реєстрації оплат, веденні складу, а також у моніторингу бізнес-показників у режимі реального часу [9]. Застосунок розроблений з акцентом на простоту використання та швидке розгортання без потреби у складному технічному налаштуванні, що робить його зручним рішенням для підприємців без IT-фахівців.

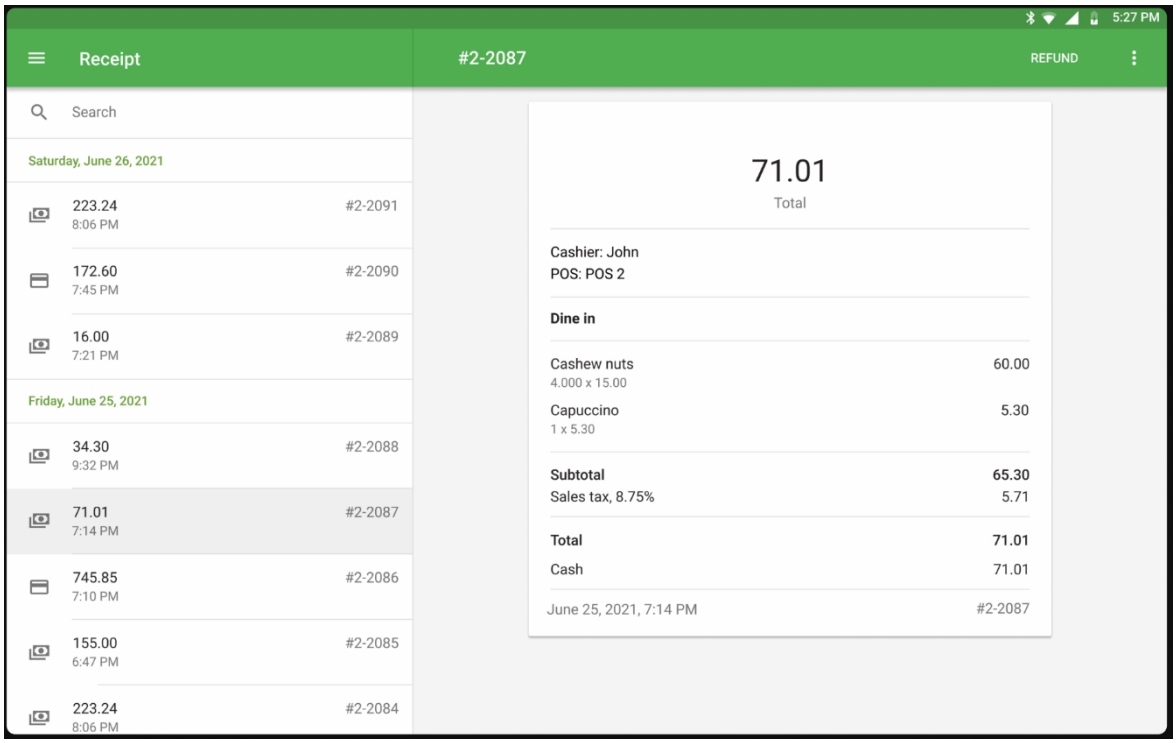

Рис. 1.4 Приклад інтерфейсу системи «Loyverse POS» [10]

Архітектура Loyverse POS базується на хмарній моделі з клієнтським застосунком для Android та iOS, який підключається до серверної інфраструктури через інтернет. Клієнтська частина відповідає за обробку

транзакцій на місці, відображення інтерфейсу користувача, формування чеків і фіксацію продажів. Вся інформація автоматично синхронізується з централізованою базою, яка доступна адміністратору через окрему веб-панель керування. Така архітектура дозволяє керувати кількома торговими точками з єдиного облікового запису, вести аналітику в хмарі, здійснювати віддалене адміністрування персоналу та оновлювати асортимент. У разі втрати інтернет-з'єднання система працює в офлайн-режимі, з подальшою передачею даних на сервер після відновлення зв'язку.

Переваги Loyverse POS:

- безкоштовна базова версія з повним функціоналом для обліку продажів, що робить систему доступною для малого бізнесу;
- простий та зрозумілий інтерфейс, який не потребує спеціальної підготовки персоналу;
- підтримка роботи в офлайн-режимі з автоматичною синхронізацією після відновлення інтернет-з'єднання [11];
- можливість керування декількома торговими точками з однієї обліковки через хмарну адміністративну панель.

Недоліки Loyverse POS:

- обмежені можливості розширення або кастомізації бізнес-логіки, оскільки система є закритою;
- додаткові функції (наприклад, розширена аналітика або управління персоналом) доступні лише за платною підпискою.
- відсутність повного доступу до бази даних або API у безкоштовній версії [12];

- не всі інтеграції (наприклад, з локальними українськими платіжними системами) реалізовані або офіційно підтримуються.

Отже, Loyverse POS – це зручне, швидке в налаштуванні та економічно привабливе рішення для невеликих кав'ярень, яке дозволяє вести базовий облік замовлень і продажів без значних фінансових чи технічних витрат. Воно особливо корисне для підприємців-початківців, однак може бути недостатньо гнучким для закладів, що потребують глибокої інтеграції, розширених звітів або локалізованих функцій.

SambaPOS – це настільна POS-система, призначена для автоматизації обліку замовлень у закладах громадського харчування, зокрема ресторанах, кав'ярнях, барах і кафе (рис. 1.5). Вона орієнтована на користувачів, які потребують високого рівня кастомізації, локального зберігання даних і повного контролю над бізнес-процесами. Основне призначення SambaPOS полягає у веденні продажів, обліку замовлень, управлінні персоналом, контролі складу, генерації чеків, звітності та логістиці в межах одного або кількох залів обслуговування [13]. Система дозволяє створювати індивідуальні сценарії обробки замовлень, визначати користувацькі ролі, автоматизувати дії при певних подіях і керувати роботою через розгалужені правила.

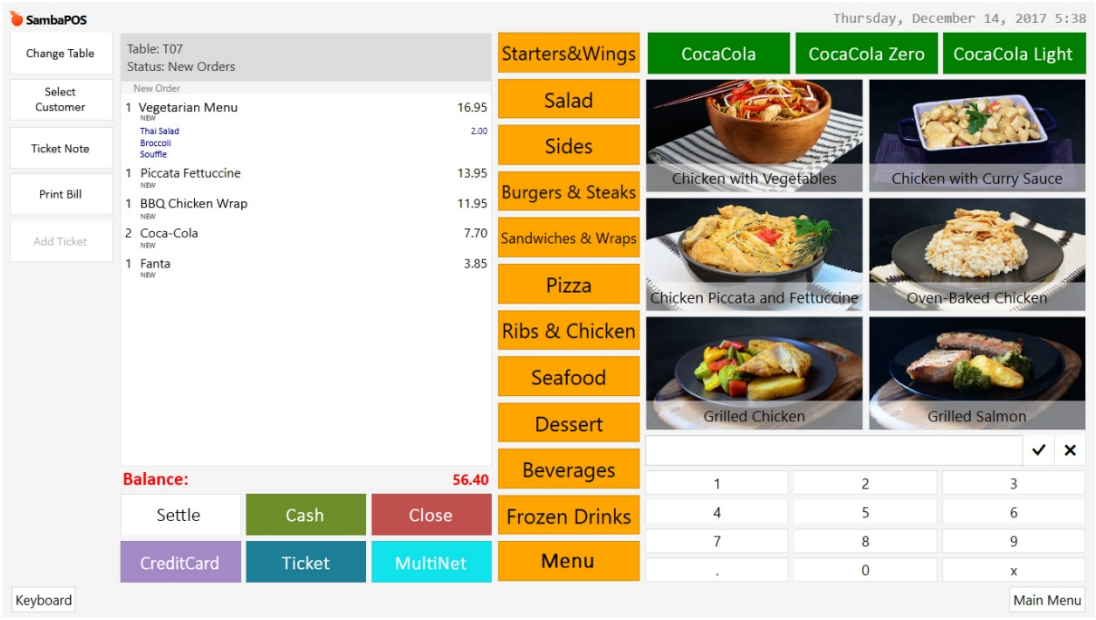

Рис. 1.5 Приклад інтерфейсу системи «SambsPOS» [14]

Архітектура SambaPOS реалізована за принципом клієнт–серверної взаємодії з локальним розгортанням. Серверна частина працює переважно на базі Microsoft SQL Server, що забезпечує централізоване зберігання даних і підтримку багатьох одночасних клієнтів. Клієнтська частина встановлюється на кожен окремий POS-термінал або ПК, зв’язується з сервером у локальній мережі й виконує роль інтерфейсу для персоналу. Завдяки відкритій архітектурі та підтримці скриптів, система дає змогу користувачеві створювати власні правила обробки подій, кастомізувати бізнес-логіку, інтегруватися з іншими локальними або сторонніми сервісами, такими як принтери, ваги, платіжні термінали або системи відеоспостереження. Це рішення ідеально підходить для закладів, які потребують гнучкості, автономності та можливості локального контролю над усіма компонентами облікової системи.

Переваги SambaPOS:

- високий рівень кастомізації бізнес-логіки, з можливістю створення власних правил, подій та сценаріїв обробки;
- локальне зберігання даних на базі Microsoft SQL Server, що забезпечує автономну роботу без залежності від хмари;
- підтримка інтеграцій із широким спектром периферійних пристроїв (принтери, термінали, камери тощо);
- потужна система звітності та логування з можливістю налаштування власних шаблонів і форм звітів [15].

Недоліки SambaPOS:

- потребує складнішого початкового налаштування, що може бути перешкодою для користувачів без технічної підготовки [16];

- відсутність хмарної версії або веб-інтерфейсу для віддаленого адміністрування «з коробки»;
- більшість розширених функцій доступні лише в платній версії системи;
- залежність від Microsoft Windows та SQL Server як серверної платформи обмежує платформну гнучкість.

Отже, SambaPOS – це потужна, локально орієнтована система обліку замовлень, яка надає гнучкі інструменти для створення індивідуального робочого середовища в межах кав'ярні чи ресторану. Вона ідеально підходить для користувачів, які мають досвід адміністрування систем та потребують високого ступеня налаштування під власні бізнес-процеси. Однак через складність у розгортанні та обмежену підтримку хмарних технологій вона менш зручна для малого бізнесу без технічного персоналу.

Для систематизації наведеного вище аналізу доцільно представити порівняльну табл. 1.1, яка демонструє особливості найвідоміших систем обліку замовлень у кав'ярнях за критеріями, що є критичними з огляду на потреби дипломного проєкту. Такий підхід дозволяє об'єктивно порівняти рішення між собою, виокремити переваги й недоліки кожного з них у межах однакової шкали оцінювання. Зіставлення відбувається за ознаками, що безпосередньо впливають на можливість впровадження програмного забезпечення в малих закладах, де важливо враховувати не лише функціональність, а й доступність, адаптивність і технічну простоту.

Таблиця 1.1

Порівняльна таблиця систем обліку замовлень у кав'ярнях

Критерій	Poster POS	Loyverse POS	SambaPOS
Модель розгортання	Хмарна, із доступом через браузер або мобільний застосунок	Хмарна, з мобільним застосунком (Android/iOS)	Локальна, з клієнт-серверною архітектурою
Офлайн-режим	Підтримується з подальшою синхронізацією	Підтримується з подальшою синхронізацією	Працює постійно в офлайн-режимі (локальна БД)
Можливість локалізації	Часткова, залежить від тарифного плану	Часткова, українська доступна, але не всюди	Повна, залежить від налаштувань користувача
Підтримка периферійних пристроїв	Обмежена стандартним обладнанням POS	Базова підтримка (принтери, сканери, термінали)	Розширена підтримка (ваги, камери, термінали, принтери тощо)
Вимоги до технічної підготовки	Мінімальні, підходить для користувачів без досвіду	Мінімальні, інтуїтивно зрозумілий інтерфейс	Високі, налаштування потребує технічної компетенції
Наявність веб-доступу	Так, адміністрування через браузер	Так, хмарна панель управління	Немає з коробки, лише через локальну мережу

Виходячи із проведеного аналізу, можна зробити висновок, що наявні рішення хоча й забезпечують широкий функціонал, однак або потребують фінансових витрат, або вимагають значного технічного налаштування. Для малих кав’ярень із обмеженим штатом і ресурсами такі системи виявляються надмірно складними або затратними. У зв’язку з цим доцільним є розроблення нового програмного забезпечення з базовими, але достатніми можливостями – додавання товару, робота каси, авторизація користувачів та облік транзакцій. Така система має бути простою у впровадженні, підтримувати автономну роботу та передбачати безкоштовне використання основної версії, що відповідає реальним потребам малих кав’ярень в умовах обмежених ресурсів.

1.4 Постановка завдання

У результаті проведеного аналізу предметної області, формалізації вимог до системи та огляду існуючих аналогічних рішень було виявлено, що більшість сучасних програмних продуктів для автоматизації кав’ярень або є надмірно складними для впровадження в умовах малого бізнесу, або вимагають регулярних фінансових витрат на ліцензії, обслуговування чи розширений функціонал. Це створює потребу в розробці компактного та доступного рішення, яке забезпечувало б базову автоматизацію обліку замовлень, не вимагаючи від користувача технічної підготовки або суттєвих інвестицій.

Таким чином, формулюється завдання створити модульну, кросплатформну систему, яка дозволяє організувати повноцінну роботу кав’ярні в контексті внутрішнього обліку. Система повинна передбачати авторизований доступ для користувачів з поділом ролей (адміністратор та касир), реалізовувати базові функції для кожної з них, забезпечувати простоту в користуванні та логічну структуру взаємодії. Архітектура програмного забезпечення має бути реалізована на основі сучасних технологій із використанням фреймворку NestJS (TypeScript), що забезпечить модульність, розширюваність і структурованість серверної частини. Для зберігання даних слід застосувати PostgreSQL у поєднанні з Prisma ORM, що дозволить забезпечити узгодженість, масштабованість і зручність роботи з базою даних.

На основі технічного завдання визначаються такі ключові функціональні блоки системи. Для адміністратора передбачається реалізація можливостей керування касирами (додавання нових у систему, перегляд списку), адміністрування товарного довідника (додавання товарів, перегляд усіх товарних позицій), а також перегляд транзакцій, здійснених у межах системи. Для касира реалізується можливість працювати з товарами (створення та перегляд), а також ведення касових операцій у вигляді окремого робочого інтерфейсу. Обидва типи користувачів мають проходити обов’язкову

авторизацію, після чого доступ до функціоналу системи реалізується згідно з їхньою роллю.

У межах системи необхідно забезпечити надійну обробку транзакцій, повноцінний облік введених даних, коректну логіку переходів між формами та дотримання принципів захисту інформації (шифрування паролів, обмеження доступу до маршрутів API). Архітектура програмного забезпечення має передбачати можливість подальшого масштабування (розширення складу, підключення мобільного клієнта), не порушуючи цілісності існуючої структури.

Таким чином, поставлене завдання передбачає створення повноцінного, локалізованого, простого у впровадженні програмного рішення, яке дозволить малим кав’ярням організувати цифровий облік процесів без використання надлишкових ресурсів та без необхідності звернення до комерційних, складних або дорогих систем.

1.5 Моделювання предметної області

З метою глибшого розуміння особливостей організації процесів у кав’ярні, доцільним є моделювання предметної області засобами об’єктно-орієнтованого аналізу. Одним із найбільш зручних інструментів для цього є діаграма прецедентів (use case diagram), яка дозволяє візуалізувати типові дії учасників процесу та встановити, як саме вони взаємодіють із окремими елементами предметної області. Це дає змогу не лише структурувати поточну організацію роботи, а й виявити потенційні місця, які в майбутньому може вирішити автоматизована система.

На рисунку 1.6 зображено діаграму прецедентів, яка моделює предметну область обліку замовлень у кав’ярні. У діаграмі виділено чотири основні актори, що відповідають реальним учасникам процесів: адміністратор, касир, бухгалтер і клієнт.

Рис. 1.6 Діаграма прецедентів

Адміністратор виконує функції управління персоналом і товарним асортиментом. Він займається наймом касирів, веденням їх обліку а також підтримує актуальність меню. Редагування меню включає як додавання нових позицій, так і оновлення даних про існуючі товари.

Касир є безпосереднім виконавцем процесів обслуговування клієнтів. Його основна задача - обробка замовлень: прийом інформації про замовлення, підбір відповідних позицій із меню, прийом оплати, завершення операції. Також касир має змогу переглядати список товарів, щоб швидко знаходити необхідні позиції для продажу.

Бухгалтер відповідає за формування та аналіз фінансової звітності. До його типових дій належить створення та перегляд звітів про продажі. Ці процеси є важливими для фінансового контролю та оцінки ефективності роботи кав’ярні.

