Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікацій та електронних систем
(назва відділення)
Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітній ступінь)

на тему: **Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»**

Виконав: студент VI курсу, групи КІб-602

Спеціальності: **123 Комп'ютерна інженерія**
(шифр і назва спеціальності)

_____ Олег ВАЛЬЧИШИН
(підпис) (ім'я та прізвище)

Керівник _____ Василь ПИЖ
(підпис) (ім'я та прізвище)

Рецензент _____
(підпис) (ім'я та прізвище)

Тернопіль – 2025

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерної інженерії
Освітній ступінь бакалавр
Освітньо-професійна програма: Комп'ютерна інженерія
Спеціальність 123 Комп'ютерна інженерія
Галузь знань: 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"06" травня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Вальчишину Олегу Петровичу
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи: **Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»**

керівник роботи: Пиж Василь Степанович

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя» від 05.05.2025 р. № 4/9-217.

2. Строк подання студентом кваліфікаційної роботи 20 червня 2025 року.

3. Вихідні дані до роботи: мова програмування C++, фреймворк Qt, система керування базами даних PostgreSQL.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Безпека життєдіяльності, основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграма варіантів використання. Схема структурна програмного забезпечення. ER-діаграма бази даних. UML-діаграма класів програмного забезпечення. Блок-схема алгоритму роботи функції. Таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Оксана РЕДЬКВА заст. директора з НВР		
Безпека життєдіяльності, основи охорони праці	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	06.05	
2	Збір і узагальнення інформації по роботі	19.05	
3	Написання першого розділу	23.05	
4	Розробка технічного та робочого проекту	28.05	
5	Написання спеціального розділу	03.06	
6	Розрахунок економічної частини	05.06	
7	Написання розділу охорони праці	07.06	
8	Виконання графічної частини	12.06	
9	Оформлення проекту	16.06	
10	Проходження нормоконтролю	18.06	
11	Попередній захист роботи	20.06	
12	Захист роботи		

7. Дата видачі завдання 06 травня 2025 року

Студент _____ Олег ВАЛЬЧИШИН
(підпис) (ім'я та прізвище)

Керівник роботи _____ Василь ПИЖ
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Метою кваліфікаційної роботи бакалавра є розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго».

В ході розробки застосовані архітектурні принципи, такі як MVC та Repository Pattern, що у поєднанні з принципами SOLID, гарантують високу якість коду, його гнучкість, розширюваність та легкість у подальшій підтримці та модифікації. Програмне забезпечення успішно інтегровано з надійною базою даних PostgreSQL, що забезпечує цілісність та збереження даних, а також підтримує розширене управління статусами замовлень.

Ключові слова: вікно замовлень, офіціант піцерії, піцерія, C++17, Qt6, PostgreSQL, GUI, віджет, діалог.

ANNOTATION

The purpose of this bachelor's qualification paper is the development of the software "Flamingo Pizzeria Waiter Order Window."

During the development process, architectural principles such as MVC (Model-View-Controller) and Repository Pattern were applied. These, combined with SOLID principles, ensure high code quality, flexibility, extensibility, and ease of future maintenance and modification. The software is successfully integrated with a reliable PostgreSQL database, which guarantees data integrity and preservation, and supports an extended order status workflow.

Keywords: order window, pizzeria waiter, pizzeria, C++17, Qt6, PostgreSQL, GUI, widget, dialog.

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ І СКОРОЧЕНЬ

API (Application Programming Interface) – інтерфейс програмування застосунків.

MVC (Model-View-Controller) – Модель-Вид-Контролер.

QoS (Quality of Service) – якість обслуговування.

RAM (Random Access Memory) – оперативна пам'ять.

SDK (Software Development Kit) – комплект для розробки програмного забезпечення.

SOLID – набір із п'яти принципів об'єктно-орієнтованого програмування, спрямованих на створення легкопідтримуваного та розширюваного коду:

SQL (Structured Query Language) – структурована мова запитів.

UAT (User Acceptance Testing) – приймальне тестування користувачем. Фінальний етап тестування, що проводиться кінцевими користувачами.

UI (User Interface) – користувацький інтерфейс.

UML (Unified Modeling Language) – уніфікована мова моделювання.

ВСТУП

У сучасних умовах ринку ресторанного бізнесу ефективність та швидкість обслуговування клієнтів є ключовими факторами успіху. Піцерія «Фламінго», прагнучи оптимізувати свої операційні процеси та підвищити рівень задоволеності відвідувачів, ініціює розробку спеціалізованого програмного забезпечення – «Вікно замовлень офіціанта піцерії Фламінго».

Метою даної розробки є створення інтегрованого інструменту, який автоматизує повний цикл роботи офіціанта з замовленнями: від приймання та формування до відправлення на кухню, відстеження статусу приготування та фінального розрахунку. Проєкт передбачає не лише спрощення повсякденних операцій, а й надання керівництву піцерії потужних аналітичних інструментів для контролю та оптимізації бізнес-процесів.

Дане програмне забезпечення буде розроблене на базі платформи Qt6/C++ з використанням PostgreSQL як системи управління базами даних, що забезпечить високу продуктивність, надійність та гнучкість рішення. Впровадження системи дозволить значно скоротити час обробки замовлень, мінімізувати помилки, пов'язані з людським фактором, та підвищити загальну ефективність роботи персоналу. Це, в свою чергу, сприятиме покращенню якості обслуговування, зростанню лояльності клієнтів та, як наслідок, збільшенню прибутковості піцерії «Фламінго».

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

В контексті розробки програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» доцільно провести аналітичний огляд існуючих на ринку рішень, що автоматизують процеси управління замовленнями в ресторанному бізнесі. Це дозволить ідентифікувати кращі практики, оцінити функціональні можливості конкуруючих продуктів та визначити ключові переваги, які можуть бути інтегровані або вдосконалені в пропонованій системі. Аналіз включатиме як комплексні POS-системи, так і спеціалізовані модулі для офіціантів:

1. POS-системи (Point of Sale Systems)

Більшість сучасних закладів громадського харчування використовують інтегровані POS-системи, які охоплюють широкий спектр функцій: від управління замовленнями та продажами до складського обліку, CRM та фінансової звітності.

Одна з найпоширеніших систем на ринку України це R-Keeper. Пропонує модулі для офіціантів (мобільні термінали, стаціонарні робочі місця), управління столами, широкі можливості налаштування меню та гнучку систему знижок. Переваги: висока функціональність, масштабованість, інтеграція з різними видами обладнання. Недоліки: висока вартість ліцензій та впровадження, складна конфігурація [1].

Poster POS – хмарна POS-система, орієнтована на невеликі та середні заклади. Має інтуїтивно зрозумілий інтерфейс, модуль для офіціантів (працює на планшетах), управління складом, фінансовий та статистичний облік. Переваги: доступна ціна, простота впровадження, регулярні оновлення. Недоліки: менша гнучкість у кастомізації складних бізнес-процесів порівняно з R-Keeper [2].

Окрім повноцінних POS-систем, існують окремі застосунки, призначені виключно для офіціантів, які часто інтегруються з більшими системами або працюють автономно.

Waiterio – мобільний застосунок для прийому замовлень, доступний на смартфонах та планшетах. Дозволяє створювати замовлення, додавати модифікатори, відправляти на кухню та керувати столами. Переваги: простота використання, доступність на мобільних пристроях, низька вартість або безкоштовні версії. Недоліки: обмежений функціонал порівняно з комплексними системами, може вимагати інтеграції з існуючою інфраструктурою [3].

Loyverse POS – Хмарна система, що також пропонує функціонал для офіціантів. Дозволяє керувати замовленнями, столами, чеками, а також має базові функції управління запасами. Переваги: безкоштовна базова версія, підтримка різних типів обладнання. Недоліки: для розширеного функціоналу потрібна платна підписка [4].

Аналіз існуючих рішень дозволяє зробити наступні висновки:

– Ринок активно розвивається у напрямку автоматизації ресторанного бізнесу, що підтверджує актуальність розробки програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго».

– Більшість систем прагнуть забезпечити максимальну швидкість та інтуїтивність роботи офіціантів, зокрема через використання сенсорних інтерфейсів та мобільних пристроїв. Це підтверджує важливість реалізації панелі швидкого створення замовлення та візуальної карти столів у пропонованій системі.

– Успішні рішення часто є частиною ширшої екосистеми (POS-систем), що дозволяє об'єднувати фронт-офісні операції з бек-офісними функціями (склад, фінанси, звіти). Пропоноване ПЗ має бути глибоко інтегроване з існуючою системою управління замовленнями піцерії "Фламінго".

– Наявність розвинених звітних та аналітичних функцій є стандартом для сучасних систем, що підкреслює необхідність реалізації персональної статистики офіціанта та звітів для керівництва.

На основі проведеного огляду, розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» матиме змогу врахувати кращі практики ринку, зосередившись на оптимізації ключових операцій офіціанта, забезпеченні

високої швидкості та надійності, а також інтеграції з поточною інфраструктурою піцерії.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Програмне забезпечення «Вікно замовлень офіціанта піцерії Фламінго» призначене для автоматизації операційної діяльності в закладах ресторанного бізнесу, зокрема в піцерії «Фламінго», які використовують модель обслуговування з офіціантами. Основною областю його застосування є оптимізація процесу приймання, обробки та управління замовленнями, що дозволяє підвищити ефективність роботи персоналу та якість обслуговування клієнтів. Назва програмного забезпечення – «PizzeriaWaiter».

1.2.2 Призначення розробки

Створення спеціалізованого програмного модуля, призначеного для автоматизації процесу прийняття та обробки замовлень офіціантами в піцерії «Фламінго», з подальшою інтеграцією в існуючу систему управління замовленнями.

Основними користувачами системи є офіціанти піцерії «Фламінго», менеджери змін та адміністратори системи.

1.2.3 Вимоги до програмного забезпечення

1.2.3.1 Вимоги до функціональних характеристик

Система надасть офіціантам інструменти для швидкого та точного формування замовлень, їх відправлення на кухню, відстеження статусів приготування та завершення розрахунків. Це значно зменшить час, що

витрачається на ручні операції, мінімізує помилки та покращить взаємодію між залом і кухнею. Інтерактивна карта столів дозволить офіціантам та менеджерам відстежувати зайнятість столів у реальному часі, оперативно керувати їхнім розподілом та оптимізувати потік клієнтів.

Модуль меню забезпечить зручний доступ до всіх позицій, включаючи категорії, розміри та модифікатори. Це спростить процес замовлення та допоможе офіціантам швидко орієнтуватися в асортименті. Завдяки вбудованим механізмам звітності, керівництво піцерії отримає доступ до детальної інформації щодо продажів, продуктивності офіціантів, популярності страв та завантаженості закладу. Це забезпечить основу для прийняття обґрунтованих управлінських рішень та оптимізації бізнес-процесів.