Клієнт - це актор, який ініціює процес створення замовлення. Він взаємодіє з касиром, озвучує свої побажання, після чого замовлення формується в системі. Клієнт, хоч і не працює з інформаційною системою безпосередньо, є ключовим суб’єктом у моделі, оскільки обслуговування його потреб є центральним процесом предметної області.

Таким чином, діаграма прецедентів відображає типову організацію роботи кав’ярні до впровадження програмного забезпечення. Вона дозволяє окреслити основні ролі та їх обов’язки, що є необхідною основою для формування функціональних вимог до майбутньої системи.

Продовжуючи моделювання предметної області, доцільно розглянути динаміку взаємодії між учасниками системи в часі, що дозволяє проаналізувати логіку обробки окремих сценаріїв. Для цього використовується діаграма послідовностей (sequence diagram), яка ілюструє послідовність повідомлень між об’єктами системи під час виконання певного бізнес-процесу. На рис. 1.7 наведено діаграму послідовності, що відображає ключові сценарії взаємодії користувачів із системою: створення замовлення, його обробка, виконання та оновлення інформації про товар, а також формування звітності. Діаграма демонструє часову вісь обміну повідомленнями між такими учасниками: клієнт, касир, меню (як об’єкт системи), бухгалтер та адміністратор.

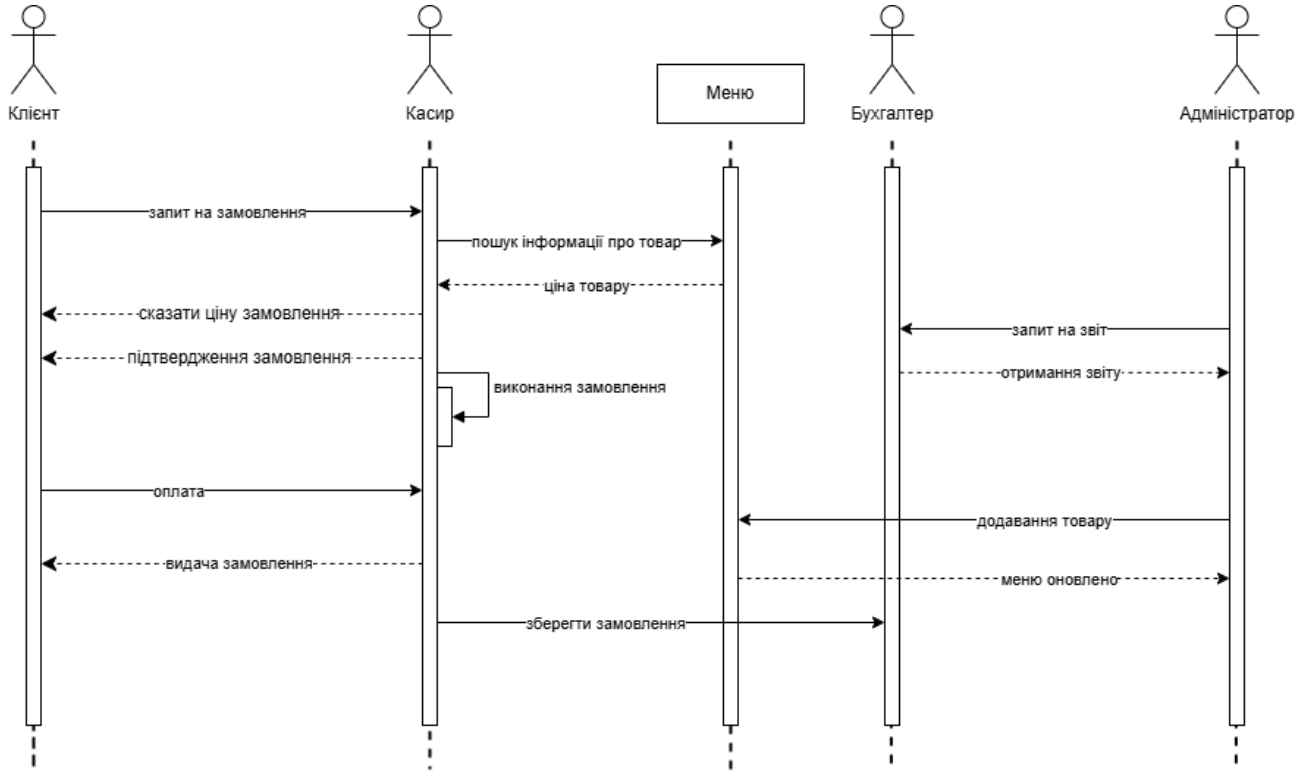

Рис. 1.7 Діаграма послідовності

Взаємодія розпочинається із запиту клієнта щодо створення замовлення, після чого касир здійснює пошук інформації про відповідні товари в базі меню, визначає вартість замовлення та озвучує її клієнту. Після підтвердження замовлення здійснюється його збереження та оплата. Завершальним етапом є видача готового замовлення клієнту. У паралельних сценаріях адміністратор може додавати нові товари до меню, що супроводжується оновленням бази, а бухгалтер, у свою чергу, ініціює запит на формування звіту й отримує його у відповідь.

На рис. 1.8 наведено діаграму активності, яка описує процес створення та обробки звіту. Участь у цьому процесі беруть три суб’єкти: касир, бухгалтер та адміністратор, які послідовно або паралельно виконують відповідні дії у межах своїх повноважень.

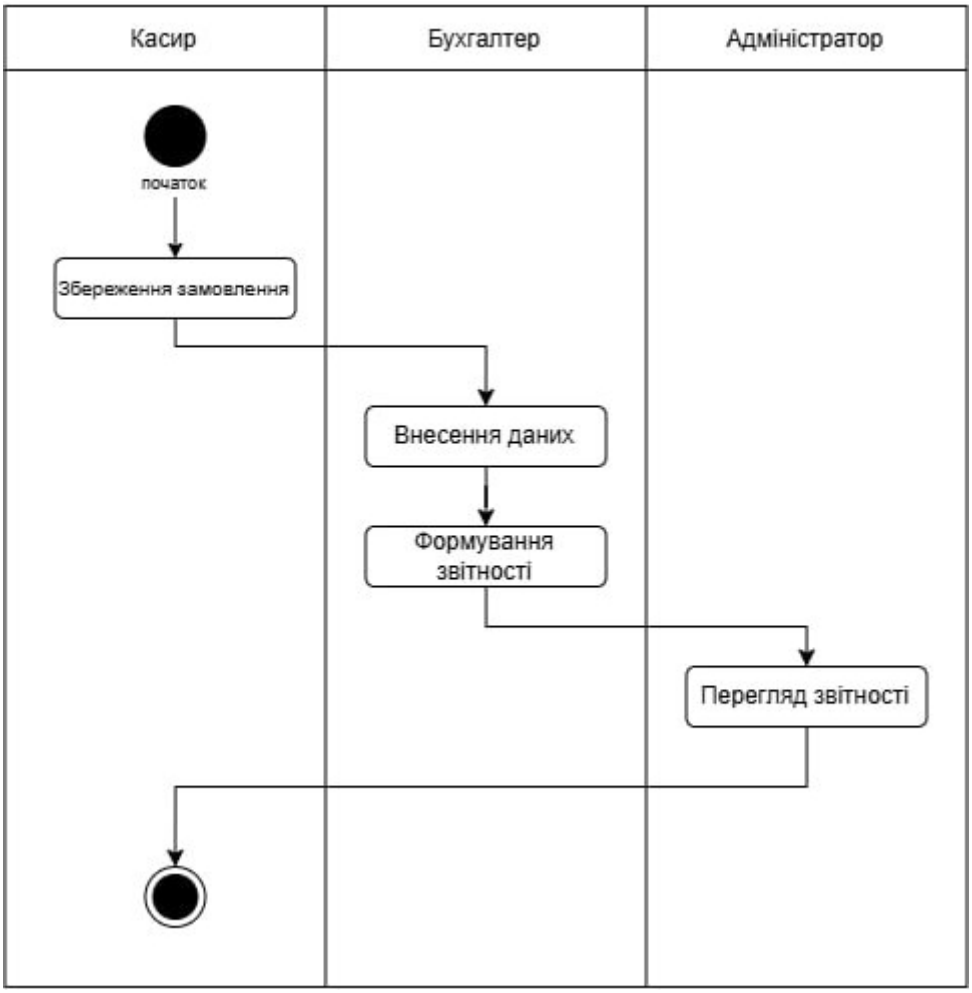

Рис. 1.8 Діаграма активності процесу створення звіту

Процес ініціюється збереженням замовлення касиром – саме ця подія формує основу для подальшого обліку даних. Наступним етапом є внесення відповідної інформації до підсистеми звітності бухгалтером, що включає обробку транзакцій, фіксацію фінансових показників і підготовку сукупних даних. Після цього система переходить до етапу формування звітності, на основі якої адміністратор отримує доступ до підсумкових аналітичних даних через функцію перегляду звіту. Завершується процес переходом у кінцевий стан, який символізує завершення циклу обробки інформації.

Зазначена діаграма дозволяє не лише структурувати бізнес-процес, але й чітко визначити логіку відповідальності між учасниками, що є критичним для правильного розмежування прав доступу, а також для забезпечення точності й

цілісності фінансових та аналітичних даних, які використовуються у процесі управління кав’ярнею.

Також важливо сформулювати структурне подання компонентів системи, що дозволяє визначити сутності, їхні атрибути та взаємозв’язки між ними, а відтак забезпечити цілісність і логічну завершеність архітектури. Такий підхід сприяє виявленню надлишкових або дубльованих елементів, оптимізації структури даних та полегшенню її реалізації в коді. Для цього застосовується діаграма класів, яка є ключовим елементом об’єктно-орієнтованого аналізу та проєктування інформаційних систем. Вона дозволяє не лише візуалізувати статику системи, а й забезпечити формалізований опис логіки функціонування її основних елементів. Діаграма класів дає змогу формально описати структуру системи в термінах об’єктів, які взаємодіють між собою в процесі виконання функцій програмного забезпечення.

На рис. 1.9 подано діаграму класів, яка моделює основні сутності інформаційної системи обліку замовлень у кав’ярні. Усі класи містять відповідні атрибути, що характеризують кожен об’єкт, а також методи (операції), які реалізують поведінку цих об’єктів у контексті бізнес-логіки.

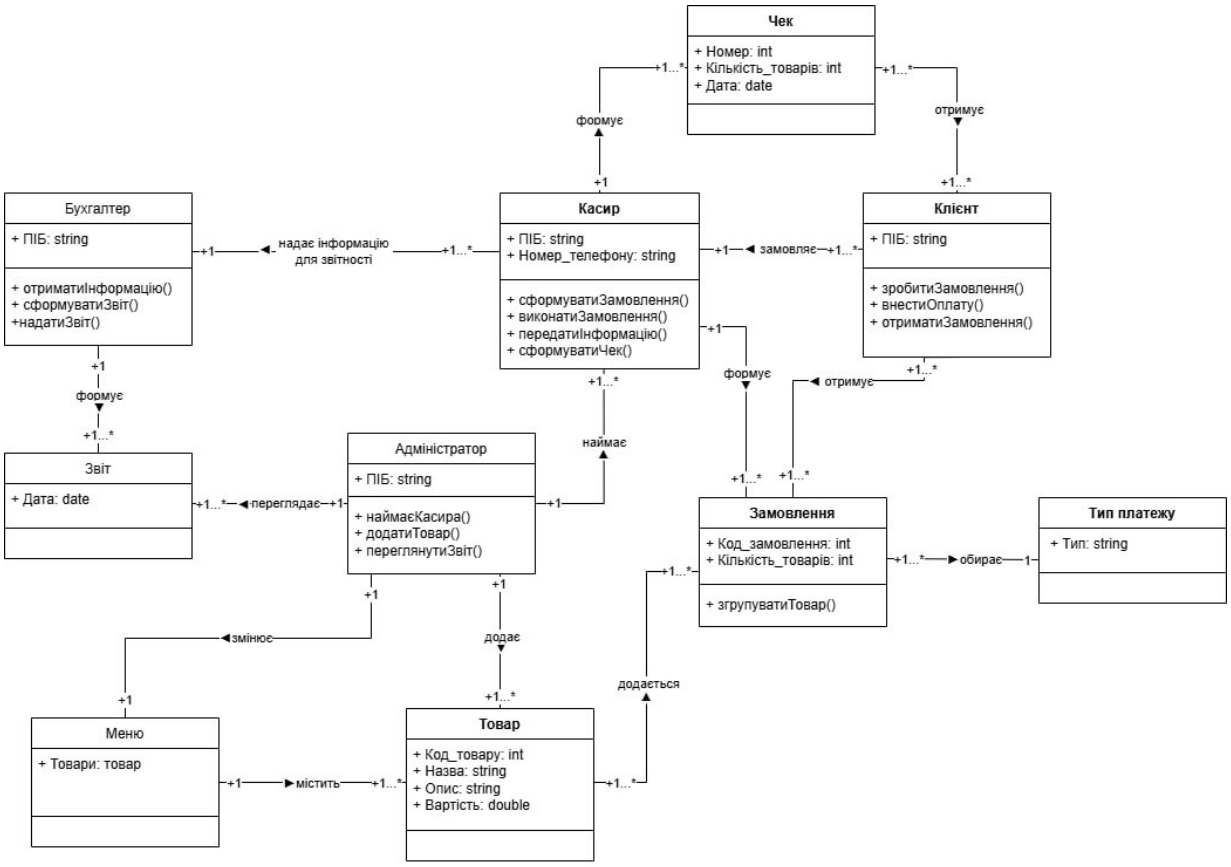

Рис. 1.9 Діаграма класів

У центрі моделі розміщено клас Замовлення, який пов’язує клієнта, касира та об'єкти товарного довідника. Клас містить базові атрибути, зокрема код замовлення та кількість товарних позицій, а також метод для структурування переліку товарів. Клас Клієнт відповідає за ініціацію замовлення, внесення оплати та отримання підтвердження, тоді як Касир забезпечує формування, підтвердження й передачу замовлення, а також створення касових документів. Кожен з класів містить атрибути, необхідні для ідентифікації та логічної взаємодії.

Важливою сутністю є Чек, що формується на основі замовлення і містить інформацію про номер, кількість товарів та дату. Паралельно існує клас Тип платежу, який пов’язаний із замовленням і дозволяє враховувати різні варіанти оплати.

Адміністратор системи реалізує можливість управління товарним довідником і персоналом – його клас містить методи додавання товару, найму

касира та перегляду звітів. У свою чергу, Бухгалтер отримує інформацію про покупки, створює нові звітні документи та передає їх для аналізу. Дані звітності представлені у вигляді окремого класу Звіт.

Товарна структура реалізована через класи Товар та Меню, що дозволяють зберігати інформацію про найменування, опис, вартість і пов’язану категорію. Зв’язок між класами «Товар» та «Меню» вказує на ієрархічну приналежність товарів до певного меню.

Діаграма абстракції, яка відображає логічний рівень подання об'єктів системи з урахуванням їхньої ролі в узагальненій структурі бізнес-процесів. Цей тип діаграми дозволяє перейти від конкретних класів до їхніх узагальнених категорій, що є корисним для виявлення спільних властивостей, спрощення логіки взаємодії та подальшої нормалізації проєктних рішень.

На рис. 1.10 зображено абстрактну модель, яка демонструє основні об’єкти: Клієнт, Касир, Адміністратор, Бухгалтер, Замовлення, Меню, Товар, Чек, Звіт та Тип платежу.

Рис. 1.10 Абстракції

Через об’єкт замовлення реалізується доступ до пов’язаної інформації: до складу позицій меню, обраного типу оплати, створеного чека й подальшого формування звітності. Адміністратор та бухгалтер виступають як функціональні ролі, що забезпечують відповідно управління ресурсами та фінансовий облік. Їхня діяльність не прив’язана до безпосереднього створення замовлення, але є критично важливою для життєвого циклу системи.

У процесі проєктування програмного забезпечення важливу роль відіграє подання загальної структурної архітектури системи у вигляді компонувальної моделі. На рис. 1.11 зображено діаграму компонентів програмного забезпечення системи обліку замовлень у кав’ярні. Вона відображає основні модулі системи та їх взаємодію в межах тришарової архітектури, що включає фронтенд, бекенд і рівень даних.