Швидкість обробки замовлень, зменшення кількості помилок та можливість враховувати особливі побажання клієнтів безпосередньо в системі сприятимуть підвищенню рівня задоволеності відвідувачів.

1.2.3.2 Вимоги до надійності

Система повинна демонструвати високий рівень надійності, забезпечуючи доступність на рівні 99.5% робочого часу. Функція автоматичного збереження даних кожні 30 секунд та резервне копіювання кожні 4 години є обов'язковими для мінімізації втрат даних.

Вимоги до надійності програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» є критично важливими для забезпечення безперебійної роботи закладу та мінімізації потенційних збитків від простоїв системи. Ці вимоги охоплюють аспекти доступності системи, механізми збереження даних та резервного копіювання.

Система повинна забезпечувати доступність на рівні 99.5% робочого часу. Це означає, що сумарний час, протягом якого система буде недоступною для використання (наприклад, через технічні збої, планове обслуговування, оновлення тощо), не повинен перевищувати 0.5% від загального робочого часу за певний період. Такий показник гарантує майже безперервну роботу офіціантів, що є ключовим для динамічного середовища піцерії.

Для запобігання втраті даних внаслідок непередбачених обставин, таких як збої живлення або програмні помилки, необхідно реалізувати функціонал автоматичного збереження даних кожні 30 секунд. Цей механізм забезпечить мінімальні втрати інформації про замовлення та операції, навіть у разі раптового припинення роботи системи. Збереження даних має бути неперервним та прозорим для користувача.

Крім того, для забезпечення високого рівня надійності та відновлюваності даних, необхідно впровадити систему резервного копіювання кожні 4 години. Це дозволить створити актуальні копії бази даних, які можуть бути використані для повного відновлення системи у випадку серйозних збоїв, пошкодження даних або інших катастрофічних подій. Механізм резервного копіювання має передбачати зберігання копій на віддалених або захищених носіях для гарантії їх збереження.

Реалізація цих вимог до надійності забезпечить стабільну та безпечну роботу програмного забезпечення, що є фундаментальним для ефективного функціонування піцерії.

1.2.3.3 Вимоги до складу і параметрів технічних засобів

Ефективна робота програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» залежить від відповідності технічних засобів визначеним параметрам. Цей розділ окреслює мінімальні та рекомендовані вимоги до апаратного та програмного забезпечення для забезпечення оптимальної продуктивності та надійності системи.

Враховуючи, що система використовуватиме базу даних PostgreSQL, та підтримуватиме одночасну роботу до 20 офіціантів, серверна інфраструктура повинна забезпечувати високу продуктивність та масштабованість.

Мінімальні вимоги до сервера бази даних:

– Процесор: Багатоядерний процесор з тактовою частотою не менше 2.0 ГГц (наприклад, Intel Xeon E3 або аналог AMD Opteron). Рекомендується використання процесорів з архітектурою, оптимізованою для роботи з базами даних.

– Оперативна пам'ять (RAM): Не менше 8 ГБ. Цей обсяг дозволить забезпечити ефективну роботу СУБД та кешування даних для швидкого доступу.

– Дискова підсистема: SSD-накопичувач об'ємом не менше 250 ГБ для операційної системи та бази даних. Використання SSD є критично важливим для забезпечення високої швидкості операцій введення/виведення, що безпосередньо впливає на швидкість завантаження списків замовлень та час відгуку системи. Рекомендується наявність надлишковості дискової підсистеми (RAID 1 або RAID 5) для підвищення надійності.

– Мережевий інтерфейс: Gigabit Ethernet (1 Гбіт/с) для забезпечення швидкого обміну даними між клієнтськими робочими місцями та сервером.

– Операційна система: Linux-дистрибутив (наприклад, Ubuntu Server LTS, Debian) або Windows Server з підтримкою PostgreSQL.

– Рекомендовані вимоги до сервера бази даних:

– Процесор: Процесор рівня Intel Xeon E5 або E7 (або аналог AMD EPYC) з 4-8 ядрами та тактовою частотою від 2.5 ГГц.

– Оперативна пам'ять (RAM): Від 16 ГБ до 32 ГБ, залежно від очікуваного обсягу даних та інтенсивності запитів.

– Дискова підсистема: RAID 10 масив з декількох SSD-накопичувачів загальним об'ємом від 500 ГБ для оптимальної продуктивності та відмовостійкості.

– Мережевий інтерфейс: Два порти Gigabit Ethernet для резервування або агрегації каналів.

– Операційна система: Ubuntu Server LTS або CentOS, з оптимізованими налаштуваннями ядра для роботи з базами даних.

Робочі місця офіціантів повинні забезпечувати плавний та швидкий відгук інтерфейсу користувача.

Мінімальні вимоги до клієнтської станції:

– Процесор: двоядерний процесор з тактовою частотою не менше 1.8 ГГц (наприклад, Intel Core i3 або аналог AMD Ryzen 3).

– Оперативна пам'ять (RAM): Не менше 4 ГБ.

– Дискова підсистема: SSD-накопичувач об'ємом від 120 ГБ для операційної системи та встановленого програмного забезпечення.

– Дисплей: сенсорний монітор або звичайний дисплей з діагоналлю від 15 дюймів та роздільною здатністю не менше 1366x768 пікселів. Підтримка сенсорного введення є перевагою для швидкої роботи офіціантів.

– Мережевий інтерфейс: Ethernet 100 Мбіт/с або Wi-Fi (802.11n/ac) для підключення до локальної мережі.

– Операційна система: Windows 10/11 або сучасний Linux-дистрибутив (наприклад, Ubuntu Desktop) з підтримкою Qt6.

1.2.4 Вимоги до програмної документації

Програмна документація є невід'ємною частиною проєкту розробки програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго». Вона забезпечує повне та чітке розуміння архітектури, функціональності, принципів роботи та процедур розгортання і підтримки системи. Наявність якісної документації є критичною для ефективного супроводу проєкту, навчання нових користувачів та подальшого розвитку програмного продукту [1].

Згідно індивідуального завдання необхідно розробити наступну програмну документацію:

– інструкція з інсталяції програмного забезпечення;

– інструкція з використання тестових наборів;

– інструкція з експлуатації програмного забезпечення.

1.2.5 Техніко-економічні показники

Прогнозований термін розробки та впровадження програмного забезпечення становить 120-150 годин розробки. Цей термін включає всі етапи життєвого циклу розробки програмного забезпечення: аналіз вимог,

проєктування, кодування, тестування, документування та впровадження.

Орієнтовний бюджет розробки програмного забезпечення становить 80 000 – 100 000 грн. Ця сума охоплює витрати на оплату праці команди розробників, ліцензії на необхідне програмне забезпечення (за наявності), витрати на тестування та управління проєктом. Точний бюджет буде визначено після деталізації всіх функціональних та нефункціональних вимог на етапі оцінки проєкту в економічному розділі кваліфікаційної роботи.

1.2.6 Стадії та етапи розробки

Процес розробки програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» буде здійснюватися відповідно до класичного водоспадного циклу (Waterfall Model) або, з огляду на можливість ітерацій, з елементами інкрементної моделі. Це забезпечить послідовне виконання робіт, контроль на кожному етапі та фіксацію вимог перед початком кодування:

1. Стадія аналізу та планування.

Ця стадія є фундаментальною для всього проєкту, оскільки на ній відбувається збір, аналіз та документування всіх вимог до системи, а також формування детального плану розробки.

Етап 1.1. Збір та аналіз вимог.

На цьому етапі проводиться ретельний аналіз потреб замовника шляхом інтерв'ю з представниками піцерії (офіціантами, менеджерами, адміністрацією) та вивченням існуючих бізнес-процесів. Збираються функціональні вимоги до основного вікна замовлень, детального управління замовленнями, роботи зі столами, управління меню, а також до звітності та аналітики. Визначаються нефункціональні вимоги: продуктивність, надійність, безпека та зручність використання. Результатом цього етапу буде формування Технічного завдання (ТЗ).

Етап 1.2. Визначення техніко-економічних показників. На основі зібраних вимог та попереднього аналізу ринку, формуються техніко-економічні показники

проєкту. До них належать прогнозована економічна ефективність, орієнтовні терміни та бюджет розробки, а також прогнозовані експлуатаційні витрати та термін окупності інвестицій.

Етап 1.3. Планування проєкту. Розробляється детальний план проєкту, що включає розподіл ресурсів, визначення ключових етапів та їх термінів, призначення відповідальних осіб, а також розробку стратегії управління ризиками.

2. Стадія проєктування.

На стадії проєктування відбувається розробка архітектурних рішень та детальна специфікація всіх компонентів системи.

Етап 2.1. Архітектурне проєктування. Розробляється загальна архітектура системи, що базується на патерні Model-View-Controller (MVC). Визначаються взаємодія між основними компонентами, принципи роботи з базою даних (PostgreSQL) та архітектура клієнтської частини (Qt6/C++). Результатом є Архітектурний опис системи.

Етап 2.2. Проєктування бази даних. Створюється логічна та фізична модель бази даних, включаючи структуру таблиць, зв'язки між ними, індекси та обмеження цілісності. Забезпечується підтримка розширеного workflow статусів замовлень та зберігання детальної інформації про них.

Етап 2.3. Проєктування інтерфейсу користувача (UI/UX). Розробляються макети та прототипи інтерфейсу основного вікна замовлень офіціанта, картки замовлення, візуальної карти столів та інших ключових елементів. Забезпечується інтуїтивність та зручність використання з урахуванням роботи на сенсорних екранах.

Етап 2.4. Деталізація вимог до технічних засобів. На цьому етапі уточнюються та деталізуються вимоги до серверної інфраструктури, робочих місць офіціантів, а також до додаткового периферійного обладнання (принтери чеків, мережеве обладнання).

3. Стадія реалізації (кодування)

На цій стадії відбувається безпосередня розробка програмного коду

відповідно до затверджених проєктних рішень.

Етап 3.1. Розробка основних модулів. Здійснюється кодування ключових функціональних блоків, таких як модуль авторизації, управління користувачами, управління столами (TableMapWidget), модуль меню (MenuWidget).

Етап 3.2. Розробка модуля замовлень. Реалізується функціонал створення, редагування, відстеження статусів замовлень, управління кошиком та розрахунком суми.

Етап 3.3. Інтеграція з базою даних та сервісами. Здійснюється інтеграція програмних модулів з базою даних PostgreSQL.

Етап 3.4. Розробка звітних та аналітичних функцій. Реалізуються модулі для формування персональної статистики офіціанта та звітів для керівництва.