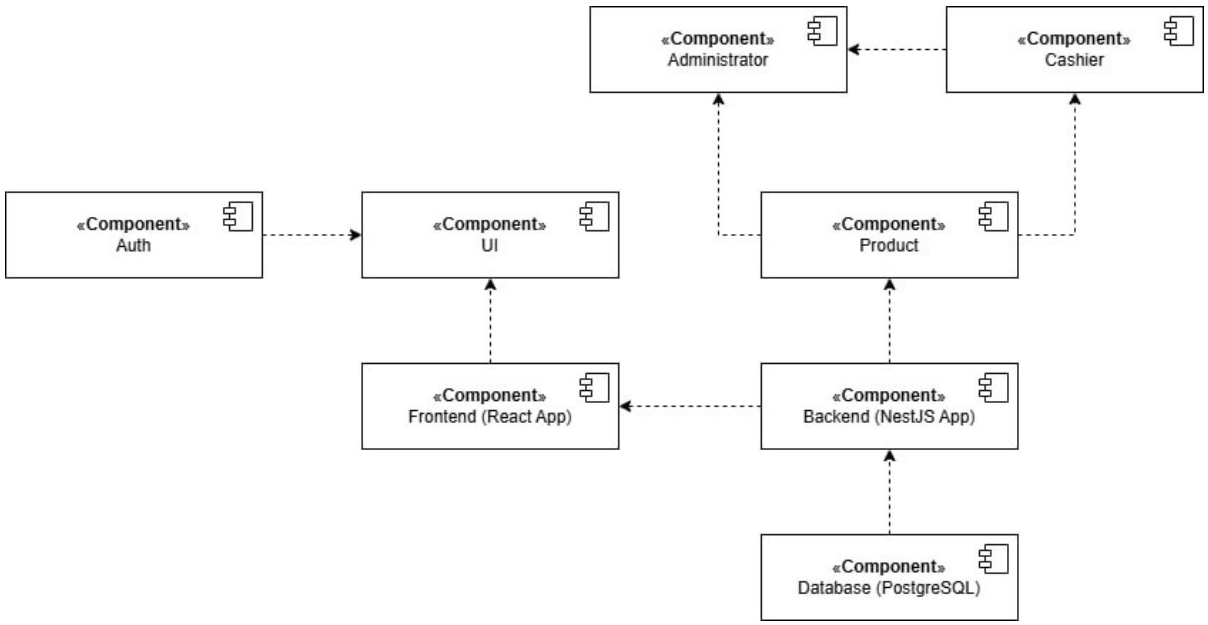

Рис. 1.11 Діаграма компонентів

Користувацький інтерфейс реалізовано у вигляді компонента Frontend, розробленого з використанням React. Цей компонент взаємодіє з UI-модулем, який слугує сполучною ланкою між візуальним інтерфейсом і логікою автентифікації (Auth) та бізнес-компонентами системи. В свою чергу, компонент Backend, реалізований на фреймворку NestJS, обробляє запити користувача, реалізує бізнес-логіку системи, маршрутизує запити до відповідних модулів (Product, Administrator, Cashier) і забезпечує обробку відповідей. Комунікація з базою даних реалізується через компонент Database, побудований на основі PostgreSQL, який відповідає за зберігання інформації про товари, користувачів та транзакції.

Модульність і розділення відповідальностей дозволяє забезпечити гнучкість у підтримці та масштабуванні системи, спрощує тестування окремих

компонентів і сприяє подальшій інтеграції додаткових функцій, таких як модуль звітності, система лояльності чи підтримка мобільного клієнта.

Побудовані вище моделі формують цілісне уявлення про структуру та функціонування програмного забезпечення системи обліку замовлень у кав’ярні, охоплюючи як поведінкові аспекти (через діаграми прецедентів, послідовностей і активностей), так і статичні структури (через діаграми класів, абстракцій і компонентів). Таке моделювання дозволяє глибоко проаналізувати процеси взаємодії між користувачами та системою, виявити критичні залежності, спроектувати логіку обробки даних і визначити архітектуру, придатну до масштабування та підтримки. Представлені схеми є концептуальним підґрунтям для переходу до технічної реалізації, зокрема – написання коду, побудови бази даних, створення API та впровадження інтерфейсу, що буде зручним для адміністратора й касира у щоденній роботі.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Розроблення інформаційної системи неможливе без формалізації структури даних, з якими вона буде працювати. Для цього на етапі проєктування створюється логічна модель, що слугує абстрактним описом інформаційних об’єктів, їхніх властивостей та взаємозв’язків. Така модель не залежить від конкретної системи управління базами даних, але відображає змістову структуру даних, необхідних для функціонування системи. Її основне призначення полягає у виявленні та фіксації всіх значущих сутностей предметної області, встановленні зв’язків між ними та визначенні набору атрибутів, які підлягають збереженню.

У логічній моделі визначаються типи зв’язків (один-до-одного, один-до-багатьох, багато-до-багатьох), уточнюються правила цілісності, унікальності та обмеження на значення полів [17]. На основі такої моделі в подальшому будується фізична реалізація бази даних, тому якість логічного моделювання безпосередньо впливає на надійність, узгодженість і масштабованість усієї інформаційної системи. У випадку автоматизації обліку замовлень у кав’ярні особливо важливо враховувати динаміку змін у даних – наприклад, оновлення товарного асортименту, ротацію персоналу або зміну статусу замовлень – тому логічна структура має бути гнучкою, стійкою до змін і позбавленою надлишковості.

Для візуалізації логічної структури даних традиційно використовується ER-діаграма (діаграма «сутність – зв’язок»), яка дає змогу описати об’єкти системи (сутності), їхні атрибути та взаємозалежності [18]. Такий інструмент дозволяє ще до створення фізичної моделі виявити конфлікти у структурі, усунути дублювання, забезпечити нормалізацію й узгодженість даних. Кожна сутність відображає конкретний тип об’єкта реального світу, наприклад, замовлення, товар, чек, працівника, а атрибути – це властивості, що

характеризують ці об’єкти. Зв’язки між сутностями відображають взаємодію між об’єктами, яка має місце у процесі обслуговування клієнтів, реєстрації операцій, формування звітності тощо.

На рис. 2.1 наведено ER-діаграму логічної моделі бази даних, що побудована для реалізації програмного забезпечення обліку замовлень у кав’ярні. У ній відображено сутності, атрибути та зв’язки між ними, які забезпечують збереження інформації про товари, транзакції, чеки, касирів і типи оплат.

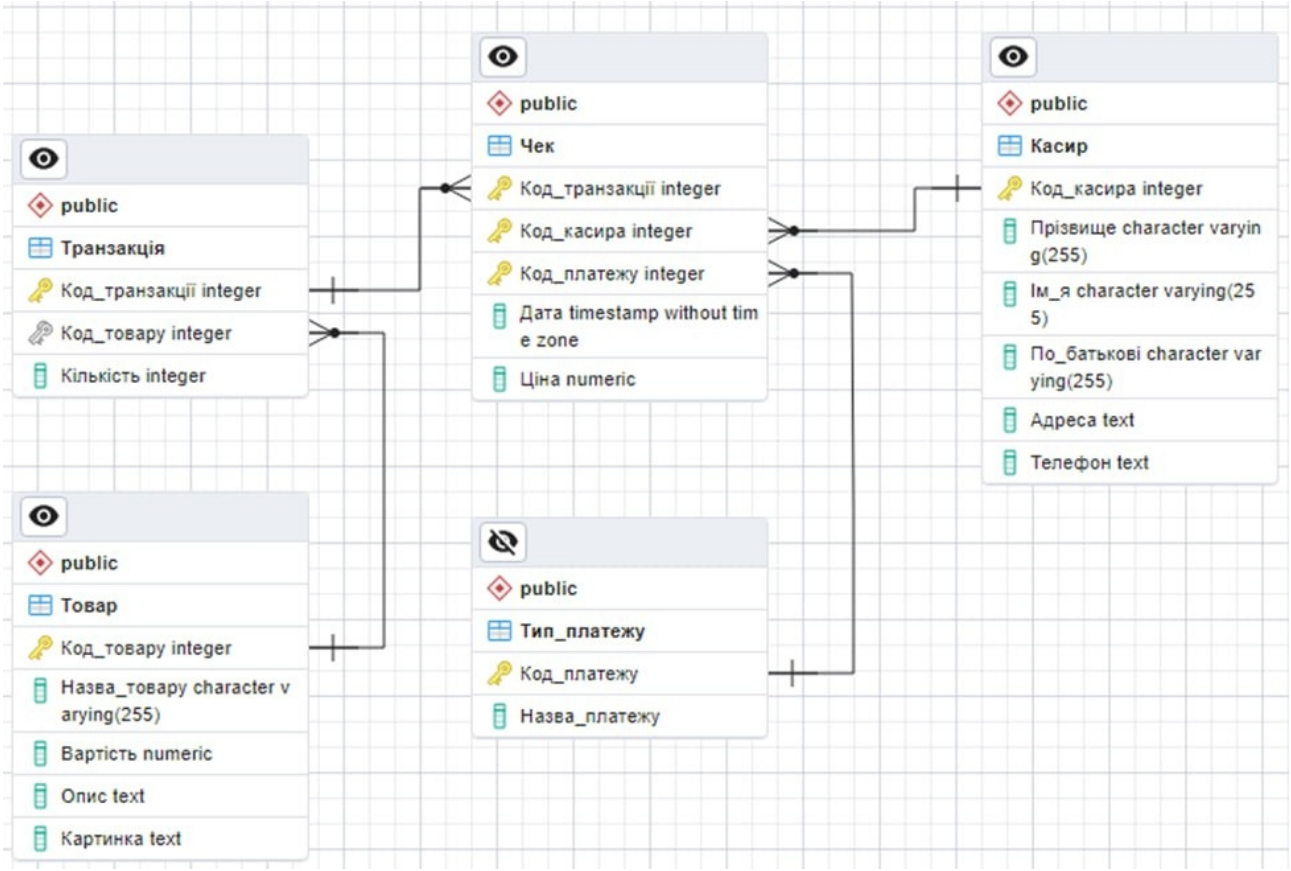

Рис. 2.1 ER-діаграма логічної моделі бази даних

Основу моделі становить сутність «Чек», яка є результатом кожної завершеної операції. Вона містить зовнішні ключі на три пов’язані таблиці: «Транзакція», «Касир» і «Тип платежу», а також фіксує дату створення та загальну суму. Кожен чек прив’язаний до конкретної транзакції, яка, у свою чергу, пов’язана з одним або кількома товарами.

Сутність «Транзакція» слугує проміжною ланкою між чеком і конкретними товарними позиціями. Вона містить зовнішній ключ до таблиці «Товар» і атрибут «Кількість», що дозволяє зберігати дані про замовлену номенклатуру й обсяги. У таблиці «Товар» зберігається перелік продукції, що пропонується кав'ярнею, з такими атрибутами, як назва, вартість, опис та унікальний код товару.

Кожен чек обов'язково пов'язаний із працівником, який обробив замовлення. Для цього використовується таблиця «Касир», що містить персональні дані працівників: прізвище, ім'я, по батькові. Це забезпечує персоніфікацію записів у базі, дозволяє будувати аналітику за змінною ознакою (наприклад, ефективність роботи окремих працівників) і реалізує контроль доступу до фінансових операцій.

Інформація про спосіб оплати міститься у таблиці «Тип платежу», до якої прив'язано відповідний зовнішній ключ із таблиці чеків. Така структура дозволяє системі зберігати різні форми розрахунку та використовувати ці дані у формуванні статистики чи звітності.

Структура бази є компактною, цілісною та придатною для масштабування. Завдяки чітко побудованим зв'язкам і розділенню обов'язків між таблицями модель дозволяє ефективно обробляти касові операції, аналізувати динаміку продажів та зберігати повну історію фінансових транзакцій у кав'ярні.

2.2 Вибір системи управління інформаційною базою

На етапі реалізації фізичної структури бази даних одним із ключових завдань є обґрунтований вибір СУБД, яка забезпечувала б надійність, продуктивність, масштабованість та захист інформації. З урахуванням специфіки предметної області – облік касових операцій, робота з товарними позиціями, фіксація транзакцій та збереження даних про персонал – доцільним було використання PostgreSQL як основної СУБД.

Вибір саме PostgreSQL обумовлений її технічними характеристиками та відповідністю вимогам інформаційної системи. Ця об’єктно-реляційна СУБД із відкритим вихідним кодом підтримує транзакції з дотриманням властивостей ACID, забезпечує механізми перевірки цілісності, підтримує зовнішні ключі, унікальність, індексацію, збережені процедури, тригери та інші засоби, необхідні для надійного зберігання фінансових даних [19]. Система також відповідає міжнародним стандартам SQL і забезпечує високу сумісність із сучасними інструментами розробки.

Однією з практичних переваг PostgreSQL є її гнучкість – вона дозволяє створювати користувацькі функції, типи даних і розширення, що дає змогу адаптувати логіку обробки даних до специфіки внутрішніх процесів кав’ярні. Підтримка паралельного виконання запитів забезпечує високу продуктивність системи, особливо під час вибірки великої кількості чеків або підготовки звітів.

Важливим фактором є й те, що PostgreSQL доступна на умовах вільної ліцензії, що суттєво зменшує загальні витрати на впровадження системи [20]. Платформа підтримує як локальне, так і хмарне розгортання, що дозволяє адаптувати її до технічних можливостей замовника. Завдяки вбудованим засобам керування правами доступу, документуванням, резервним копіям та підтримці шифрування, ця СУБД відповідає вимогам щодо безпечного зберігання конфіденційних даних.

Використання PostgreSQL у системі обліку замовлень дозволило побудувати логічно зв’язану та структуровану базу даних із підтримкою сутностей «чек», «товар», «касир», «тип платежу» тощо. Реалізовані транзакційні механізми гарантують узгодженість даних навіть за умов одночасного доступу кількох користувачів, що сприяє стабільній роботі системи та забезпечує основу для її подальшого розвитку.

2.3 Створення інформаційної бази

З урахуванням обраної архітектури програмного забезпечення, функціональних потреб, передбачуваного навантаження та особливостей середовища експлуатації, було вирішено реалізувати централізовану інформаційну базу, яка функціонує в межах серверної частини бекенд-застосунку на платформі PostgreSQL. База даних зберігається на окремому екземплярі серверного середовища, до якого мають доступ лише авторизовані користувачі через інтерфейс клієнтської частини програми, реалізованої на React з використанням механізмів авторизації та обмеження прав доступу.

Модель бази даних побудована на основі реляційної структури з дотриманням принципів логічної нормалізації до третьої нормальної форми. Кожна сутність предметної області – товар, касир, чек, транзакція, тип платежу – винесена в окрему таблицю з унікальним первинним ключем. Зв’язки типу «один до багатьох» реалізуються через зовнішні ключі, що дозволяє підтримувати цілісність та узгодженість даних у процесі виконання транзакцій. Такий підхід дозволяє уникнути дублювання інформації, пришвидшує доступ до записів і мінімізує ймовірність логічних суперечностей.

Створення таблиць та визначення обмежень реалізовано засобами SQL-скриптів, що були сформовані на основі раніше побудованої логічної моделі. Усі таблиці мають чітко описані типи атрибутів, обмеження на порожні значення, унікальність і зовнішні залежності. Наприклад, таблиця «Чек» включає поля для дати, загальної суми, способу оплати та посилання на транзакцію і касира. У таблиці «Транзакція» зберігається перелік товарів із зазначенням кількості, що дає змогу формувати складні замовлення. Пов’язані товари описані в таблиці «Товар», яка містить код, назву, опис та вартість одиниці продукції. Таблиця «Касир» забезпечує зберігання інформації про працівників, які здійснюють обслуговування клієнтів, а «Тип платежу» дозволяє гнучко управляти методами розрахунку.

У рамках безпеки реалізовано базову рольову модель: касир має обмежений доступ лише до модуля каси, тоді як адміністратор отримує права на перегляд, додавання та редагування товарів, а також контроль за транзакціями. Права доступу реалізуються через механізми авторизації на стороні бекенду та обмеження SQL-запитів за допомогою ORM-рівня (Prisma), що додатково убезпечує пряме втручання в базу.

Інформаційна база має підтримку регулярного резервного копіювання. Для цього налаштоване створення щоденних бекапів із використанням утиліти pg_dump, які зберігаються на ізольованому носії або відправляються на резервний сервер у хмарі. Це дозволяє забезпечити відновлення даних у разі критичних збоїв або втрати інформації, пов’язаної з людським фактором. У разі потреби адміністратор має змогу повернути систему до попереднього стабільного стану без порушення структури та залежностей між таблицями.

Хоч інформаційна база наразі реалізована у вигляді централізованої моделі, її структура передбачає можливість подальшого масштабування. Наприклад, за умови розширення функціоналу системи (контролю складу, системи лояльності або багатоточкової мережі кав’ярень), база даних може бути адаптована до нових вимог без необхідності кардинальної перебудови логіки. Це стало можливим завдяки початково спроєктованій структурі з чітким розділенням функціональних блоків.