4. Стадія тестування

На стадії тестування проводиться всебічна перевірка програмного забезпечення на відповідність вимогам ТЗ, виявлення та усунення дефектів:

Етап 4.1. Модульне тестування. Кожен окремий програмний модуль тестується ізольовано для перевірки його коректної роботи.

Етап 4.2. Інтеграційне тестування. Перевіряється коректність взаємодії між різними модулями системи.

Етап 4.3. Системне тестування. Система тестується як єдине ціле, перевіряється її відповідність всім функціональним та нефункціональним вимогам (продуктивність, надійність, безпека, зручність використання).

Етап 4.4. Приймальне тестування. Замовник проводить тестування системи для підтвердження її відповідності власним очікуванням та вимогам, викладеним у ТЗ.

5. Стадія впровадження та супроводу.

Ця стадія охоплює розгортання програмного забезпечення в робочому середовищі замовника та подальшу підтримку системи.

Етап 5.1. Розгортання (деплоймент). Встановлення програмного забезпечення на серверне та клієнтське обладнання замовника, налаштування мережевих з'єднань та бази даних.

Етап 5.2. Навчання користувачів. Проведення тренінгів для офіціантів, менеджерів та адміністраторів з метою ознайомлення їх з функціоналом системи та навчання ефективній роботі.

Етап 5.3. Документування. Фіналізація та надання всієї необхідної програмної документації: посібників користувача та адміністратора, архітектурних описів та тестової документації.

Етап 5.4. Гарантійне та післягарантійне обслуговування. Надання технічної підтримки, усунення виявлених помилок, а також реалізація подальших оновлень та вдосконалень системи за запитом замовника.

1.2.7 Порядок контролю та прийому

Цей розділ Технічного завдання визначає процедури та критерії, за якими буде здійснюватися контроль якості розробки та приймання програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго». Метою є забезпечення відповідності розробленої системи всім вимогам, зазначеним у даному ТЗ, та її готовності до експлуатації.

Контроль якості буде здійснюватися на всіх стадіях життєвого циклу розробки програмного забезпечення, включаючи:

– Контроль на етапі аналізу та проєктування.

Перевірка повноти та коректності зібраних вимог, архітектурних рішень, проєктів бази даних та інтерфейсів користувача. Затвердження документації (ТЗ, архітектурний опис) замовником.

– Контроль на етапі реалізації (кодування).

Проведення регулярних оглядів коду (code reviews) для забезпечення його якості, відповідності стандартам кодування, ефективності та безпеки. Використання систем контролю версій для відстеження змін.

– Контроль на етапі тестування.

Виконання всіх типів тестування, визначених у відповідному розділі ТЗ (модульне, інтеграційне, системне, приймальне). Фіксація та усунення виявлених

дефектів.

2. Процедура приймання програмного забезпечення

Приймання програмного забезпечення буде здійснюватися поетапно, згідно з наступною процедурою:

– Попереднє тестування розробником.

Команда розробки проводить повний цикл внутрішнього тестування, підтверджуючи відповідність системи функціональним та нефункціональним вимогам. Результати тестування документуються.

– Демонстрація функціоналу.

Розробник проводить демонстрацію ключового функціоналу програмного забезпечення представникам замовника. Під час демонстрації акцент робиться на відповідності реалізованих можливостей вимогам ТЗ.

– Приймальне тестування (User Acceptance Testing – UAT).

Замовник самостійно або із залученням своїх представників проводить приймальне тестування системи в умовах, максимально наближених до реальної експлуатації. В ході UAT перевіряється відповідність програмного забезпечення бізнес-процесам піцерії та зручність його використання кінцевими користувачами. Замовник надає перелік виявлених зауважень та дефектів.

– Усунення зауважень та повторне тестування.

Розробник усуває виявлені зауваження та дефекти, після чого проводиться повторне тестування виправлених компонентів.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЕКТУ

2.1 Розробка загальної структури і варіантів використання програми

Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» передбачає створення інтуїтивно зрозумілої та функціональної системи, що ефективно підтримує основні бізнес-процеси піцерії. Загальна структура програмного забезпечення буде базуватися на модульному підході, що забезпечить гнучкість, масштабованість та легкість підтримки. Варіанти використання деталізують взаємодію користувачів із системою, охоплюючи ключові сценарії її експлуатації. Варіанти використання описують типові сценарії взаємодії користувачів із системою. Вони охоплюють основні функції, які повинна виконувати система для задоволення потреб цільової аудиторії:

1) Авторизація користувача.

Дійові особи: Офіціант, Менеджер, Адміністратор.

Користувач запускає програму та вводить свої облікові дані (логін, пароль). Система перевіряє дані, аутентифікує користувача та надає доступ до відповідних функціональних можливостей згідно з його роллю.

2) Створення нового замовлення.

Дійові особи: Офіціант.

Офіціант обирає вільний стіл на візуальній карті. Система відкриває нове вікно замовлення, де офіціант може додавати позиції з меню, вказувати кількість, розміри піц та модифікатори.

3)Додавання позицій до замовлення:

Дійові особи: Офіціант.

Офіціант переглядає категорії меню. Після вибору категорії відображаються доступні страви. Офіціант обирає страву, вказує кількість, розмір (якщо це піца) та можливі модифікатори, після чого позиція додається до кошика замовлення.

4)Відправлення замовлення на кухню.

Дійові особи: Офіціант.

Після формування замовлення офіціант підтверджує його. Система відправляє замовлення на кухню для приготування, змінюючи статус замовлення на "Подано". Одночасно може друкуватися квиток для кухні.

5) Зміна статусу замовлення:

Дійові особи: Офіціант, Менеджер.

Офіціант відстежує стан замовлень у панелі активних замовлень. При отриманні інформації від кухні (наприклад, "Готово") або після подачі страв клієнту, офіціант може змінити статус замовлення на "Готово" або "Подано до столу" відповідно. Система фіксує ці зміни в історії статусів.

6) Розрахунок замовлення та приймання платежу:

Дійові особи: Офіціант.

Після завершення обслуговування офіціант ініціює розрахунок замовлення. Система відображає загальну суму, дозволяє застосувати знижки, додати чайові та обрати метод оплати (готівка, картка, змішана). Після підтвердження платежу, статус замовлення змінюється на "Оплачено", а дані про платіж фіксуються.

7) Перегляд статистики за зміну.

Дійові особи: Офіціант.

Офіціант може переглянути свою персональну статистику за поточну зміну, включаючи кількість обслужених столів, загальну суму продажів та середній час обробки замовлення.

8) Перегляд звітів для керівництва.

Дійові особи: Менеджер, Адміністратор.

Опис: Менеджер або адміністратор може генерувати та переглядати звіти про продажі за період, популярні позиції меню, завантаженість столів та ефективність роботи офіціантів.

Ця структура та визначені варіанти використання формують основу для подальшої детальної розробки та реалізації програмного забезпечення, забезпечуючи відповідність системи потребам піцерії «Фламінго». Для графічного представлення розроблено UML-діаграму варіантів використання, представлену на плакаті 2025.КРБ.123.602.04.00.00 ДВ.

2.2 Розробка системи класів

Програмний комплекс «PizzeriaWaiter» організований відповідно до чіткої архітектурної структури, що розділяє класи за їх функціональним призначенням. Цей підхід сприяє підвищенню читабельності коду, спрощенню підтримки та масштабування системи. Класи згруповані за наступними категоріями:

– Models - ця категорія містить класи, що представляють структури даних та бізнес-сутності системи. Вони відповідають за зберігання та маніпуляцію даними, що відображають реальні об'єкти піцерії, такі як замовлення, столи, позиції меню, користувачі тощо.

– Services - у цій категорії зосереджені класи, які інкапсулюють основну бізнес-логіку застосунку. Вони координують взаємодію між моделями даних та компонентами інтерфейсу користувача, забезпечуючи виконання операцій, пов'язаних з обробкою замовлень, управлінням користувачами, друком та іншими ключовими функціями.

– Repositories – класи цієї категорії відповідають за взаємодію з джерелами даних, такими як база даних PostgreSQL/SQLite. Вони надають методи для збереження, вибірки, оновлення та видалення даних, абстрагуючи бізнес-логіку від деталей роботи з конкретною базою даних.

– UI Widgets – ця категорія включає класи, що відповідають за візуальне представлення даних та взаємодію з користувачем. До них належать вікна, форми, кнопки та інші елементи графічного інтерфейсу, розроблені за допомогою Qt6, зокрема TableMapWidget, MenuWidget та OrderListWidget.

– Database – класи цієї категорії призначені для управління конфігурацією та підключенням до бази даних. Вони можуть включати функціонал для міграції схеми даних, ініціалізації бази даних та управління з'єднаннями.

– Utils – у цій категорії містяться допоміжні класи та функції, які не належать до жодної з вищезазначених категорій, але надають загальні сервіси або утиліти, що використовуються в різних частинах застосунку. Це можуть бути

класи для форматування даних, обробки винятків або інших загальних операцій.

Така структуризація забезпечує модульність, підвищує ремонтопридатність та полегшує подальший розвиток програмного забезпечення «PizzeriaWaiter».

Далі розглянемо детальніше кожну категорію програмного забезпечення. Визначені наступну моделі даних (Models):

– User – представляє користувача системи з ролевою моделлю доступу. Підтримує три ролі: офіціант, менеджер та адміністратор. Кожна роль має різні рівні доступу до функціональності системи. Клас відстежує час останнього входу, статус активності користувача та надає методи для перевірки дозволів на виконання різних операцій.

– Order – основна модель замовлення з повним життєвим циклом. Підтримує п'ять основних статусів: чернетка, подано, готово, подано до столу та оплачено. Автоматично розраховує загальну суму замовлення на основі доданих позицій. Забезпечує функціональність додавання та видалення позицій, підрахунок кількості товарів та генерацію текстових описів статусів.

– OrderItem – представляє окрему позицію в замовленні. Зберігає інформацію про кількість, ціну за одиницю та автоматично розраховує загальну вартість позиції. Підтримує спеціальні інструкції для кухні та прив'язку до розмірів піц. Інтегрується з MenuItem для отримання базової інформації про товар.

– MenuItem – позиція меню з підтримкою категоризації на піцу, напої, закуски та десерти. Містить базову ціну, опис, час приготування та статус доступності. Має спеціальну обробку для піц з можливістю вибору розміру. Підтримує зображення товарів та індикатори популярності.

– Table – модель столу ресторану з візуальним позиціонуванням на карті залу. Підтримує три статуси: вільний, зайнятий та зарезервований. Зберігає інформацію про ємність столу та координати для відображення на карті залу. Забезпечує кольорове кодування статусів для зручності офіціантів.

Категорія сервісів (Services) містить такі сервіси:

– AuthService – управляє всією системою автентифікації та авторизації. Забезпечує безпечний вхід користувачів з хешуванням паролів за алгоритмом SHA-256 з додаванням солі. Підтримує сесійне управління з відстеженням часу входу та автоматичним виходом при втраті з'єднання. Надає методи для зміни паролів, перевірки дозволів на основі ролей та управління користувачами.

– OrderService – містить всю бізнес-логіку управління замовленнями. Автоматично прив'язує створені замовлення до поточного офіціанта через інтеграцію з AuthService. Забезпечує валідацію операцій, розрахунок цін з урахуванням розмірів піц та модифікаторів. Надає методи для зміни статусів замовлень з відповідним логуванням всіх операцій.

– UserActionLogger – спеціалізований сервіс для аудиту дій користувачів. Автоматично логує всі критичні операції: вхід/вихід з системи, створення та модифікацію замовлень, управління користувачами. Зберігає детальну інформацію про кожну дію включно з IP-адресами, часовими мітками та контекстом операції. Надає інструменти для аналізу активності та виявлення порушень безпеки.

– PrintService – керує всіма аспектами друку документів. Підтримує три типи документів: чеки для клієнтів, замовлення для кухні та повні звіти. Генерує HTML-шаблони документів з автоматичним форматуванням та стилізацією. Інтегрується з Qt Print Framework для підтримки різних принтерів та налаштувань друку.

Категорія репозиторіїв (Repositories) містить:

– BaseRepository – базовий клас для всіх репозиторіїв, що забезпечує єдиний інтерфейс роботи з базою даних. Надає методи для створення запитів, обробки помилок та логування операцій БД. Забезпечує консистентність у роботі з параметрами запитів та автоматичне управління транзакціями.

– UserRepository - спеціалізується на роботі з користувачами в базі даних. Надає методи пошуку користувачів за різними критеріями, управління паролями та відстеження активності. Забезпечує безпечне зберігання хешованих паролів та

валідацію автентифікаційних даних.

– OrderRepository - управляє замовленнями в базі даних з підтримкою каскадного збереження позицій. Забезпечує оптимізовані запити для отримання активних замовлень та фільтрації за різними критеріями. Автоматично прив'язує замовлення до офіціантів та відстежує зміни статусів.

– MenuRepository – працює з меню ресторану включно з підтримкою категорій та розмірів піц. Забезпечує швидкий доступ до популярних позицій та спеціальних пропозицій. Підтримує динамічне ціноутворення на основі розмірів та модифікаторів товарів.

– TableRepository – керує інформацією про столи ресторану з підтримкою візуального позиціонування. Забезпечує швидке оновлення статусів столів та синхронізацію з активними замовленнями. Надає оптимізовані запити для відображення карти залу.

Категорія UI Widgets містить наступні інтерфейси користувача:

– MainWindow – головне вікно програми що інтегрує всі модулі системи. Забезпечує систему авторизації з автоматичним блокуванням UI для неавторизованих користувачів. Підтримує ролевий доступ до різних функцій через динамічне формування меню. Координує роботу всіх підсистем та забезпечує централізоване управління сесіями користувачів.

– TableMapWidget – візуальна карта столів ресторану з інтерактивними елементами. Відображає столи у вигляді кнопок з кольоровим кодуванням статусів: зелений для вільних, червоний для зайнятих, жовтий для зарезервованих. Підтримує швидкий вибір столу для створення нового замовлення та автоматичне оновлення статусів в реальному часі.

– MenuWidget – організований у вкладки віджет відображення меню з категоріями. Кожна категорія представлена окремою вкладкою з прокручуваним списком позицій. Для піц підтримується вибір розміру через випадаючий список. Забезпечує швидке додавання позицій до поточного замовлення з автоматичним розрахунком цін.

– OrderListWidget – відображає список активних замовлень у вигляді карток з ключовою інформацією. Кожна картка містить номер замовлення, номер столу, статус, загальну суму та елементи управління. Підтримує швидку зміну статусів через випадаючі списки та кольорове кодування для візуального розрізнення статусів.

Категорія діialогів містить:

– LoginDialog – модальний діалог входу в систему з валідацією введених даних. Підтримує збереження логіна користувача для зручності. Інтегрується з AuthService для перевірки облікових даних та надає зворотний зв'язок при помилках автентифікації.

– UserManagementDialog – комплексний діалог для управління користувачами системи, доступний тільки адміністраторам. Надає можливості створення нових користувачів, редагування існуючих, зміни ролей та скидання паролів. Відображує детальну інформацію про кожного користувача включно з останнім часом входу та статистикою активності.

– UserActivityDialog – спеціалізований діалог для перегляду журналу дій користувачів з розширеними можливостями фільтрації. Підтримує пошук за періодом, типом дії та конкретним користувачем. Надає можливість експорту даних у CSV формат для подальшого аналізу.

Категорія Database містить наступні клас управління базою даних – DatabaseManager. Це Singleton-клас для централізованого управління підключенням до бази даних. Підтримує як PostgreSQL, так і SQLite з автоматичним визначенням типу СКБД. Забезпечує автоматичне створення таблиць при першому запуску та виконання міграцій для оновлення схеми. Надає централізоване логування помилок БД та моніторинг стану підключення.

2.3 Розробка методів

Програмне забезпечення «Вікно замовлень офіціанта піцерії Фламінго» розроблене з урахуванням сучасних принципів інженерії програмного

забезпечення, що забезпечує його гнучкість, розширюваність, ремонтопридатність та стабільність. Основними архітектурними підходами та принципами проектування є:

Архітектурні принципи

Система базується на архітектурному патерні Model-View-Controller (MVC), що передбачає чітке розділення компонентів на три взаємодіючі шари. Шари Model включають класи, такі як User, Order, OrderItem, MenuItem та Table, які містять бізнес-логіку та правила валідації, будучи незалежними від користувацького інтерфейсу. Шари View представлені візуальними компонентами, зокрема MainWindow, TableMapWidget, MenuWidget та OrderListWidget, які відповідають виключно за відображення даних та реагують на зміни в моделях через сигнали Qt. Контролери (шар Controller) реалізовані у вигляді сервісів, таких як OrderService, AuthService та PrintService, які обробляють взаємодію між представленням та моделями, інкапсулюючи бізнес-логіку операцій.

Додатково застосований Repository Pattern для абстракції доступу до даних. Це реалізовано через базовий клас BaseRepository, який надає загальну функціональність управління з'єднанням з базою даних, обробки помилок та виконання загальних запитів. Спеціалізовані репозиторії, як-от UserRepository, OrderRepository, MenuRepository та TableRepository, успадковують цю функціональність та надають специфічні методи для роботи з відповідними сутностями. Перевагами такого підходу є інкапсуляція логіки доступу до даних, можливість тестування компонентів без прив'язки до реальної бази даних, а також легкість заміни типу СУБД (наприклад, перехід між PostgreSQL та SQLite).

Принцип Dependency Injection активно використовується для впровадження залежностей через конструктори класів. Наприклад, MainWindow отримує необхідні сервіси, такі як AuthService для авторизації, OrderService для роботи з замовленнями та UserActionLogger для аудиту, безпосередньо через свій конструктор. Цей підхід сприяє слабкій зв'язаності компонентів, спрощує тестування системи за допомогою mock-об'єктів та забезпечує гнучкість у заміні

реалізацій.

Застосування цих принципів створює міцну основу для розробки високоякісного програмного забезпечення, що є стійким до змін, легко піддається тестуванню та має потенціал для подальшого масштабування та розвитку.

2.4 Розробка структури бази даних

База даних програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» спроєктована для ефективного зберігання та управління даними, необхідними для автоматизації процесів у піцерії. Її структура включає десять основних таблиць, які взаємопов'язані для забезпечення цілісності та функціональності системи:

1) Користувачі системи (users).

Ця таблиця зберігає інформацію про всіх користувачів, що мають доступ до системи. Кожен запис містить унікальний ідентифікатор (id), унікальне ім'я користувача (username), хеш пароля (password_hash), повне ім'я (full_name), роль користувача (role — за замовчуванням 'waiter'), статус активності (is_active), а також часові мітки створення, останнього оновлення та останнього входу в систему.

2) Столи ресторану (restaurant_tables).

Таблиця призначена для обліку всіх столів у закладі. Вона включає унікальний ідентифікатор (id), унікальний номер столу (table_number), його місткість (capacity), поточний статус (status — за замовчуванням 'available'), координати розташування (x_position, y_position) та час створення запису.

3) Категорії меню (menu_categories).

Ця таблиця містить перелік категорій страв та напоїв, представлених у меню. Кожен запис має унікальний ідентифікатор (id), назву категорії (name), порядок відображення (display_order) та статус активності (is_active).

4) Позиції меню (menu_items).

Основна таблиця для зберігання інформації про всі окремі позиції меню.

Вона пов'язана з menu_categories через category_id (що дозволяє автоматично видаляти позиції меню при видаленні категорії). Включає назву позиції (name), опис (description), базову ціну (price), шлях до зображення (image_path), статус доступності (is_available), орієнтовний час приготування (preparation_time) та час створення запису.

5) Розміри піц (pizza_sizes).

Ця таблиця визначає різні доступні розміри піц. Вона містить унікальний ідентифікатор (id), назву розміру (name), діаметр у сантиметрах (diameter_cm), множник ціни (price_multiplier) та порядок відображення (display_order).

6) Ціни за розмірами (menu_item_prices).

Таблиця використовується для зберігання цін на позиції меню, які можуть мати різні розміри (насамперед піци). Вона пов'язує menu_item_id з menu_items та pizza_size_id з pizza_sizes, а також містить конкретну price для цієї комбінації. Комбінація menu_item_id та pizza_size_id є унікальною.

7) Замовлення (orders).

Центральна таблиця для управління замовленнями клієнтів. Кожне замовлення має унікальний ідентифікатор (id), посилання на стіл (table_id) та офіціанта (waiter_id), кількість клієнтів (customer_count), поточний статус (status — за замовчуванням 'draft'), загальну суму (total_amount), додаткові примітки (notes) та часові мітки створення та останнього оновлення.

8) Позиції замовлення (order_items).

Ця таблиця деталізує склад кожного замовлення. Кожен запис пов'язаний з конкретним order_id (з автоматичним видаленням при видаленні замовлення) та menu_item_id, може посилатися на size_id. Вона містить кількість (quantity), ціну за одиницю (unit_price), загальну ціну позиції (total_price), спеціальні інструкції (special_instructions) та час створення.

9) Платежі (payments).

Таблиця для обліку всіх платежів, що стосуються замовлень. Вона містить унікальний ідентифікатор (id), посилання на order_id та користувача, що обробив платіж (processed_by), суму платежу (amount), метод оплати (payment_method) та

час обробки платежу.