3 Прикладне програмне забезпечення

3.1 Організаційна структура програмного забезпечення

Одним із важливих аспектів розробки програмного забезпечення є його раціональне логічне структурування. Для відображення внутрішньої організації системи та впорядкування її складових використовується діаграма пакетів (package diagram), яка належить до структурних засобів моделювання в нотації UML. Така діаграма дозволяє представити програмну систему у вигляді взаємопов’язаних логічних компонентів – пакетів, кожен з яких виконує визначене функціональне призначення. Основне призначення діаграми пакетів полягає у впорядкуванні модулів системи за функціональними ознаками, що сприяє кращому розумінню загальної архітектури застосунку, полегшує орієнтацію в коді та забезпечує чітке розмежування відповідальностей. Кожен пакет, умовно зображений у вигляді прямокутника, містить логічно пов’язані класи, інтерфейси або підпакети, які разом реалізують певну підсистему.

На рис. 3.1 наведено діаграму пакетів, яка ілюструє загальну структурну організацію програмного забезпечення, реалізованого для автоматизованого обліку замовлень у кав’ярні. Архітектура системи побудована з дотриманням принципів модульності, що забезпечує розмежування відповідальностей між компонентами та сприяє підтримуваності коду. Кожен програмний пакет виконує окрему функцію в межах загальної логіки застосунку, а взаємозв’язки між ними чітко регламентовані та реалізовані через визначені інтерфейси.

Центральне місце в архітектурі займає модуль обробки бізнес-логіки, що реалізований засобами фреймворку NestJS. Саме цей компонент відповідає за виконання основних дій, пов’язаних із реєстрацією замовлень, проведенням транзакцій, перевіркою доступу користувачів і координацією обміну даними між інтерфейсом і базою. Модуль забезпечує перевірку вхідних запитів, реалізацію бізнес-правил і гарантує логічну цілісність дій, що відбуваються в системі.

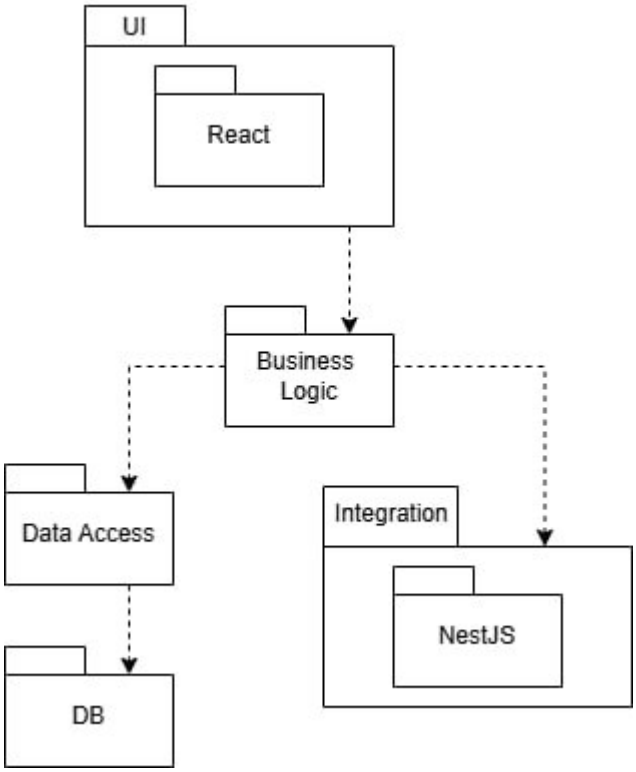

Рис. 3.1 Діаграма пакетів

Інтерфейс користувача реалізовано у вигляді окремого клієнтського модуля на основі React, який відповідає за візуалізацію даних, збір інформації від користувача, навігацію між формами та відображення результатів операцій. Комунікація між клієнтським і серверним боком реалізована через REST API, що підтримує відокремлення інтерфейсного шару від прикладної логіки та дозволяє змінювати вигляд інтерфейсу без втручання в серверну частину.

Компонент доступу до бази даних, інтегрований у бекенд, побудовано з використанням ORM Prisma, що забезпечує ефективну роботу з СУБД PostgreSQL. Завдяки цьому реалізовано чітке розмежування між логікою обробки запитів і фізичним розташуванням даних. Компонент відповідає за створення, оновлення, читання й видалення записів у таблицях бази даних, зберігаючи при цьому узгодженість і стабільність транзакцій.

Підсистема автентифікації та авторизації реалізована у вигляді окремого модуля, який відповідає за перевірку облікових даних користувачів, формування токенів доступу та розмежування прав відповідно до ролей (адміністратор, касир). Всі виклики до захищених маршрутів проходять через

відповідні middleware-перевірки, що гарантує безпеку доступу до конфіденційної інформації.

Залежності між модулями показані у вигляді стрілок, що відображають напрямок звернення одного пакета до іншого. Така побудова дозволяє знизити зв’язність між компонентами та забезпечує гнучкість при модифікації або масштабуванні окремих частин системи. Поділ системи на пакети відповідає принципам структурного проєктування, сприяє повторному використанню коду, пришвидшує процес розробки та тестування, а також полегшує адаптацію системи до нових умов роботи або функціональних змін.

3.2 Вибір інструментарію для створення ППЗ

Розробка сучасних веборієнтованих інформаційних систем вимагає застосування перевірених, продуктивних та взаємосумісних інструментів, які забезпечують ефективну реалізацію як серверної, так і клієнтської частини застосунку. При створенні системи було використано стек технологій, орієнтований на побудову масштабованої багаторівневої архітектури з чітким розмежуванням ролей між модулями.

У якості серверної частини (бекенду) обрано фреймворк NestJS, що працює поверх середовища Node.js і використовує мову програмування TypeScript. NestJS є сучасним інструментом для побудови структурованих серверних застосунків, що підтримує принципи об’єктно-орієнтованого програмування, модульності та інверсії залежностей [21]. Його архітектура дозволяє впроваджувати багаторівневі обробники, контролери, сервіси та репозиторії, забезпечуючи високу гнучкість при побудові бізнес-логіки та спрощуючи супровід проєкту. Вибір TypeScript як основної мови обумовлений її статичною типізацією, що дозволяє виявляти помилки ще на етапі компіляції, та високою сумісністю з JavaScript-екосистемою.

Для побудови клієнтської частини застосунку використано React – одну з найпопулярніших JavaScript-бібліотек, орієнтованих на створення

інтерфейсів користувача [22]. React дозволяє ефективно реалізувати компонетну модель інтерфейсу, що забезпечує повторне використання елементів, їх ізоляцію та керування станом. Це особливо актуально в умовах динамічного оновлення даних у відповідь на дії користувача. React також забезпечує просту інтеграцію з інтерфейсами REST API, що реалізовані на серверній частині через NestJS.

В якості системи керування базами даних застосовано PostgreSQL – надійну об’єктно-реляційну СУБД з відкритим вихідним кодом, яка забезпечує високу продуктивність, масштабованість і відповідність стандартам SQL. PostgreSQL підтримує транзакції, зовнішні ключі, унікальні обмеження, перевірки цілісності, а також розширені засоби роботи з JSON і масивами. Взаємодія між сервером і базою даних реалізована за допомогою Prisma ORM – сучасного інструменту об’єктно-реляційного відображення, який автоматично генерує типізовані моделі для роботи з даними в межах TypeScript-застосунків. Prisma дозволяє уникати помилок, пов’язаних із ручним написанням SQL-запитів, та суттєво спрощує розробку й супровід доступу до бази.

Застосування даного стеку технологій дало змогу реалізувати програмну систему за принципами розділення обов’язків: клієнтський рівень відповідає за візуалізацію та взаємодію з користувачем, серверний – за обробку бізнес-логіки та взаємодію з базою даних, а рівень збереження – за надійне та узгоджене зберігання інформації. Архітектурна модель не лише відповідає сучасним вимогам до якості ПЗ, але й забезпечує гнучкість для подальшого розширення функціоналу системи без порушення існуючих залежностей між її компонентами.

Для побудови UML-діаграм, які описують архітектуру програмного забезпечення, використано інструмент draw.io. Ця платформа надала зручні засоби для створення структурованих графічних моделей, що охоплюють взаємозв’язки між компонентами системи, сценарії використання, логіку взаємодії модулів та розміщення елементів на фізичних пристроях. Усі

діаграми виконано згідно з нотацією UML 2.x, що забезпечує дотримання загальноприйнятих стандартів моделювання та полегшує інтерпретацію архітектурних рішень іншими учасниками команди розробки.

3.3. Алгоритмізація та програмування програмних модулів

Алгоритмізація є невід’ємним етапом процесу інженерії програмного забезпечення, під час якого здійснюється формалізація логіки функціонування системи у вигляді послідовності обчислювальних дій, спрямованих на реалізацію заданих функціональних вимог. Побудова алгоритмів забезпечує не лише детерміновану поведінку програмного застосунку в умовах типових і нетипових сценаріїв, але й створює структурну основу для модульної реалізації окремих компонентів. У межах розробки інформаційної системи обліку замовлень у кав’ярні було спроєктовано та формалізовано набір алгоритмічних процедур, що охоплюють базові сценарії взаємодії користувача з системою, включаючи автентифікацію, реєстрацію замовлень, генерацію фіскальних чеків, додавання товарних позицій до замовлення та перегляд історії транзакцій.

На рис. 3.3 наведено блок-схему алгоритму авторизації користувача, яка реалізує один із критично важливих процесів доступу до системи.

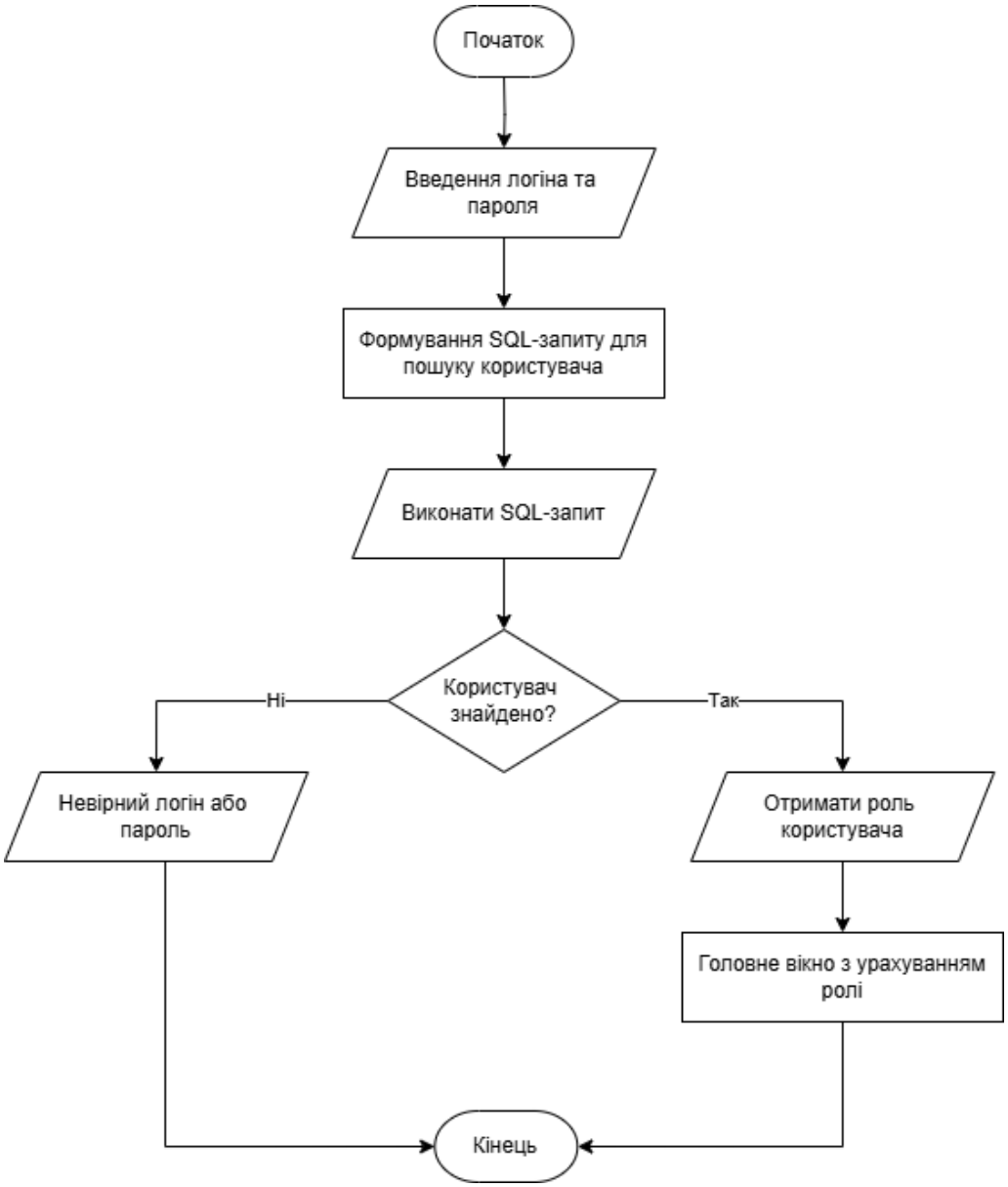

Рис. 3.3 Блок-схема алгоритму авторизації користувача

Процес авторизації розпочинається з введення користувачем логіна та пароля. Далі формується SQL-запит для перевірки наявності такого запису в базі даних. Запит виконується на сервері, після чого система аналізує, чи знайдено відповідного користувача. У випадку, якщо користувача не виявлено, на екран виводиться повідомлення про помилку автентифікації, і виконання алгоритму завершується. Якщо обліковий запис підтверджено, система зчитує відповідну роль користувача, після чого відкривається головне вікно програми з урахуванням призначених прав доступу.

На рис. 3.4 подано блок-схему, що відображає алгоритм додавання нового касира до системи обліку замовлень. Процедура реалізується користувачем із правами адміністратора та належить до адміністративного функціоналу системи. Алгоритм забезпечує створення запису касира з перевіркою валідності введених даних.

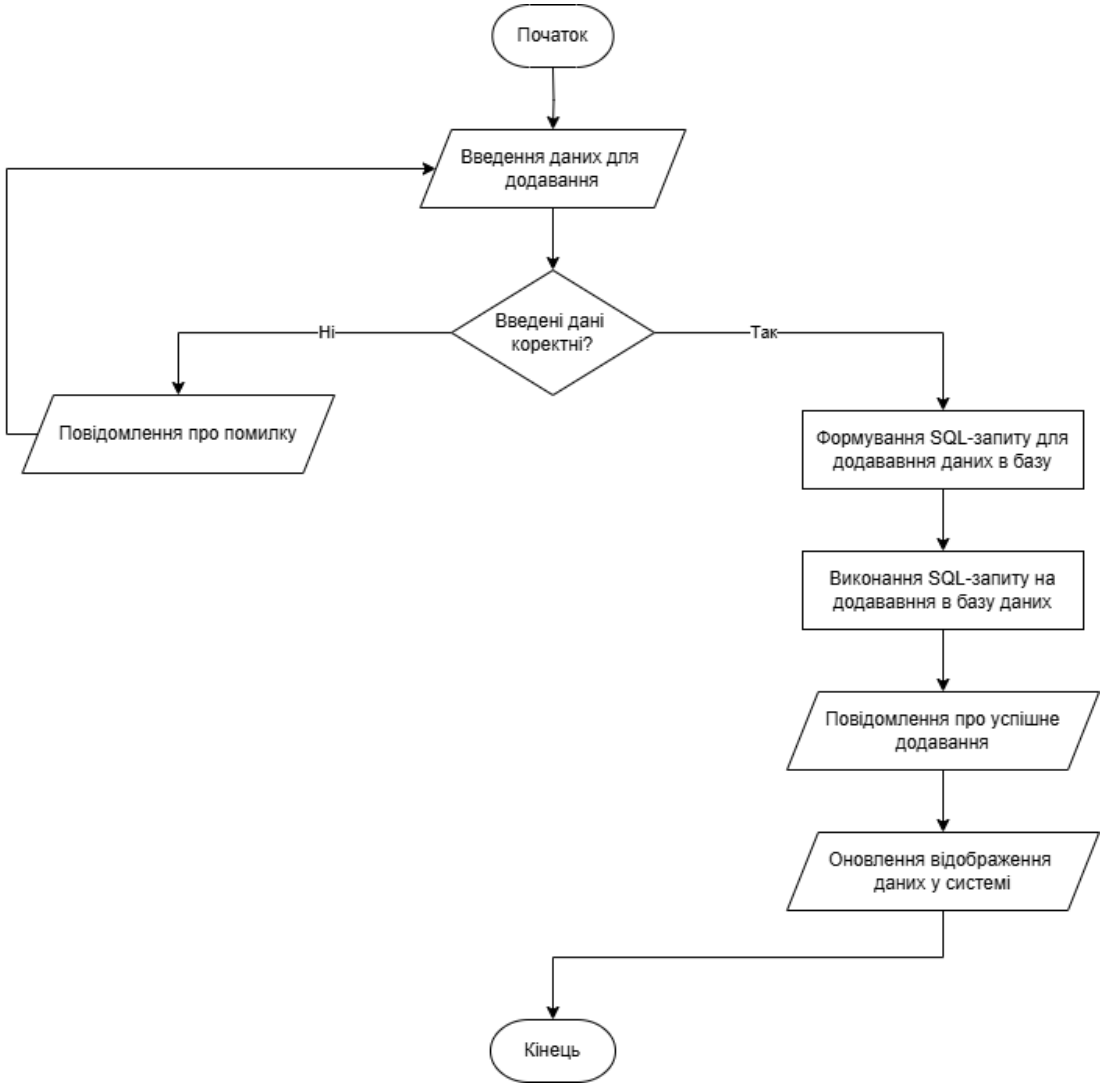

Рис. 3.4 Блок-схема алгоритму додавання касира

Процес починається з ініціалізації форми введення, у яку вносяться основні персональні дані працівника (ПІБ, контактна інформація). Система виконує валідацію полів: перевіряється їх заповненість та відповідність формату. У разі виявлення помилки користувач отримує повідомлення з коментарем і повертається до редагування.