10) Лог дій користувачів (user_actions).

Ця таблиця призначена для ведення детального журналу всіх значущих дій, виконаних користувачами в системі. Вона фіксує ідентифікатор дії (id), посилання на user_id, тип виконаної action, назву таблиці (table_name) та ідентифікатор запису (record_id), до якого застосовувалася дія. Також зберігаються деталі дії у форматі JSONB (details) та IP-адреса, з якої була виконана дія (ip_address), а також часова мітка створення запису.

База даних містить визначені функції та тригери для автоматизації рутинних операцій та підтримки консистентності даних.

Функція update_updated_at_column(): Ця функція, написана мовою PL/pgSQL, автоматично оновлює значення поля updated_at до поточного часу (CURRENT_TIMESTAMP) при кожній зміні запису в таблицях, де вона використовується. Це забезпечує актуальність часових міток останніх змін даних.

Тригер update_users_updated_at на таблиці users: Цей тригер спрацьовує перед оновленням кожного рядка в таблиці users (BEFORE UPDATE FOR EACH ROW) та викликає функцію update_updated_at_column(). Таким чином, поле updated_at для користувача автоматично оновлюється при будь-якій модифікації його даних.

Тригер update_orders_updated_at на таблиці orders: Аналогічно, цей тригер спрацьовує перед оновленням кожного рядка в таблиці orders (BEFORE UPDATE FOR EACH ROW) та викликає функцію update_updated_at_column(), автоматично оновлюючи часову мітку updated_at для замовлення при кожній зміні його даних.

Ця структура бази даних забезпечує надійне зберігання та взаємодію з даними, що є основою для ефективного функціонування програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго».

2.5 Проектування і опис інтерфейсу користувача

Програмне забезпечення «Вікно замовлень офіціанта піцерії Фламінго»

реалізує ряд ключових інтерфейсів користувача, що забезпечують ефективну та інтуїтивну взаємодію з системою.

MainWindow є центральним елементом системи, що об'єднує функціональність в єдиному інтерфейсі. Воно організовано як багатовкладковий інтерфейс з трьома основними розділами. Верхня панель містить меню програми з ролевим доступом та статус-бар з інформацією про поточного користувача. Центральна область представлена як QTabWidget з вкладками "Столи", "Меню" та "Замовлення". Нижня панель відображає інформацію про поточне замовлення та статистику сесії. Інтерфейс автоматично блокує UI для неавторизованих користувачів, динамічно формує меню залежно від ролі, забезпечує real-time оновлення активних замовлень та використовує кольорове кодування статусів. Також відображається ім'я та роль користувача, час активної сесії та індикатори статусу підключення до бази даних. Форма даного вікна зображена на рисунку 2.1.

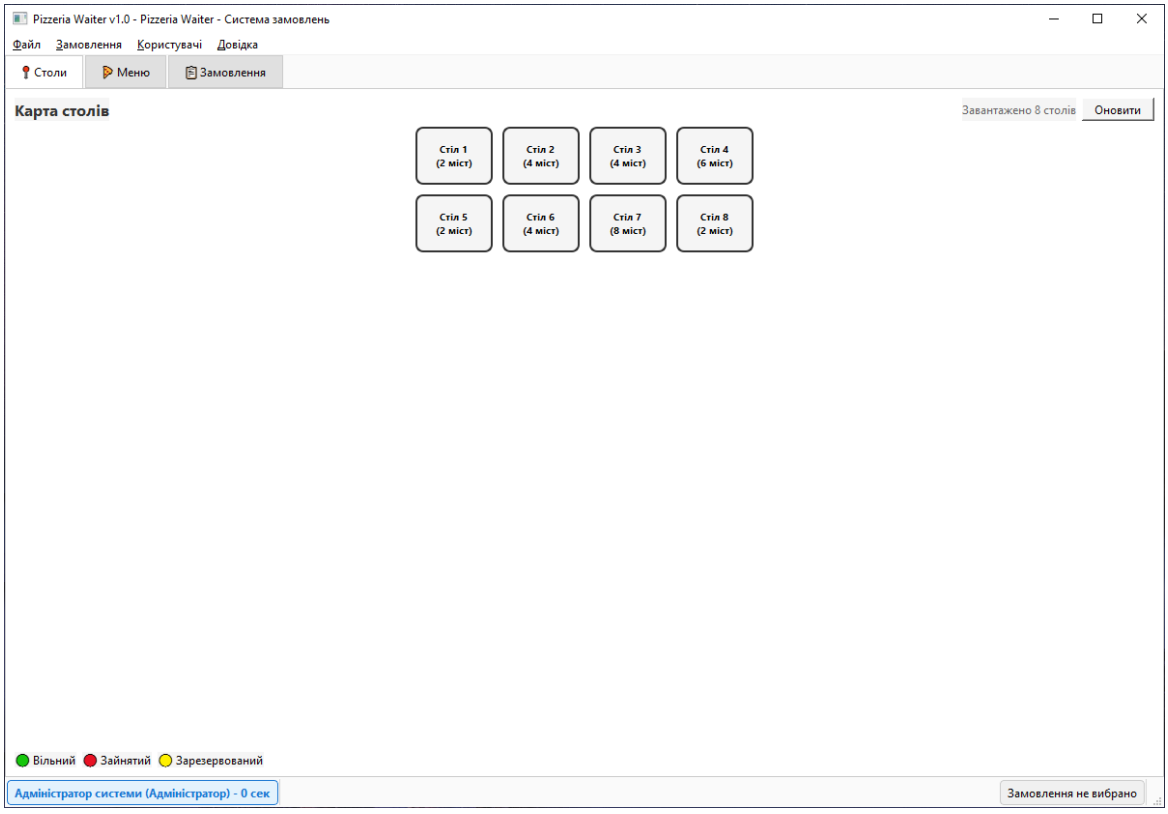


Рисунок 2.1 – Головне вікно програми

TableMapWidget є інтерактивною візуальною картою залу ресторану, що відображає розташування та поточний статус всіх столів. Столи відображаються як кнопки різних форм (прямокутні, круглі, овальні) з кольоровим кодуванням статусів (зелений – вільний, червоний – зайнятий, жовтий – зарезервований). Віджет відображає номер столу та кількість місць, підтримує масштабування, швидкий вибір столу для нового замовлення, tooltip з детальною інформацією та автоматичне оновлення статусів у реальному часі. Також передбачена підтримка різних планувань залу та можливість зміни розташування столів через drag-and-drop. Форма даного вікна зображена на рисунку 2.2.

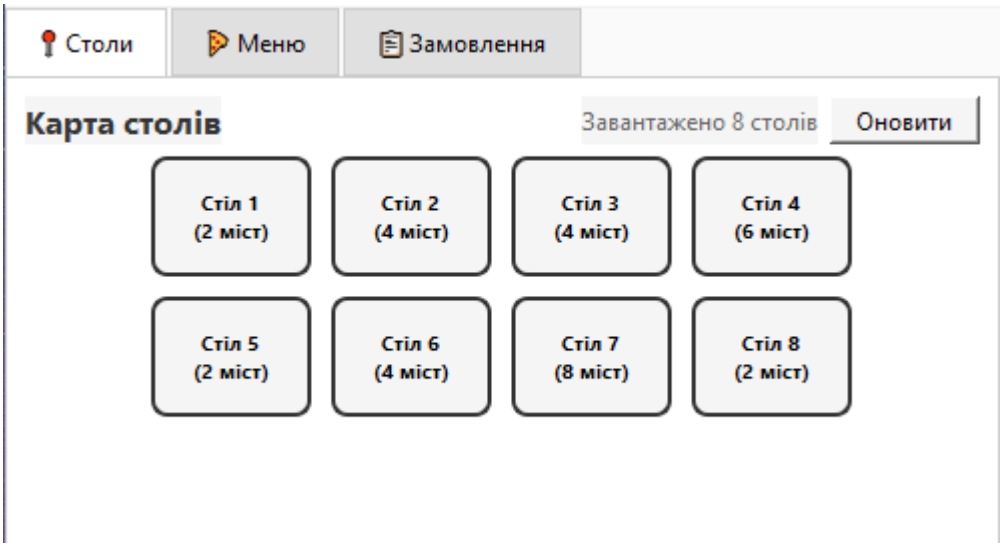


Рисунок 2.2 – Карта столів ресторану

MenuWidget є комплексним віджетом для роботи з меню, організованим у табульовану структуру за категоріями товарів. Він має верхню панель з кнопкою оновлення та пошуком. Основна область містить вкладки категорій (Піца, Напої, Закуски, Десерти), кожна з яких містить прокручувану область з позиціями. Функціональність пошуку дозволяє шукати за назвою страви, фільтрувати за алергенами та дієтичними обмеженнями, сортувати за популярністю, ціною або часом приготування. Також відображаються спеціальні пропозиції та інформація про наявність інгредієнтів. Форма даного вікна зображена на рисунку 2.3.