Після успішної перевірки формується SQL-запит на додавання запису до таблиці «Касир», який передається на сервер бази даних. Після підтвердження операції система оновлює інтерфейс, відображаючи нового користувача в списку працівників.

Рис. 3.5 висвітлює блок-схему алгоритму створення замовлення в інформаційній системі обліку замовлень у кав’ярні. Цей процес є базовим з погляду щоденного функціонування касової системи та передбачає послідовне виконання кроків, що дозволяють зафіксувати нову транзакцію з урахуванням вибраного касира, товару та способу оплати.

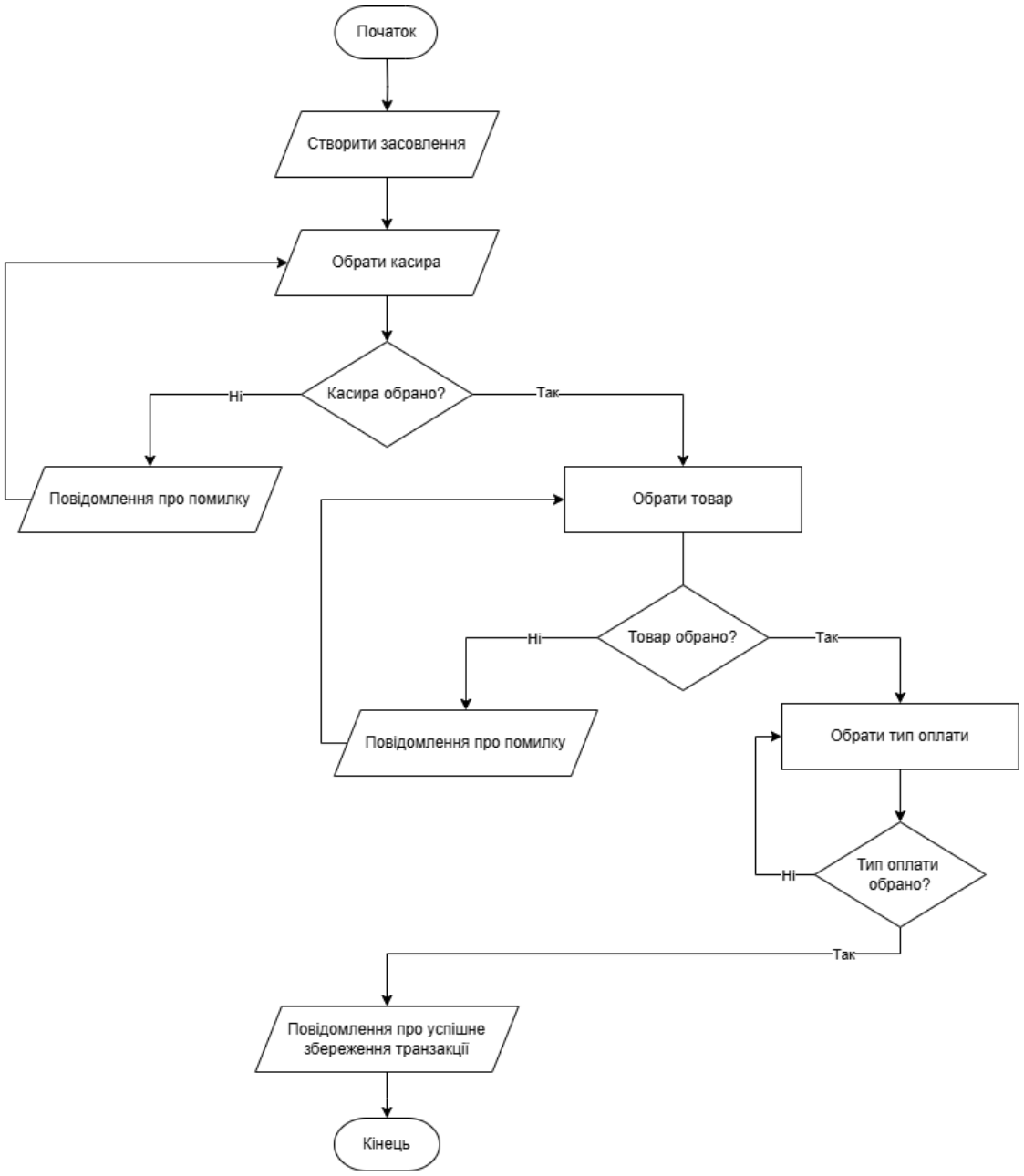

Рис. 3.5 Блок-схема алгоритму створення замовлення

Процедура починається з ініціалізації створення замовлення через клієнтський інтерфейс. На першому етапі система пропонує обрати касира, який обслуговує поточну операцію. Якщо користувач не вказує жодного працівника, виводиться повідомлення про помилку.

У разі, якщо касира обрано, наступним кроком є вибір товару зі списку доступної продукції. Перевіряється, чи справді обрано хоча б одну товарну позицію – якщо ні, користувач отримує відповідне повідомлення та має можливість повернутися до цього кроку. Після успішного вибору товару відбувається вибір способу оплати: готівкою, карткою або іншим підтримуваним методом. Як і на попередніх етапах, у разі відсутності вибору система повідомляє про помилку та припиняє подальше виконання.

Після успішного заповнення всіх обов’язкових полів – касир, товар, тип оплати – система переходить до завершального етапу: створення SQL-запиту на додавання нового замовлення до бази даних. Після виконання цього запиту користувачу виводиться повідомлення про успішне збереження транзакції, і замовлення вважається сформованим.

Завдяки послідовному виконанню перевірок та обробці виняткових ситуацій описаний алгоритм забезпечує стійкість системи до помилок введення, підвищує її надійність та покращує користувацький досвід. Така реалізація дозволяє зберігати повноту й цілісність даних, водночас забезпечуючи простий та зрозумілий механізм створення замовлень у реальних умовах роботи кав’ярні.

4 Рекомендації щодо впровадження та експлуатації системи

4.1 Тестування системи

Розроблене програмне забезпечення для автоматизованого обліку замовлень у кав’ярні реалізоване як повноцінний клієнт-серверний застосунок із чітко структурованими модулями для інтерфейсної, логічної та серверної частин. У процесі проєктування було враховано особливості щоденної роботи кав’ярні, вимоги до швидкого обслуговування клієнтів, мінімізації людського фактору та простоти використання системи для касирів і адміністратора. Програмна архітектура побудована за трирівневою моделлю, де React відповідає за відображення користувацького інтерфейсу, NestJS реалізує прикладну логіку, а PostgreSQL – зберігання всіх бізнес-даних. Для забезпечення безпеки доступу до функціоналу кожен користувач повинен пройти процедуру автентифікації.

На рис. 4.1 зображено форму авторизації, яка є стартовою точкою взаємодії з системою. Інтерфейс форми побудований з урахуванням принципів зручності: два текстові поля дозволяють ввести логін та пароль, а кнопка «Увійти» ініціює перевірку введених даних.

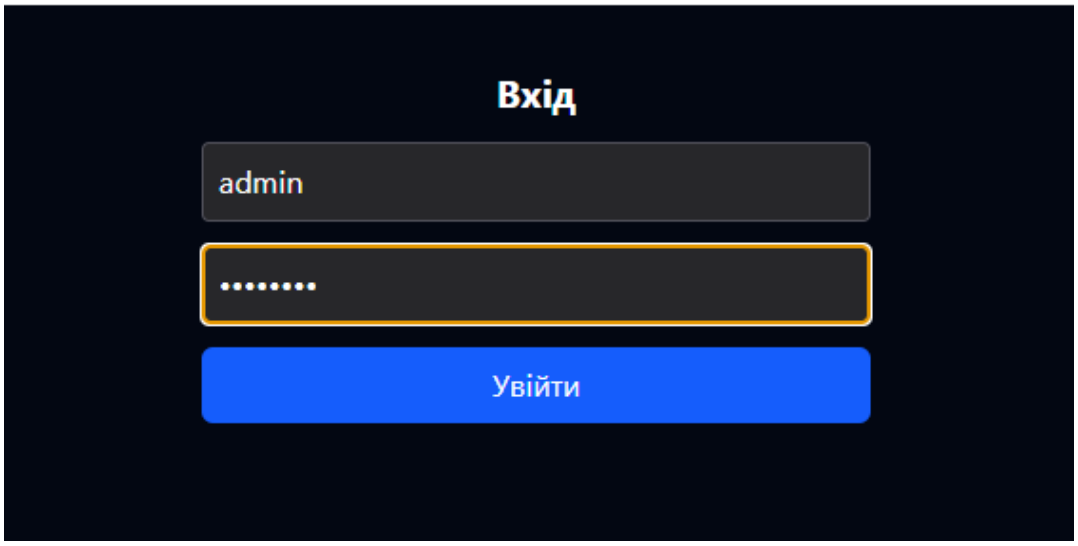

Рис. 4.1 Форма авторизації

У системі підтримуються ролі адміністратора й касира, що дозволяє розмежувати доступ до різних модулів системи. Форма відрізняється лаконічним дизайном, контрастними кольорами та акцентом на ключові елементи взаємодії. У разі помилки користувач отримає відповідне повідомлення, що сприяє безпечному використанню системи та виключає несанкціонований доступ.

Після успішної авторизації з обліковим записом адміністратора користувачу відкривається доступ до головного меню керування системою. У цьому режимі відображаються всі основні функціональні блоки, необхідні для обслуговування та адміністрування системи обліку замовлень. На рисунку 4.2 наведено інтерфейс меню адміністратора, що дозволяє здійснювати повний контроль над інформаційною базою.

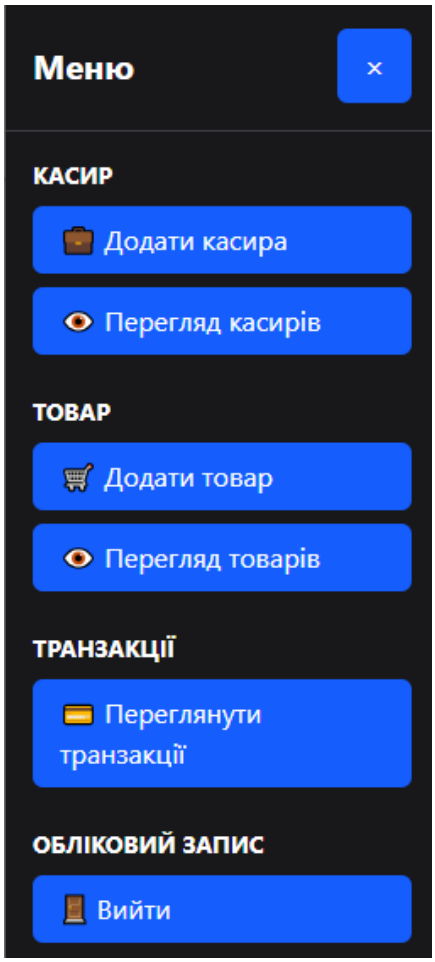

Рис. 4.2 Головне меню адміністратора системи

Меню структуроване за категоріями, що відповідають ключовим об’єктам системи: касири, товари, транзакції та обліковий запис. У розділі «Касир» передбачено можливість додавання нового касира та перегляду вже зареєстрованих у системі працівників. Розділ «Товар» забезпечує аналогічний функціонал для роботи з асортиментом продукції – адміністратор може додавати нові позиції або переглядати наявні. Окремий блок присвячений транзакціям, де реалізовано можливість перегляду історії операцій, що дає змогу здійснювати базовий аудит діяльності. Нижня частина меню містить кнопку для завершення сеансу роботи й виходу з облікового запису.

У режимі адміністратора користувач має можливість додавати нових касирів до системи через відповідну функціональну форму. Цей механізм дозволяє оперативно реєструвати нових працівників, забезпечуючи повноцінне ведення обліку персоналу, що бере участь у виконанні замовлень. На рис. 4.3 наведено інтерфейс форми додавання касира, який відображається після вибору відповідного пункту з адміністративного меню.

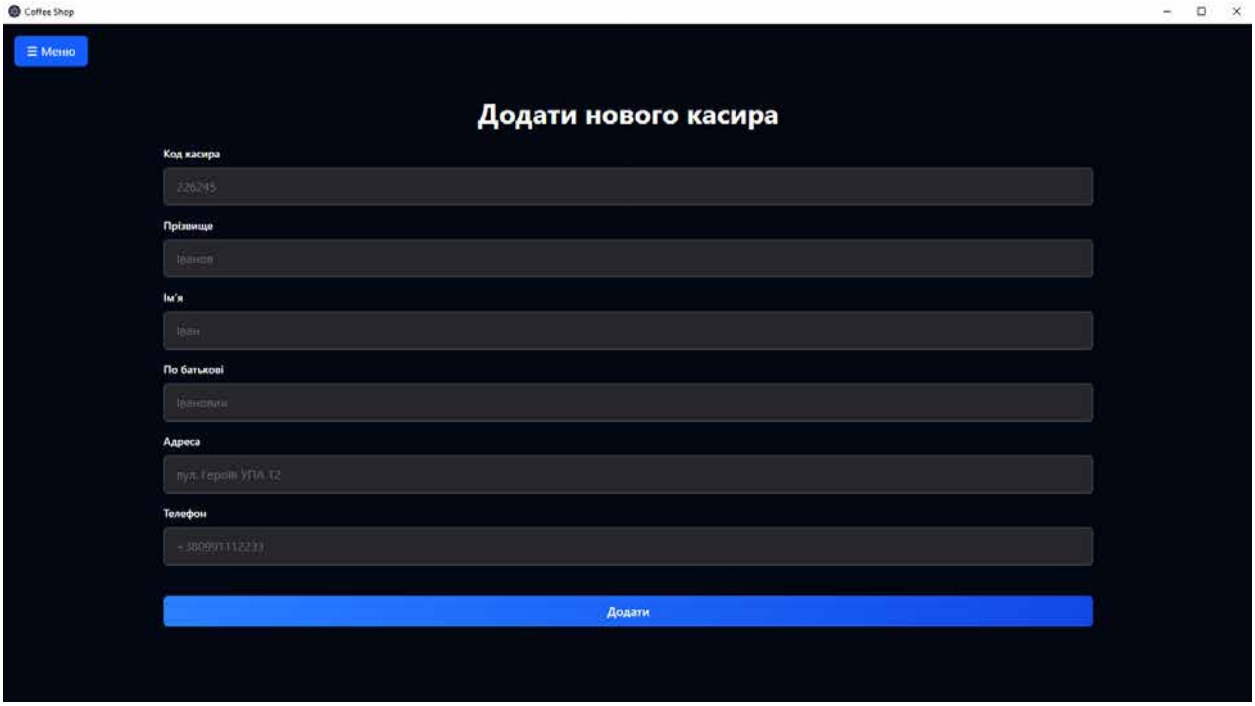

Рис. 4.3 Форма для додавання нового касира

Форма містить набір полів для введення обов’язкових персональних даних: коду касира, прізвища, імені, по батькові, адреси проживання та номера

телефону. Усі поля реалізовані як текстові, з інтуїтивно зрозумілими підказками, що допомагає уникнути помилок при введенні. Після заповнення всієї інформації користувач має можливість зберегти дані, натиснувши кнопку «Додати». У разі успішного завершення операції відображається повідомлення про збереження, а форма автоматично очищується для введення нової інформації.

Важливо зазначити, що під час тестування перевірялася наявність валідації введених даних, обробка порожніх полів і помилок підключення до серверної частини. Алгоритм додавання виконується коректно, а введена інформація надійно фіксується у базі даних через відповідний POST-запит до API. Форма демонструє високу зручність у використанні, що дозволяє адміністратору ефективно керувати кадровою інформацією в межах інформаційної системи кав’ярні.