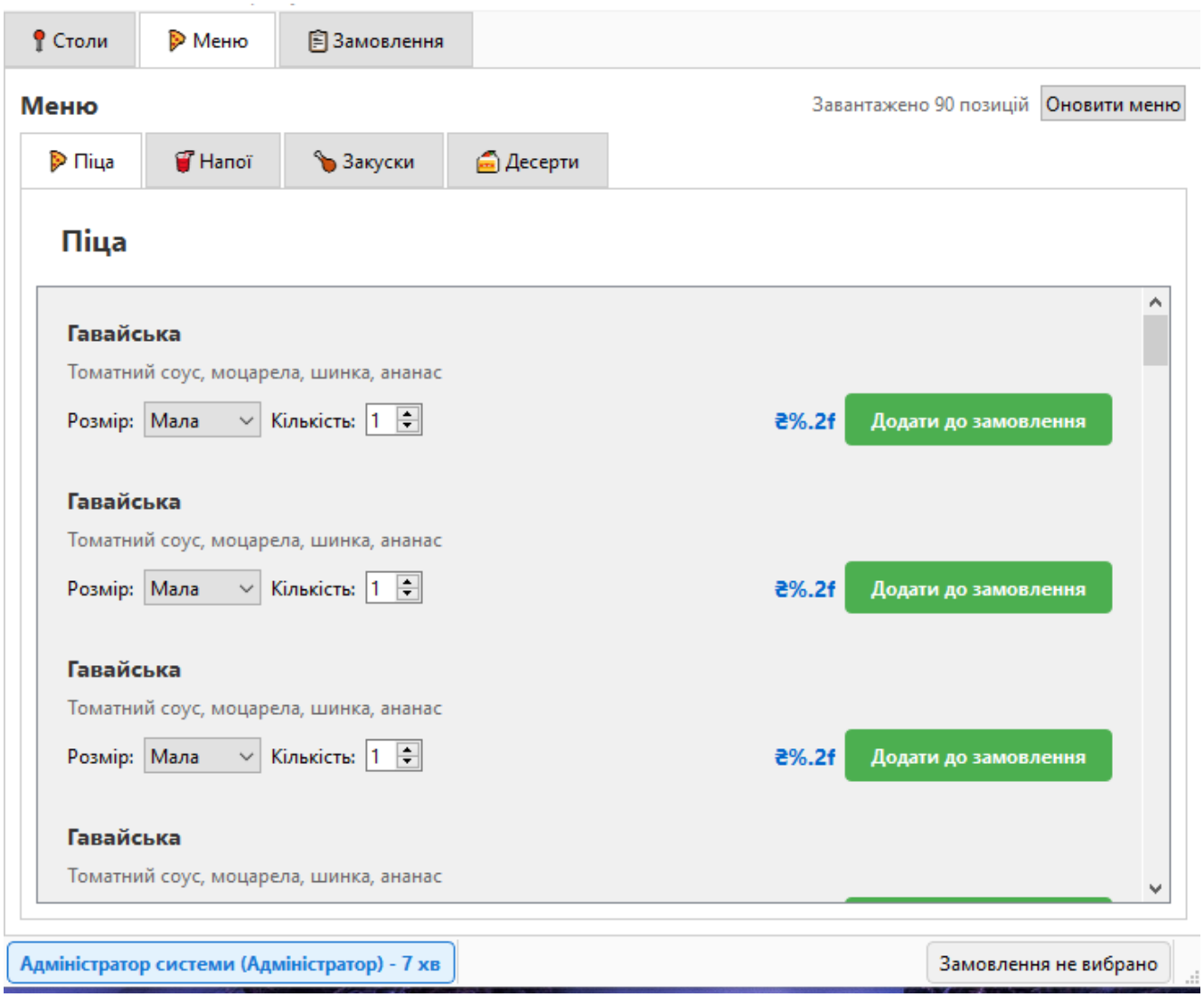


Рисунок 2.3 – Каталог меню ресторану

OrderListWidget є центральним віджетом для відображення та управління активними замовленнями з розширеними можливостями фільтрації та сортування. Верхня панель управління містить фільтри за статусом, пошук та кнопки сортування, а також кнопку оновлення. Область списку замовлень автоматично оновлюється кожні 30 секунд, може групуватися за статусами та виділяти власні замовлення офіціанта. Нижня інформаційна панель відображає загальну кількість активних замовлень, суму продажів за день, середній час обробки замовлення та індикатор останнього оновлення. Віджет підтримує швидке перемикання статусів, друк документів, перехід до деталей та копіювання замовлення. Форма даного вікна зображена на рисунку 2.4.

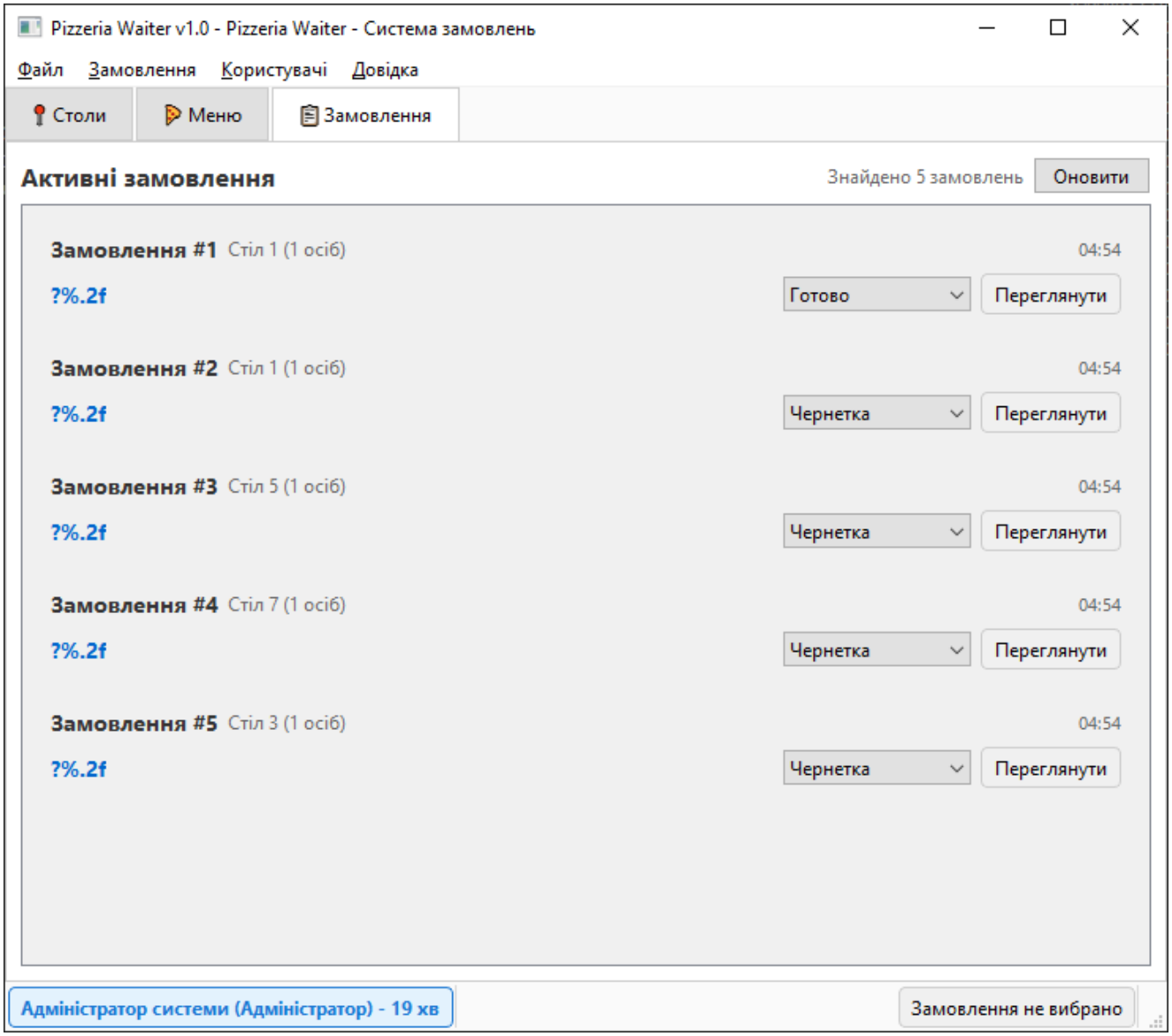


Рисунок 2.4 – Список замовлень

Діалоги авторизації та управління

LoginDialog є модальним діалогом для автентифікації користувачів. Він містить логотип піцерії "Фламінго", назву системи та версію. Форма входу включає поля для логіна (з автозаповненням) та пароля (з можливістю показати/сховати), чекбокс "Запам'ятати логін" та область для відображення помилок. Нижня панель містить кнопки "Увійти" та "Скасувати", а також посилання на відновлення пароля (функція майбутнього розвитку). Діалог підтримує автофокус, клавіатурну навігацію, запам'ятовування останнього успішного логіна та анімації. Форма даного вікна зображена на рисунку 2.5.

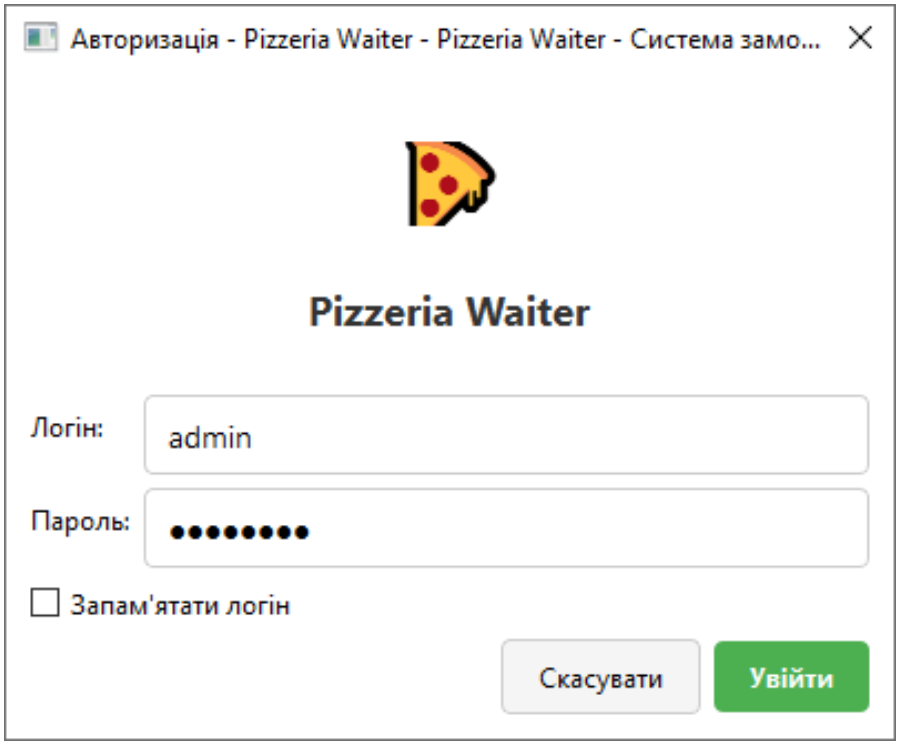


Рисунок 2.5 – Діалог входу в систему

UserManagementDialog – це комплексний діалог для адміністрування користувачів системи. Він має таблицю зі списком користувачів, що містить детальну інформацію та можливості сортування, пошуку та фільтрації. Панель управління дозволяє додавати, редагувати, активувати/деактивувати користувачів та скидати паролі. Інформаційна панель відображає статистику користувачів, графік активності та попередження. Також передбачені можливості експорту списку користувачів у CSV. Форма даного вікна зображена на рисунку 2.6.

UserEditDialog – спеціалізований діалог для створення нових користувачів або редагування існуючих облікових записів. Він містить поля для логіна (з перевіркою унікальності), повного імені, контактних даних та коду працівника. Налаштування доступу включають вибір ролі, статус активності, дату прийняття на роботу та комісійні.

Для безпеки передбачені поля пароля з вимогами до складності та опцією примусової зміни. Діалог забезпечує real-time валідацію даних та індикатори сили пароля. Форма даного вікна зображена на рисунку 2.7.