Також передбачено можливість перегляду всієї актуальної інформації про зареєстрованих касирів. Така опція є необхідною для візуального контролю, зручного пошуку працівників, а також оперативного доступу до редагування їхніх даних. На рис. 4.4 зображено інтерфейс списку касирів, який відкривається після вибору відповідного пункту в меню керування.

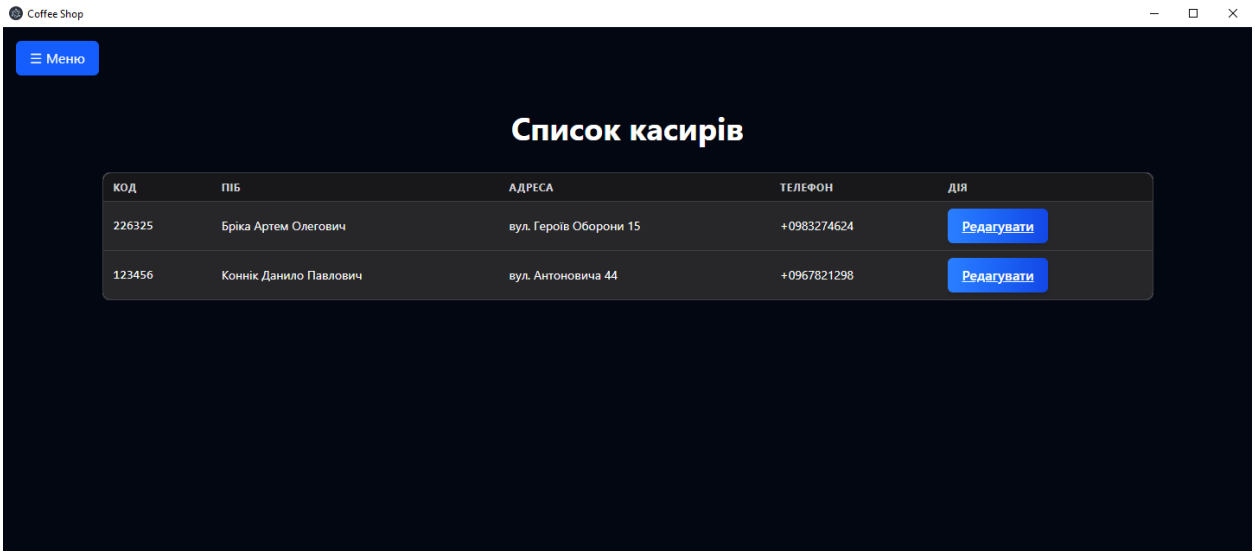

Рис. 4.4 Відображення списку касирів у системі

Виведена таблиця містить п’ять основних колонок: унікальний код касира, повне ім’я (ПІБ), адресу проживання, номер телефону та стовпець для

виконання дій. Дані відображаються у впорядкованому вигляді, що полегшує навігацію навіть за наявності великої кількості записів. Кожен рядок таблиці містить кнопку «Редагувати», натискання на яку ініціює завантаження відповідної форми редагування даних вибраного користувача.

У ході тестування перевірено коректність завантаження інформації з бази даних, швидкість відображення, адаптивність верстки та інтерактивність кнопок. Таблиця відображає лише дійсні записи без дублювання, а усі поля мають вирівняну структуру, що забезпечує зручне візуальне сприйняття. Загалом, функціонал перегляду списку касирів реалізований відповідно до очікувань і демонструє надійну взаємодію з бекендом, дозволяючи адміністратору ефективно контролювати кадровий облік.

На рис. 4.5 висвітлено інтерфейс редагування даних касира, який відкривається після натискання кнопки «Редагувати» в таблиці списку персоналу.

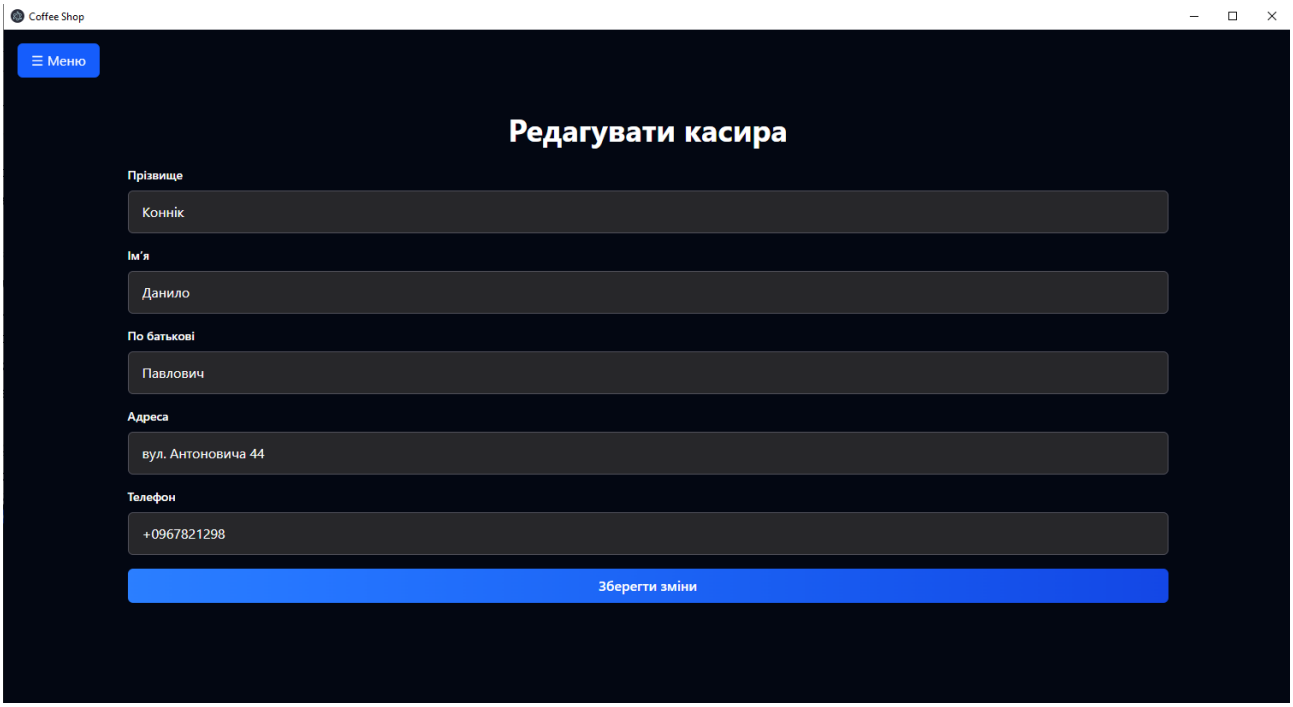

Рис. 4.5 Редагування інформації касира

Форма містить набір текстових полів для внесення змін до персональних даних працівника. У нижній частині вікна розташовано кнопку «Зберегти зміни», яка ініціює процедуру оновлення запису в базі даних.

У процесі тестування перевірено відповідність значень, що підтягуються у поля форми, фактичним даним обраного запису, правильність їх оновлення після редагування, реакцію системи на введення некоректних або порожніх значень, а також обробку виключень при збереженні змін.

Інтерфейс забезпечує логічне групування елементів, що робить взаємодію інтуїтивною навіть для недосвідчених користувачів. Усі внесені зміни зберігаються без затримки, а повторне відкриття форми підтверджує актуалізацію даних. Таким чином, функціональність редагування касирів успішно пройшла тестування та забезпечує коректну роботу згідно з вимогами до системи.

У складі адміністративного функціоналу також передбачено можливість додавання нових товарних позицій до системи, що є необхідною умовою для підтримання актуального асортименту продукції кав'ярні. Відповідний інтерфейс дозволяє швидко внести нову позицію до бази, вказавши ключові характеристики продукту. На рис. 4.6 зображено форму додавання товару, яка є доступною після вибору відповідного пункту в меню адміністратора.

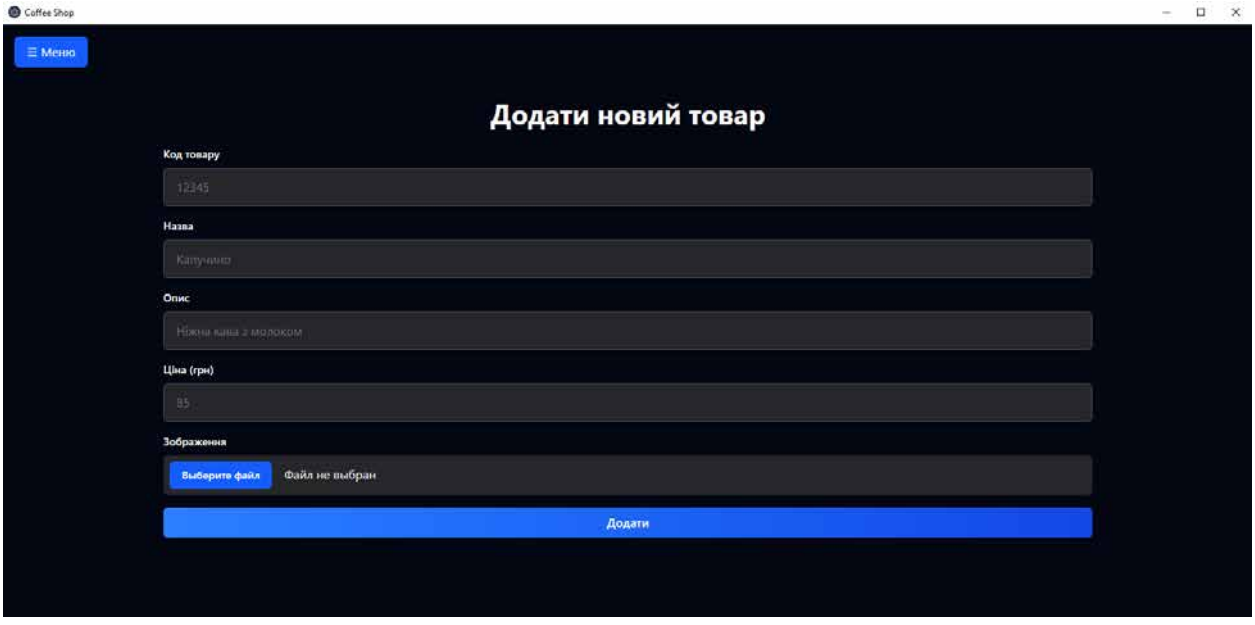

Рис. 4.6 Форма додавання нового товару

Інтерфейс містить поля для введення коду товару, його назви, короткого опису, ціни у гривнях, а також опцію додавання зображення. Поля мають підказки та підтримують ручне введення, що дозволяє адміністратору

оперативно додати як напої, так і супутні товари. Завантаження фото реалізоване через кнопку вибору файлу, що забезпечує подальше візуальне представлення продукту у клієнтському інтерфейсі. Після заповнення форми та натискання кнопки «Додати» дані передаються на сервер, де зберігаються у відповідній таблиці бази даних.

Під час тестування перевірено коректність обробки введених значень, відображення повідомлень про успішне збереження, а також поведінку форми у разі помилкового або неповного заповнення. Інтерфейс виявився зручним для використання, а логіка валідації й обробки запитів працює стабільно. Реалізоване рішення дозволяє адміністратору підтримувати актуальність меню та гнучко реагувати на зміну асортименту в межах бізнес-процесів кав'ярні.

Функціональність адміністратора також включає модуль перегляду усіх наявних товарних позицій, що зберігаються у базі даних. Цей інтерфейс дозволяє швидко оцінити актуальний асортимент, перевірити коректність внесених найменувань, візуалізувати продукцію та здійснювати подальше адміністрування товарного складу. На рис. 4.7 зображено сторінку перегляду товарів, яка відображає повний перелік доступних позицій у зручному графічному форматі.

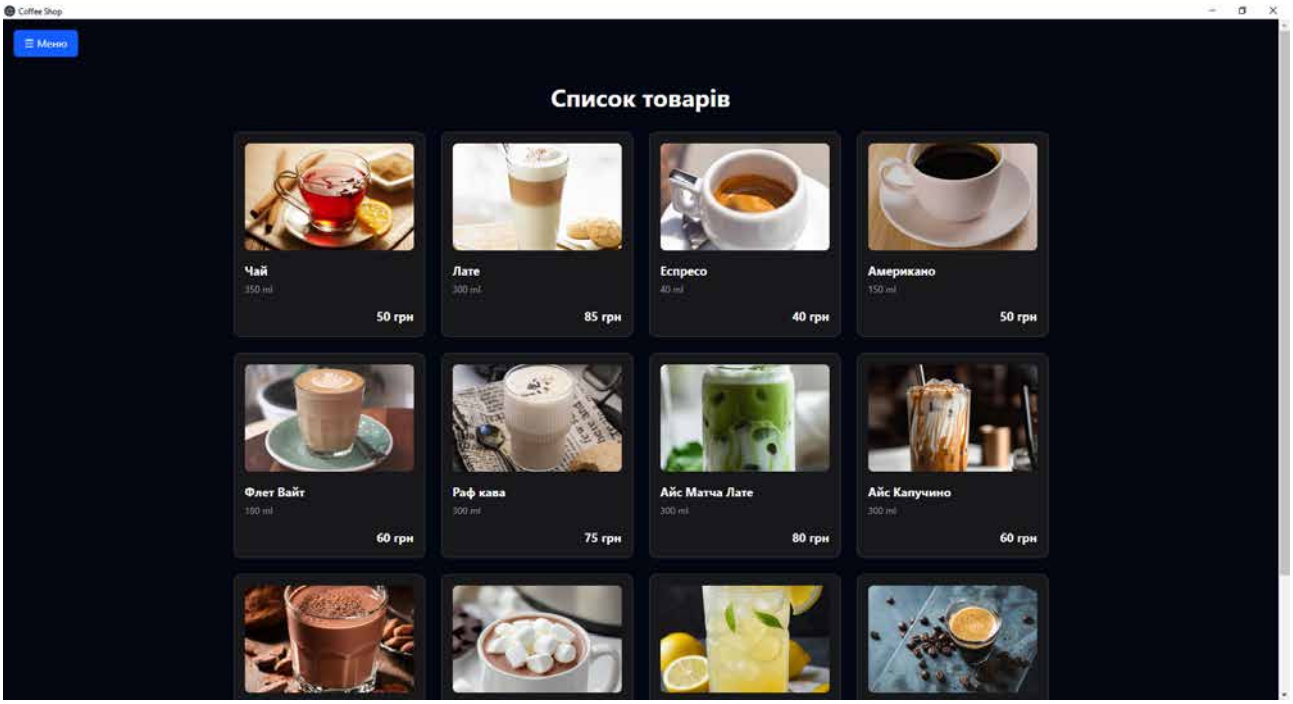

Рис. 4.7 Відображення списку товарів у системі

Кожен товар представлений у вигляді картки з фотографією, назвою, об’ємом (мл) та ціною у гривнях. Інформація згрупована в сітку з адаптивною верстkою, що дозволяє комфортно працювати з інтерфейсом як на широкоформатних моніторах, так і на пристроях з меншою роздільною здатністю. Картки візуально відокремлені одна від одної, що забезпечує чітке сприйняття й запобігає помилковому вибору.

У ході тестування було перевірено відображення різних наборів товарів, включаючи позиції з довгими назвами, великою ціною та відсутністю зображення. Система коректно відображає всі записи, що підтверджує успішну інтеграцію з базою даних та стабільність механізму завантаження інформації. Завдяки поєднанню текстової та графічної інформації інтерфейс стає зручним як для перегляду, так і для візуального контролю цінової політики.

На рис. 4.8 зображено інтерфейс редагування товару, що використовується для оновлення характеристик продукції у системі обліку замовлень кав’ярні.