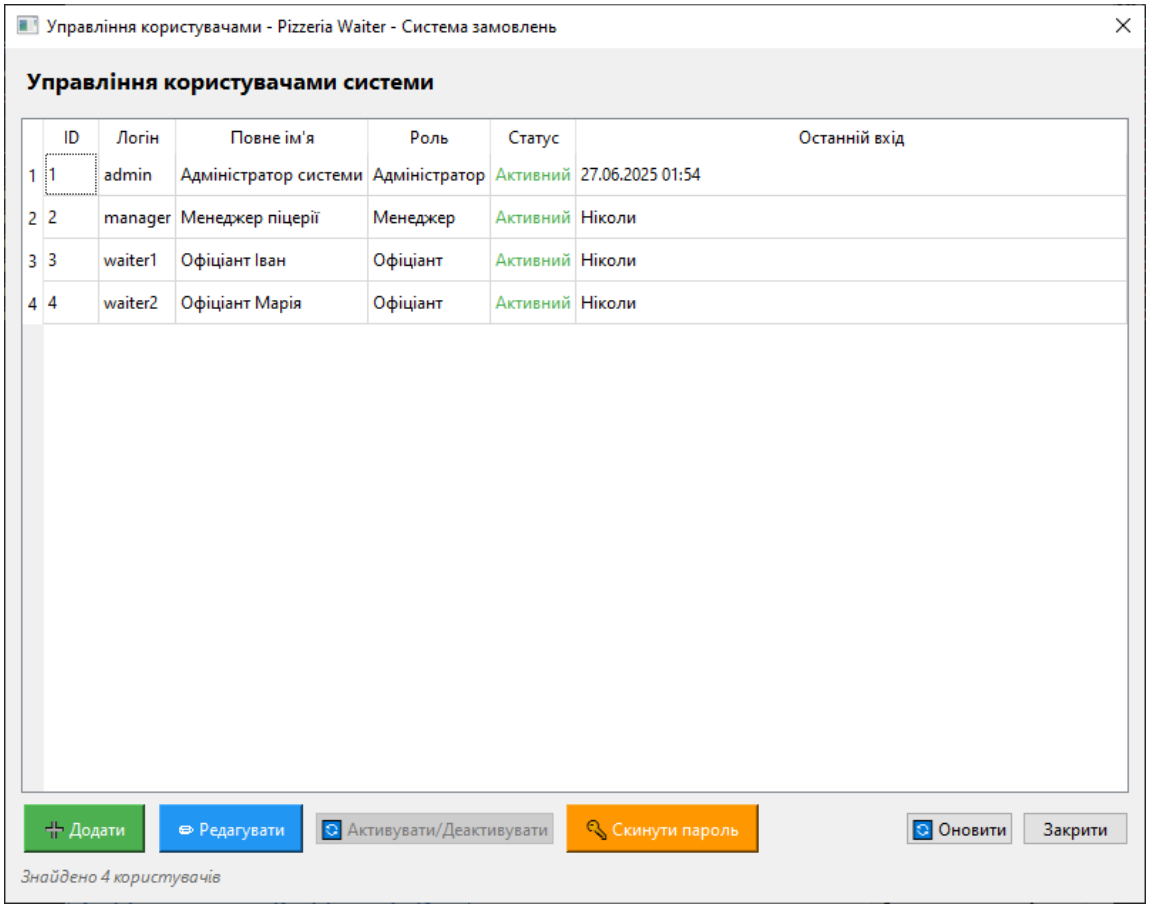


Рисунок 2.6 – Управління користувачами

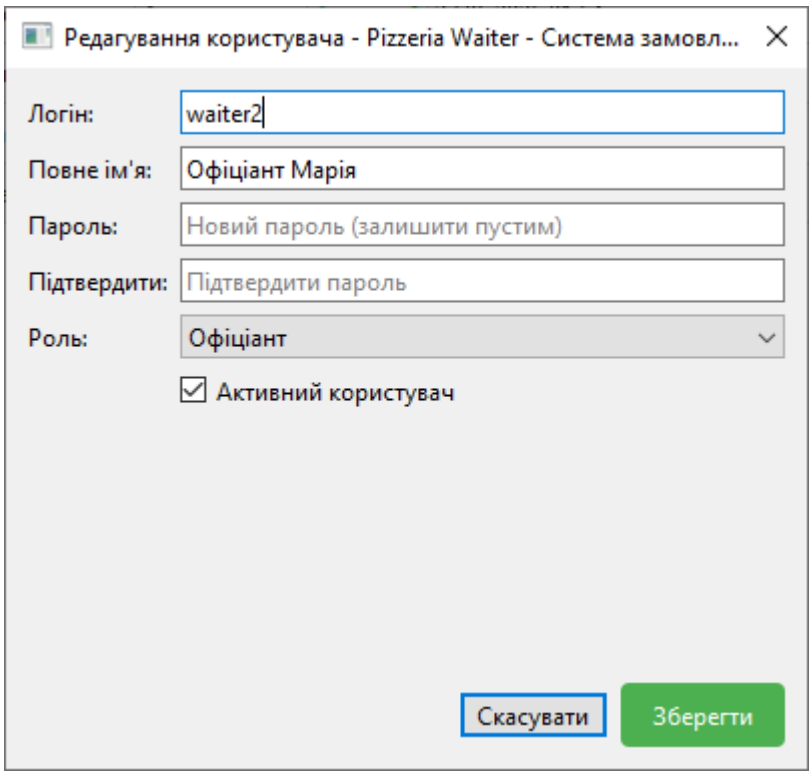


Рисунок 2.7 – Редагування користувача

UserActivityDialog є потужним інструментом аудиту для перегляду та аналізу дій користувачів у системі. Панель фільтрів дозволяє обирати період, тип дії, користувача та здійснювати пошук за ключовими словами. Таблиця журналу відображає час дії, користувача, тип дії (з кольоровим кодуванням), об'єкт дії, детальний опис зміни, IP-адресу та інформацію про пристрій. Панель аналізу надає графік активності, топ користувачів та розподіл дій за типами. Функції експорту дозволяють вивантажувати дані у CSV та генерувати PDF звіти. Форма даного вікна зображена на рисунку 2.8.

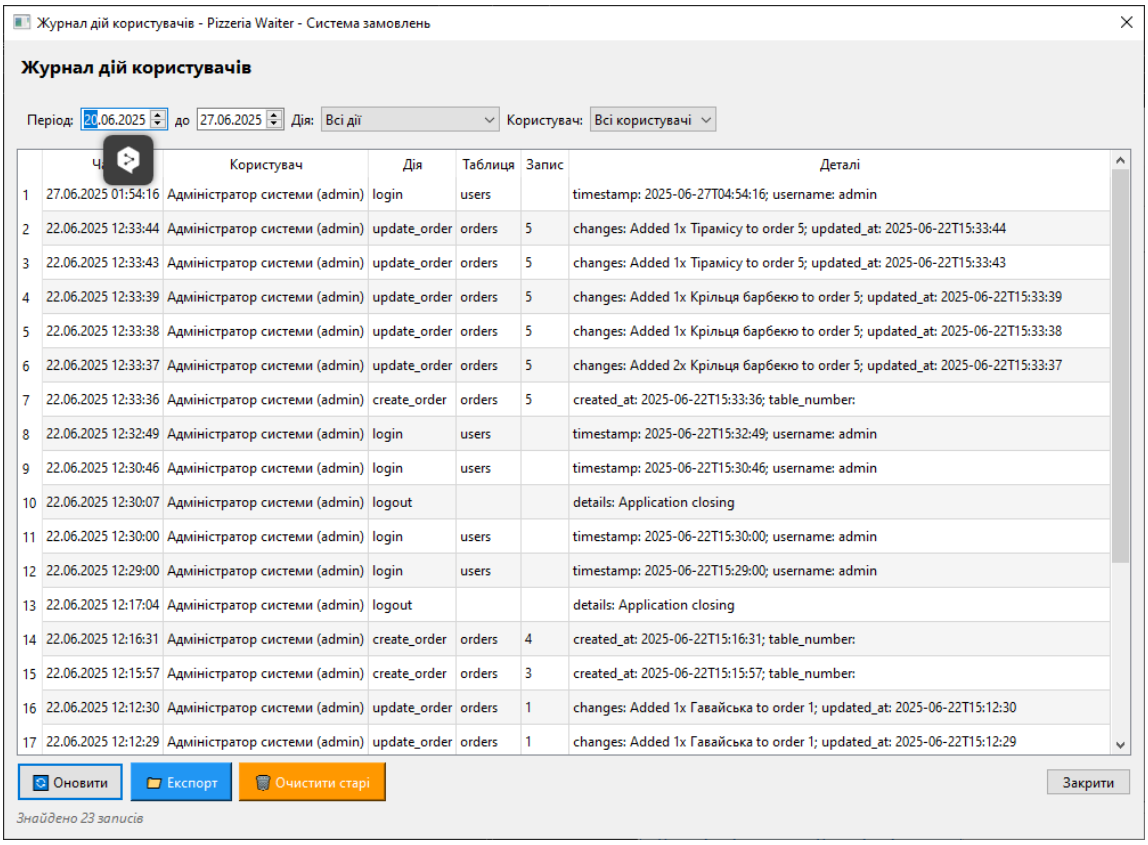


Рисунок 2.8 – Журнал дій користувачів

2.6 Опис файлової структури програми

Файлова структура програми «Вікно замовлень офіціанта піцерії Фламінго»

Файлова структура програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» організована ієрархічно, що забезпечує логічне розділення компонентів та спрощує навігацію по проєкту:

– Корінь проєкту.

На верхньому рівні кореневого каталогу проєкту PizzeriaWaiter/ знаходяться ключові файли, що стосуються загальної конфігурації та документації. Серед них — README.md для основної документації проєкту, PizzeriaWaiter.pro як основний файл проєкту Qt, CMakeLists.txt для альтернативної системи збірки, а також .gitignore, що визначає файли та каталоги, які слід ігнорувати системою контролю версій Git.

– Вихідний код (src/).

Каталог src/ містить весь вихідний код програми. Тут розташований файл main.cpp, який є точкою входу в програму. Всередині src/ знаходяться підкаталоги, що відповідають за функціональні категорії: models/ для моделей даних, services/ для бізнес-логіки, database/ для роботи з базою даних, ui/ для компонентів інтерфейсу користувача, utils/ для допоміжних утиліт та resources/ для ресурсів Qt.

– Моделі даних (models/).

У каталозі models/ зберігаються файли, що визначають структури даних та бізнес-сутності системи. Це включає файли User.h/cpp для моделі користувача, Order.h/cpp для моделі замовлення, OrderItem.h/cpp для позицій замовлення, MenuItem.h/cpp для позицій меню та Table.h/cpp для моделі столу.

– Сервіси (services/).

Каталог services/ містить реалізацію бізнес-логіки системи. Тут розташовані файли AuthService.h/cpp, що відповідають за авторизацію, OrderService.h/cpp для управління замовленнями, UserActionLogger.h/cpp для логування дій користувачів, а також PrintService.h/cpp для функцій друку документів.