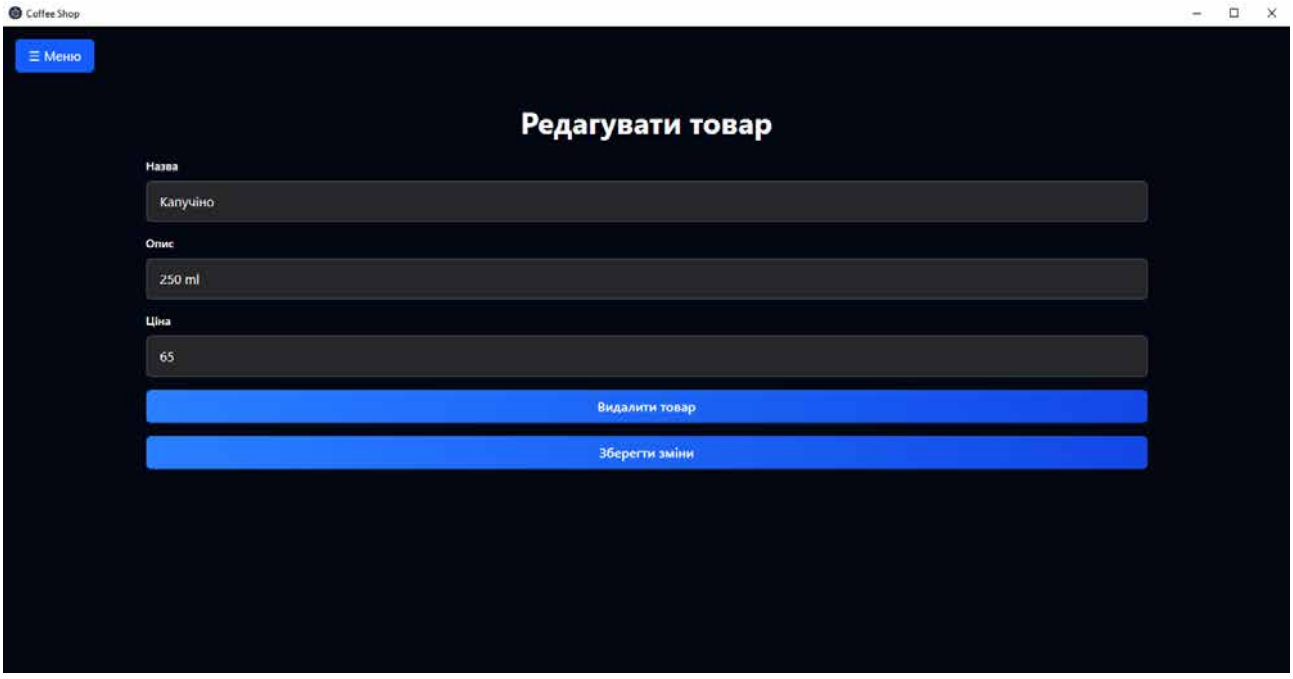

Рис. 4.8 Редагування інформації про товар

Форма включає три основні поля для введення: назва товару, опис та ціна. Кнопка «Зберегти зміни» розміщена в нижній частині форми і виконує

збереження оновлених даних до бази. Кнопка «Видалити товар» прибирає товар з системи.

У процесі тестування перевірялися такі аспекти: автозаповнення полів відповідно до обраного товару, коректність збереження нових значень, робота форми при введенні нечислових значень у поле «Ціна». Особливу увагу було приділено обробці помилок при неправильній структурі вхідних даних.

Результати тестування підтвердили коректність передачі та збереження даних між клієнтською формою та серверною частиною системи. Інтерфейс забезпечує мінімалістичний, але зручний вигляд, що дозволяє швидко редагувати властивості товарів. Таким чином, функціональність редагування товарів реалізована відповідно до вимог, забезпечуючи зручність роботи персоналу.

Останнім елементом адміністративної частини системи є функціонал перегляду транзакцій, що дозволяє контролювати історію проведених операцій, перевіряти факти продажів та здійснювати базовий облік фінансової активності. Цей інструмент особливо важливий у контексті щоденного аналізу діяльності кав'ярні, оскільки надає швидкий доступ до інформації про здійснені замовлення. На рис. 4.9 зображено інтерфейс модуля перегляду транзакцій, який є доступним для адміністратора системи.

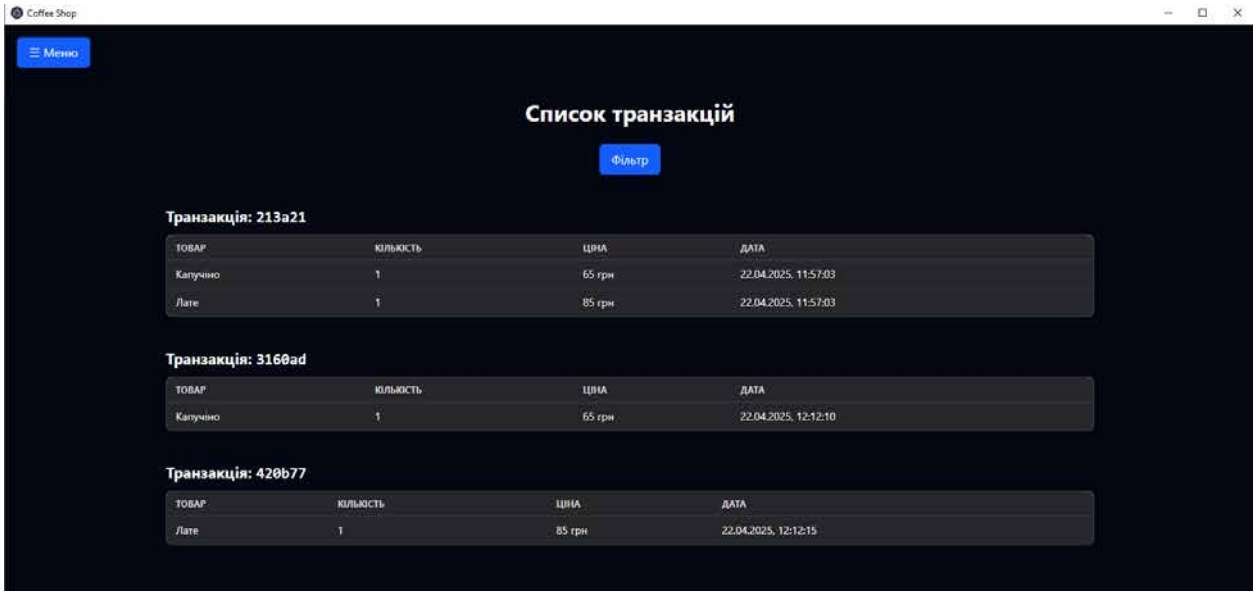

Рис. 4.9 Перегляд транзакцій у системі

Кожна транзакція виводиться у вигляді окремого блоку з унікальним ідентифікатором. Усередині блоку зазначено перелік товарів, що були замовлені, їхню кількість, ціну та точну дату й час оформлення. Інформація відображається у впорядкованій табличній формі, що дозволяє швидко оцінити зміст кожної операції. Для зручності користувача передбачено кнопку фільтрації, яка дає змогу сортування за датою та назвою товару.

Під час тестування перевірено правильність відображення структурованих даних, послідовність сортування та швидкість завантаження при наявності великої кількості записів. Система стабільно обробляє запити та коректно відтворює усі транзакції, що зберігаються у базі. Оформлення та логіка відображення повністю відповідають очікуванням від звітного інтерфейсу, забезпечуючи адміністративному користувачу необхідні засоби для оперативного аналізу комерційної діяльності закладу.

У разі входу до системи під обліковим записом касира користувач отримує доступ до спрощеного меню, адаптованого до виконання операцій, що безпосередньо пов’язані з процесом обслуговування клієнтів. Це дозволяє уникнути перевантаження інтерфейсу непотрібними функціями та зосередитись виключно на оперативній роботі в межах визначених повноважень. На рис. 4.10 зображено меню, яке стає доступним після авторизації касира.

Інтерфейс містить три логічно згруповані розділи. У блоці «Товар» передбачено можливість додавання нового товару та перегляду вже наявного асортименту, що дозволяє касиру вносити зміни у список продукції в межах своєї компетенції. Основний акцент зроблено на розділі «Робота», де розміщено кнопку переходу до модуля «Каса» – основного інструменту для здійснення продажів і обробки замовлень. Завершується меню пунктом «Вийти», що дозволяє завершити сеанс і повернутись до форми авторизації.

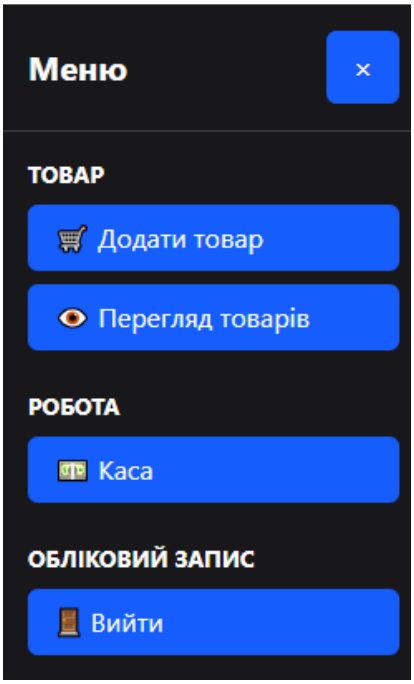

Рис. 4.10 Меню касира після входу в систему

Під час тестування було перевірено правильність відображення доступного функціоналу залежно від ролі користувача. Система коректно обмежує доступ касира до адміністративних функцій, таких як перегляд транзакцій або управління персоналом. Структура меню є лаконічною, інтуїтивно зрозумілою та повністю відповідає потребам роботи касира в умовах кав'ярні.

Головним робочим середовищем касира в межах програмного забезпечення є модуль «Каса», що забезпечує виконання основного бізнес-процесу – оформлення замовлення та проведення продажу. Даний інтерфейс дозволяє у зручному форматі взаємодіяти з клієнтом у режимі реального часу, додаючи товари до замовлення, обираючи тип оплати та формуючи транзакцію. На рис. 4.11 представлено вигляд модуля каси, доступного касиру після входу в систему.

У лівій частині екрану розміщено панель керування, яка містить кілька випадаючих списків: вибір касира, перелік доданих товарів, тип оплати, а також кнопку для видалення обраного товару з поточного замовлення. У правій частині відображається візуальний каталог товарів із зображенням, назвою, об'ємом та ціною – касир має змогу швидко додати потрібні позиції до

замовлення одним кліком. У нижній частині екрану виводиться сума поточного замовлення, яка оновлюється динамічно залежно від доданих товарів. Завершує процес кнопка «Покупка», яка ініціює формування транзакції та її збереження у базі даних.

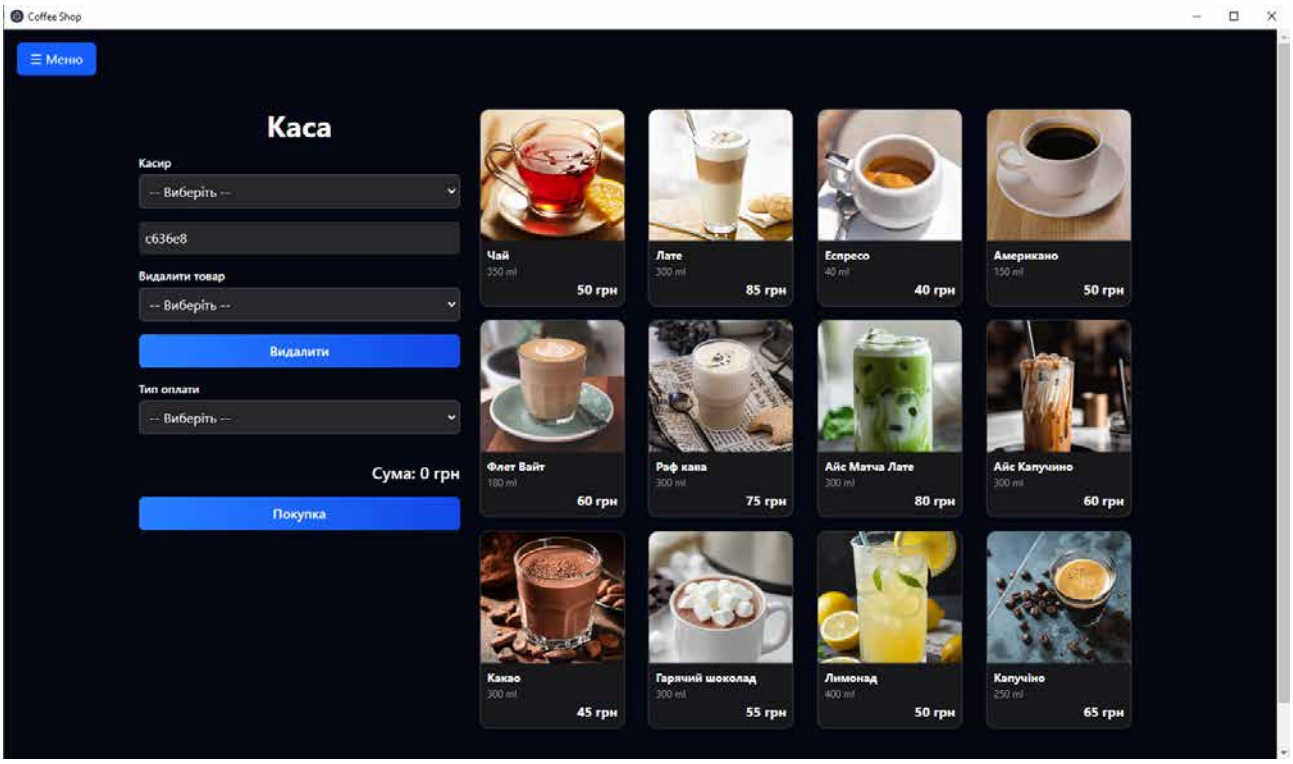

Рис. 4.11 Інтерфейс касира для оформлення замовлення

У процесі тестування було перевірено правильність обробки натискань, додавання товарів, видалення позицій із замовлення, зміни способу оплати, а також завершення покупки. Система стабільно обробляє дії користувача, вчасно оновлює інтерфейс та забезпечує передбачувану поведінку в усіх стандартних сценаріях. Інтерфейс побудований таким чином, щоб максимально зменшити кількість дій касира та пришвидшити обслуговування клієнта. Загалом, модуль каси демонструє високу зручність, інтерактивність і готовність до використання в умовах реального торгового процесу.

4.2 Вимоги до апаратного та програмного забезпечення

Під час розробки системи обліку замовлень у кав’ярні особлива увага приділялася визначенню вимог до апаратного та програмного забезпечення, які

повинні забезпечити її стабільну й безпечну роботу за умов реального комерційного використання. Враховуючи архітектуру застосунку – вимоги до обчислювальних ресурсів є достатньо гнучкими, щоб забезпечити роботу як на локальному середовищі, так і в умовах розгортання на окремому сервері.

Для коректної роботи достатньо клієнтського комп’ютера з процесором Intel Core i3 третього покоління або еквівалентним AMD Ryzen 3, частотою від 2.0 ГГц. Рекомендований обсяг оперативної пам’яті – 4 ГБ мінімум, однак для більш комфортної роботи доцільним є використання 8 ГБ і більше. З урахуванням потреб зберігання логів, резервних копій та обробки запитів рекомендовано мати 10 ГБ вільного простору на диску.

Сервер бази даних потребує встановлення PostgreSQL (версія 13 або новіша), з можливістю створення ролей, таблиць, тригерів і збережених процедур. У випадку розгортання локально вся система може бути встановлена на одному пристрої. Для мережевого використання рекомендується розділення на дві логічні частини: сервер бази даних та застосунок. Це дозволить обслуговувати декілька клієнтських робочих місць, зокрема касирів і адміністратора, одночасно.

З боку програмного забезпечення бажаним є наявність сучасного браузера, такого як Google Chrome або Microsoft Edge. Операційна система може бути як Windows 10/11, так і Linux-дистрибутиви, залежно від середовища розгортання.

Загалом, система не потребує надмірно потужного обладнання, однак її продуктивність і стабільність значною мірою залежать від налаштувань серверної частини, коректності роботи PostgreSQL та якості мережевого з’єднання у випадку розподіленої архітектури. Правильно підібране технічне середовище забезпечує безперервну роботу, швидке реагування інтерфейсу, надійне зберігання даних і комфортну взаємодію з користувачем.

4.3 Склад інсталяційного пакету

Під час підготовки програмного забезпечення системи обліку замовлень у кав’ярні до розгортання було сформовано інсталяційний пакет, що містить усі необхідні компоненти для запуску і повноцінної роботи застосунку. Система побудована за клієнт-серверною архітектурою із використанням технологій NestJS (на базі Node.js), React (для фронтенду) та PostgreSQL (для бази даних), інсталяційний пакет складається з кількох логічних модулів, кожен з яких виконує роль у загальній інфраструктурі.

Основним виконуваним компонентом є файл запуску застосунку, що ініціалізує графічний інтерфейс, забезпечує маршрутизацію запитів, обробку подій користувача та взаємодію з базою даних. У складі інсталяційного пакету присутній конфігураційний файл, який містить налаштування середовища: параметри з’єднання з базою даних, робочий порт, ключі безпеки, а також інші змінні, які можуть бути адаптовані відповідно до середовища розгортання.

Також до пакету входять необхідні зовнішні бібліотеки та залежності, що забезпечують роботу основного функціоналу, включаючи валідацію даних, шифрування, авторизацію користувачів, збереження сесій та інші службові процеси. Ці компоненти інтегруються у систему під час встановлення або вже постачаються у готовому вигляді в окремому каталозі.

Для взаємодії з базою даних у комплекті поставки надається SQL-скрипт, який дозволяє автоматизовано створити необхідну структуру бази – таблиці, зв’язки, обмеження та початкові записи. Це забезпечує коректний запуск системи навіть у середовищі, де база даних ще не розгорнута.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено, змодельовано та реалізовано інформаційну систему обліку замовлень у кав’ярні, що відповідає сучасним вимогам до швидкості, простоти й надійності обслуговування клієнтів. Початково було здійснено аналіз предметної області, що дало змогу окреслити основні бізнес-процеси кав’ярні та виявити ключових учасників взаємодії. Сформульовано як функціональні, так і нефункціональні вимоги до системи, що стали основою для формування технічного завдання.

Для формалізації вимог і структури системи було побудовано комплекс UML-діаграм: діаграму прецедентів, послідовностей, активності, класів та абстракції, що забезпечили цілісне бачення як логіки взаємодії, так і фізичної архітектури програмного забезпечення.

Прикладне програмне забезпечення реалізовано відповідно до сучасних підходів, зокрема використано фреймворк NestJS для побудови серверної частини, React для створення клієнтського інтерфейсу та Prisma ORM для взаємодії з PostgreSQL. Здійснено алгоритмізацію ключових бізнес-процесів – авторизації користувача, додавання касира, формування замовлення – у вигляді блок-схем і реалізовано їх у вигляді програмних модулів. У процесі тестування перевірено коректність роботи всіх елементів інтерфейсу для адміністратора і касира: форми додавання та перегляду товарів, касирів, транзакцій, модуль каси, логіка авторизації та обробки помилок. Система продемонструвала стабільну роботу, чітку реакцію на дії користувача та відповідність заданим вимогам.

Таким чином, було розроблено програмне забезпечення системи обліку замовлень у кав’ярні, що дозволяє ефективно керувати даними, покращити обслуговування клієнтів і зменшити навантаження на персонал. Отримані результати свідчать про можливість практичного впровадження системи у малих кав’ярнях з подальшою можливістю розширення її функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Акачакія К., Ліщинська Л.Б. Аналіз вимог до створення системи точки обслуговування для ресторанів і кафе в Україні. Вінницький національний технічний університет, 2021. 12 с.

2. Гармон, К. Що я знаю про роботу кав’ярень: реалії бізнесу від власника мережі 3fe Coffee / К. Гармон ; пер. з англ. О. Любенко. – Київ : Наш Формат, 2021. 264 с.

3. Марченко, В. М. Основи підприємницької діяльності : підручник / В. М. Марченко [та ін.] ; за ред. В. М. Марченка. – Київ : КПІ ім. Ігоря Сікорського, 2022. 350 с.

4. Poster POS Reviews - Pros & Cons, Ratings & more - GetApp. URL: https://www.getapp.com/retail-consumer-services-software/a/poster-pos/reviews/ (Дата звернення: 08.05.2025).

5. Poster POS. URL: https://www.capterra.co.za/software/141470/poster-pos (Дата звернення: 08.05.2025).

6. Poster POS Pricing, Alternatives & More 2025. URL: https://www.capterra.com/p/141470/Poster-POS/ (Дата звернення: 08.05.2025).

7. Poster POS Reviews & Ratings 2025. URL: https://www.trustradius.com/products/poster-pos/reviews (Дата звернення: 08.05.2025).

8. Poster POS Software Reviews, Demo & Pricing - 2025. URL: https://www.softwareadvice.com/retail/poster-profile/ (Дата звернення: 08.05.2025).

9. Loyverse POS Software Reviews, Pros and Cons. URL: https://www.softwareadvice.com/retail/loyverse-profile/reviews/ (Дата звернення: 08.05.2025).

10. Loyverse POS Software Reviews, Demo & Pricing - 2025. URL: https://www.softwareadvice.com/retail/loyverse-profile/ (Дата звернення: 08.05.2025).

11. Loyverse POS Reviews 2025. Verified Reviews, Pros & Cons. URL: https://www.capterra.com/p/150632/Loyverse-POS/reviews/ (Дата звернення: 08.05.2025).

12. Loyverse POS Reviews - Pros & Cons, Ratings & more - GetApp. URL: https://www.getapp.com/retail-consumer-services-software/a/loyverse-pos/reviews/ (Дата звернення: 08.05.2025).

13. SambaPOS Reviews in 2025. URL: https://sourceforge.net/software/product/SambaPOS/ (Дата звернення: 08.05.2025).

14. SambsPOS V5 Lisansı. URL: https://www.sambaposdestek.com/urun/1/sambapos-v5-lisasi (Дата звернення: 08.05.2025).

15. SambaPOS Pricing, Reviews & Features. URL: https://www.capterra.ca/software/161780/sambapos (Дата звернення: 08.05.2025).

16. Key Features of SambaPOS. URL: https://www.softwaresuggest.com/sambapos (Дата звернення: 08.05.2025).

17. Пушкар М.С. Ідеальна система обліку: концепція, архітектура, інформація: [монографія] / М.С. Пушкар, М.Г. Чумаченко. – Тернопіль: Карт-бланш, 2011. – 336 с.

18. Chen, P. P. Entity-relationship modeling: historical events, future trends, and lessons learned / P.P. Chen // Entity-Relationship Approach to Software Engineering: international conference, November 27–30, 2011, Yokohama, Japan: proceedings. – 2011. – 541 p.

19. PostgreSQL Tutorial. URL: https://neon.tech/postgresql/tutorial (Дата звернення: 08.05.2025).

20. Makris A., Tserpes K., Spiliopoulos G., Zissis D., Anagnostopoulos D. MongoDB Vs PostgreSQL: A comparative study on performance aspects. GeoInformatica. 2021. Vol. 25, pp. 243-268.

21. Exploring Nest.js and TypeScript: A Powerful Combo for Building Scalable Web API. URL: https://medium.com/@nwonahr/exploring-nest-js-and-typescript-a-powerful-combo-for-building-scalable-web-api-7b9919b05975 (Дата звернення: 08.05.2025).

22. Odeniran Q., Wimmer H., Du J. Javascript frameworks–a comparative study between react. js and angular. js. In Interdisciplinary Research in Technology and Management. CRC Press. 2024. pp. 319-327.

ДОДАТОК

Додаток А

Код програми

Сторінок 15

Київ-2025

Реалізація авторизації користувача на фронт частині:

```
import { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { PAGES } from "../types/pages";
import { ROLES } from "../types/roles";

const Login = () => {
  const [form, setForm] = useState({ username: "", password: "" });
  const [error, setError] = useState("");
  const navigate = useNavigate();

  useEffect(() => {
    const isAuth = localStorage.getItem("auth") === "true";
    const role = localStorage.getItem("role");

    if (isAuth) {
      navigate(role === ROLES.ADMIN ? PAGES.VIEW_CASHIER :
PAGES.FUND);
    }
  }, [navigate]);

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleLogin = async () => {
    setError("");
    try {
      const res = await fetch("http://localhost:3000/auth/login", {
```

```
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(form),
   });

   if (!res.ok) {
    const err = await res.json();
    throw new Error(err.message || "Помилка входу");
   }

   const data = await res.json();

   localStorage.setItem("auth", "true");
   localStorage.setItem("role", data.role);
   console.log(data.role);

   if (data.role === "admin") {
    navigate(PAGES.VIEW_CASHIER);
    return;
   }

   navigate(PAGES.FUND);
  } catch (err: unknown) {
   if (err instanceof Error) {
    setError(err.message);
   } else {
    setError("Невідома помилка");
   }
  }
```

```
  };

  return (
    <div className="p-4 max-w-sm mx-auto space-y-3">
      <h2 className="text-xl font-bold text-white text-center">Вхід</h2>

      <input
        name="username"
        placeholder="Логін"
        value={form.username}
        onChange={handleChange}
        className="border border-zinc-600 bg-zinc-800 text-white p-2 rounded w-
full"
      />

      <input
        name="password"
        type="password"
        placeholder="Пароль"
        value={form.password}
        onChange={handleChange}
        className="border border-zinc-600 bg-zinc-800 text-white p-2 rounded w-
full"
      />

      {error && <div className="text-red-500 text-sm">{error}</div>}

      <button
        onClick={handleLogin}
```

```
        className="bg-blue-600 hover:bg-blue-700 text-white p-2 rounded w-full"
      >
        Увійти
      </button>
    </div>
  );
};

export default Login;
```

Реалізація авторизації користувача на бек частині:

```
import { Controller, Post, Body, UnauthorizedException } from '@nestjs/common';

@Controller('auth')
export class AuthController {
  @Post('login')
  login(@Body() body: { username: string; password: string }) {
    const { username, password } = body;

    if (username === 'admin' && password === 'admin123') {
      return { success: true, role: 'admin' };
    }

    if (username === 'cashier' && password === 'cashier123') {
      return { success: true, role: 'cashier' };
    }

    throw new UnauthorizedException('Невірний логін або пароль');
  }
```

```
}
```

Реалізація додавання касира на фронт частині:

```
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import * as yup from "yup";
import { Input } from "../components/ui/Input";
import { Button } from "../components/ui/Button";
import toast from "react-hot-toast";
import { cashierScheme } from "../validation/cashierScheme";

type FormData = yup.InferType<typeof cashierScheme>;

const AddCashier = () => {
  const {
    register,
    handleSubmit,
    reset,
    formState: { errors, isSubmitting },
  } = useForm<FormData>({
    resolver: yupResolver(cashierScheme),
  });

  const onSubmit = async (data: FormData) => {
    try {
      const res = await fetch("http://localhost:3000/cashiers", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(data),
```

```
    });

    if (!res.ok) throw new Error("Server error");

    toast.success("Касира успішно додано!");
    reset();
  } catch {
    toast.error("Помилка при створенні касира");
  }
};

return (
  <div className="container max-w-xl mx-auto space-y-6 py-10">
    <h1 className="text-3xl sm:text-4xl text-center text-white font-bold">
      Додати нового касира
    </h1>
    <form className="space-y-4" onSubmit={handleSubmit(onSubmit)}>
      <Input
        label="Код касира"
        placeholder="226245"
        {...register("code")}
        error={errors.code?.message}
      />
      <Input
        label="Прізвище"
        placeholder="Іванов"
        {...register("lastName")}
        error={errors.lastName?.message}
      />
```

```
<Input
  label="Ім’я"
  placeholder="Іван"
  {...register("firstName")}
  error={errors.firstName?.message}
/>
<Input
  label="По батькові"
  placeholder="Іванович"
  {...register("middleName")}
  error={errors.middleName?.message}
/>
<Input
  label="Адреса"
  placeholder="вул. Героїв УПА 12"
  {...register("address")}
  error={errors.address?.message}
/>
<Input
  label="Телефон"
  placeholder="+380991112233"
  {...register("phone")}
  error={errors.phone?.message}
/>
<Button
  className="mt-6"
  type="submit"
  fullWidth
  disabled={isSubmitting}
```

```
        >
          {isSubmitting ? "Завантаження..." : "Додати"}
        </Button>
      </form>
    </div>
  );
};

export default AddCashier;
```

Реалізація додавання касира на бек частині:

```
import {
  Controller,
  Get,
  Post,
  Body,
  Param,
  Delete,
  Put,
} from '@nestjs/common';
import { CashierService } from './cashier.service';
import { CreateCashierDto } from './create-cashier.dto';

@Controller('cashiers')
export class CashierController {
  constructor(private readonly cashierService: CashierService) {}

  @Get()
  findAll() {
```

```
    return this.cashierService.findAll();
  }

  @Post()
  create(@Body() body: CreateCashierDto) {
    return this.cashierService.create(body);
  }

  @Get(':code')
  findByCode(@Param('code') code: string) {
    return this.cashierService.findByCode(code);
  }

  @Put(':code')
  updateByCode(@Param('code') code: string, @Body() body: CreateCashierDto) {
    return this.cashierService.updateByCode(code, body);
  }

  @Delete(':id')
  delete(@Param('id') id: string) {
    return this.cashierService.delete(+id);
  }
}
```

Реалізація створення замовлення на фронт частині:

```
import { useFund } from "../hooks/useFund";
import { Cart } from "../components/ui/Cart";
import { Button } from "../components/ui/Button";
import { Select } from "../components/ui/Select";
```

```
import { useTransaction } from "../hooks/useTransaction";
import { v4 as uuidv4 } from "uuid";
import toast from "react-hot-toast";

const Fund = () => {
  const {
    cashiers,
    products,
    cart,
    form,
    setForm,
    handleAddProduct,
    handleRemove,
    removeId,
    setRemoveId,
    total,
    setCart,
  } = useFund();
  const { sendTransaction, loading } = useTransaction();

  const handlePurchase = async () => {
    if (!form.cashier) {
      toast.error("Оберіть касира перед покупкою");
      return;
    }

    if (!form.paymentType) {
      toast.error("Оберіть тип оплати");
      return;
```

```
  }

  if (cart.length === 0) {
    toast.error("Кошик порожній");
    return;
  }

  const now = new Date().toISOString();

  const items = cart.map((item) => ({
    transactionCode: form.transactionCode,
    productName: item.name,
    quantity: item.quantity,
    price: item.price,
    date: now,
  }));

  const success = await sendTransaction(items);
  if (success) {
    setCart([]);
    setForm((prev) => ({
      ...prev,
      transactionCode: uuidv4().slice(0, 6),
    }));
  }
};

return (
```

```
    <div className="grid grid-cols-1 lg:grid-cols-3 gap-6 container mx-auto py-10 text-white">
      {/* Left */}
      <div className="space-y-4 lg:col-span-1">
        <h1 className="text-4xl text-center font-bold">Каса</h1>

        <Select
          label="Касир"
          value={form.cashier}
          onChange={(e) => setForm({ ...form, cashier: e.target.value })}
          options={cashiers.map((c) => ({
            value: c.code,
            label: `${c.lastName} ${c.firstName}`,
          }))}
        />

        <input
          value={form.transactionCode}
          readOnly
          className="w-full p-2 rounded bg-zinc-800 text-white"
        />

        {cart.length !== 0 && <Cart items={cart} />}

        <Select
          label="Видалити товар"
          value={removeId}
          onChange={(e) => setRemoveId(e.target.value)}
          options={cart.map((c) => ({ value: c.productId, label: c.name }))}
```

```
        />

        <Button onClick={handleRemove} fullWidth>
          Видалити
        </Button>

        <Select
          label="Тип оплати"
          value={form.paymentType}
          onChange={(e) => setForm({ ...form, paymentType: e.target.value })}
          options={[
            { value: "Готівка", label: "Готівка" },
            { value: "Картка", label: "Картка" },
          ]}
        />

        <div className="text-right text-xl font-semibold pt-4">
          Сума: {total} грн
        </div>

        <Button onClick={handlePurchase} disabled={loading} fullWidth>
          Покупка
        </Button>
      </div>

      {/* Right */}
      <div className="lg:col-span-2 grid grid-cols-[repeat(auto-
fill,minmax(160px,1fr))] gap-4 justify-center">
        {products.map((product) => (
```

```
<div
  key={product.code}
  onClick={() => handleAddProduct(product, 1)}
  className="cursor-pointer group border border-zinc-700 rounded-xl overflow-hidden bg-zinc-900 hover:shadow-lg transition flex flex-col w-40 sm:w-44 h-60"
>
  <div className="h-2/3 overflow-hidden">
    {product.imageUrl ? (
      <img
        src={`http://localhost:3000${product.imageUrl}`}
        alt={product.name}
        className="w-full h-full object-cover group-hover:scale-105 transition"
      />
    ) : (
      <div className="flex items-center justify-center w-full h-full text-zinc-500">
        Немає фото
      </div>
    )}
  </div>

  <div className="p-2 flex-1 flex flex-col justify-between">
    <h2 className="font-semibold text-sm truncate">{product.name}</h2>
    <p className="text-xs text-zinc-400 truncate">
      {product.description || "—"}
    </p>
    <div className="text-right font-bold text-base">
      {product.price} грн
    </div>
```

```
          </div>
        </div>
      ))}
    </div>
  </div>
 );
};

export default Fund;
```

Реалізація створення замовлення на бек частині:

```
import { Controller, Post, Body, Get } from '@nestjs/common';
import { TransactionService } from './transactions.service';
import { CreateTransactionDto } from './create-transaction.dto';

@Controller('transactions')
export class TransactionController {
 constructor(private readonly transactionService: TransactionService) {}

 @Post()
 async create(@Body() transactions: CreateTransactionDto[]) {
  return this.transactionService.createMany(transactions);
 }

 @Get()
 findAll() {
  return this.transactionService.findAll();
 }
}
```