– База даних (database/).

У каталозі database/ знаходяться компоненти, що керують взаємодією з базою даних. Це DatabaseManager.h/cpp, що виступає менеджером бази даних. Цей каталог також містить підкаталоги repositories/ та migrations/.

Каталог repositories/ містить класи репозиторіїв, що абстрагують доступ до даних. Тут знаходиться BaseRepository.h/cpp як базовий репозиторій, а також

спеціалізовані репозиторії: UserRepository.h/cpp, OrderRepository.h/cpp, MenuRepository.h/cpp та TableRepository.h/cpp.

Каталог migrations/ містить скрипти або файли, що відповідають за міграції схеми бази даних, дозволяючи контролювати зміни в її структурі.

– Інтерфейс користувача (ui/).

Каталог ui/ містить компоненти користувацького інтерфейсу. Він включає MainWindow.h/cpp, що представляє головне вікно програми, а також підкаталоги widgets/ та dialogs/.

У каталозі widgets/ розташовані окремі віджети, які є складовими частинами інтерфейсу. Це TableMapWidget.h/cpp для карти столів, MenuWidget.h/cpp для віджета меню та OrderListWidget.h/cpp для списку замовлень.

Каталог dialogs/ містить файли, що реалізують різні діалогові вікна. Серед них LoginDialog.h/cpp для входу в систему, UserManagementDialog.h/cpp для управління користувачами, UserEditDialog.h/cpp для редагування даних користувача, UserActivityDialog.h/cpp для перегляду журналу дій, NewOrderDialog.h/cpp для створення нового замовлення та OrderDialog.h/cpp для деталізації замовлення.

– Утиліти (utils/).

Каталог utils/ містить допоміжні утиліти, зокрема Logger.h/cpp для системи логування.

– Ресурси (resources/).

Каталог resources/ містить файли ресурсів програми, об'єднані у app_resources.qrc. Тут розташовані підкаталоги database/ для SQL схем, styles/ для CSS стилів, icons/ для іконок та images/ для інших графічних зображень.

– Документація (docs/).

Каталог docs/ містить додаткову документацію проєкту. Це можуть бути AUTHENTICATION_GUIDE.md як керівництво по авторизації, API_REFERENCE.md як довідник API, а також підкаталог diagrams/ для UML діаграм та інших візуальних представлень архітектури.

Ця структура забезпечує чітке розділення відповідальності між

компонентами та сприяє ефективній розробці та підтримці програмного забезпечення.

2.7 Тестування програми

Процес тестування програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» був проведений з метою забезпечення високої якості продукту, його відповідності вимогам Технічного завдання та готовності до промислової експлуатації. Тестування охопило всі ключові компоненти системи та було виконано послідовно на декількох етапах, що дозволило ідентифікувати та усунути дефекти на різних рівнях інтеграції.

1. Модульне тестування (Unit Testing)

На цьому етапі було виконано перевірку окремих компонентів та модулів системи в ізольованому середовищі. Розробники створювали та запускали набори тестів для кожного класу та функції, зокрема для моделей даних (User, Order, MenuItem), сервісів бізнес-логіки (AuthService, OrderService, PrintService) та репозиторіїв доступу до даних (UserRepository, OrderRepository). Основним завданням було переконатися, що кожен модуль коректно виконує свою передбачену функцію та обробляє різні вхідні дані, включаючи граничні випадки та некоректні параметри. Це дозволило виявити та виправити логічні помилки на ранніх стадіях розробки, перш ніж вони могли б вплинути на взаємодію між модулями.

2. Інтеграційне тестування (Integration Testing)

Після успішного модульного тестування було проведено інтеграційне тестування, яке зосередилося на перевірці взаємодії між модулями та підсистемами. Цей етап включав тестування зв'язків між UI-віджетами та сервісами, а також між сервісами та базою даних через репозиторії. Наприклад, перевірялася коректність збереження нового замовлення в базі даних після його створення через інтерфейс користувача, або правильність отримання списку меню з бази даних для відображення у MenuWidget. Особливу увагу було приділено

перевірці коректності передачі даних між компонентами та обробці можливих помилок при їх взаємодії.

3. Системне тестування (System Testing)

Системне тестування оцінювало програмне забезпечення як єдину, повністю інтегровану систему. На цьому етапі було проведено перевірку відповідності системи всім вимогам Технічного завдання, як функціональним, так і нефункціональним.

Функціональне тестування – були перевірені всі основні сценарії використання, такі як авторизація офіціанта, створення та редагування замовлень, зміна статусів замовлень, розрахунок та оплата, перегляд звітів. Тестування охопило повний цикл життя замовлення від "Чернетка" до "Оплачено".

Тестування продуктивності – перевірявся час відгуку інтерфейсу (не більше 200мс) та час завантаження списку замовлень (не більше 2 секунд). Була імітована одночасна робота до 20 офіціантів для оцінки стабільності системи під навантаженням.

Тестування надійності – перевірялася коректність роботи механізмів автоматичного збереження даних та резервного копіювання. Були проведені тести на стійкість до збоїв, наприклад, імітація раптового вимкнення живлення з подальшою перевіркою цілісності даних.

Загалом, виконаний процес тестування дозволив забезпечити високий рівень якості програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго», підтвердивши його готовність до ефективної та надійної роботи в умовах реальної піцерії.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції програмного забезпечення

Назва: PizzeriaWaiter - Система управління замовленнями піцерії

Версія: 1.0

Розробник: Олег Вальчишин

Підтримувані ОС: Windows 10/11, Ubuntu 20.04+, macOS 10.15+

Мінімальні системні вимоги:

- ОС: Windows 10, Ubuntu 20.04, macOS 10.15.
- Процесор: Intel Core i3 або AMD Ryzen 3.
- Оперативна пам'ять: 4 GB RAM.
- Вільне місце: 2 GB.
- Роздільна здатність: 1280x720.
- Мережа: Ethernet або Wi-Fi для з'єднання з БД.

Рекомендовані системні вимоги:

- ОС: Windows 11, Ubuntu 22.04, macOS 12+.
- Процесор: Intel Core i5 або AMD Ryzen 5.
- Оперативна пам'ять: 8 GB RAM.
- Вільне місце: 5 GB.
- Роздільна здатність: 1920x1080 або вище.
- SSD: Для кращої продуктивності БД.

Передумови встановлення:

1. База даних PostgreSQL

Windows:

```
# Завантажити PostgreSQL з офіційного сайту
# https://www.postgresql.org/download/windows/

# Або через Chocolatey
choco install postgresql
```

```
# Або через Scoop
scoop install postgresql
```

Ubuntu/Debian:

```
bash# Оновлення пакетів
sudo apt update

# Встановлення PostgreSQL
sudo apt install postgresql postgresql-contrib

# Запуск сервісу
sudo systemctl start postgresql
sudo systemctl enable postgresql
macOS:
bash# Через Homebrew
brew install postgresql

# Запуск сервісу
brew services start postgresql
```

2. Qt Runtime

Windows:

– Qt Runtime включено в .msi інсталятор.
– Додаткове встановлення не потрібне.

Linux:

```
# Ubuntu/Debian
sudo apt install qt6-base-dev libqt6sql6-psql

# CentOS/RHEL/Fedora
sudo dnf install qt6-qtbase qt6-qtbase-postgresql
```

Встановлення на Windows

Метод 1: MSI Інсталятор (Рекомендовано):

1) Завантаження:

– Завантажте PizzeriaWaiter-1.0-Windows-x64.msi

– Перевірте цифровий підпис файлу

2) Запуск інсталяції:

```
# Запуск від імені адміністратора
# Подвійний клік по .msi файлу
# Або через командний рядок:
msiexec /i PizzeriaWaiter-1.0-Windows-x64.msi
```

3) Майстер встановлення:

– Прийняття ліцензійної угоди;

– Вибір директорії встановлення (за замовчуванням: C:\Program Files\PizzeriaWaiter);

– Вибір компонентів:

– Основна програма;

– Qt Runtime Libraries;

– PostgreSQL ODBC Driver;

– Демонстраційна база даних;

– Ярлики на робочому столі.

4) Завершення встановлення:

– Створення ярликів у меню Пуск.

– Реєстрація файлових асоціацій.

– Налаштування Windows Firewall.

3.2 Інструкція з використання тестових наборів

Цей документ містить інструкції щодо використання тестових наборів для перевірки функціональності та надійності програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго». Дотримання цих кроків є критично важливим для забезпечення коректності тестування та валідації результатів.

1. Передумови для тестування

Перед початком тестування необхідно переконатися, що виконано такі передумови:

– На тестовому середовищі має бути інстальована остання версія програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго».

– База даних (PostgreSQL або SQLite) має бути розгорнута та сконфігурована відповідно до вимог. Доступ до бази даних із тестового застосунку має бути встановлений.

– База даних має містити попередньо завантажені тестові дані. Це включає інформацію про користувачів (офіціантів, менеджерів, адміністраторів), столи, категорії меню, позиції меню з різними розмірами та модифікаторами. Наявність реалістичних, але анонімізованих даних критична для всебічного тестування.

– Для тестування необхідні дійсні облікові записи користувачів з різними ролями (waiter, manager, admin) та відповідні паролі.

– Якщо використовується віддалена база даних або мережеві принтери, має бути забезпечено стабільне мережеве підключення.

– Принтери чеків (кухонний та фіскальний) мають бути підключені та налаштовані, якщо сценарії тестування передбачають друк.

2. Загальний порядок використання тестових наборів.

Тестові набори організовані за функціональними блоками системи. Для кожного тестового випадку вказано попередні умови, кроки виконання та очікуваний результат:

– Оберіть функціональну область, яку планується тестувати (наприклад, «Авторизація», «Створення замовлення», «Управління столами»).

– Підготовка тестового середовища. Забезпечте, що всі передумови для конкретного тестового набору виконані. Це може включати скидання бази даних до початкового стану або завантаження специфічних тестових даних.

– Послідовно виконуйте кроки кожного тестового випадку, дотримуючись наданих інструкцій.

– Для кожного тестового випадку зафіксуйте фактичний результат. Порівняйте його з очікуваним результатом.

– Логування дефектів. У разі виявлення розбіжностей між фактичним та очікуваним результатом (дефекту), задокументуйте його. Лог дефекту має включати:

  – Унікальний ідентифікатор.
  – Опис проблеми.
  – Кроки для відтворення дефекту.
  – Фактичний результат.
  – Очікуваний результат.
  – Важливість (Severity) та пріоритет (Priority).
  – Скріншоти або відеозаписи (за необхідності).

– Повторне тестування (Regression Testing). Після виправлення дефектів, проведіть повторне тестування виправлених функцій, а також регресійне тестування для впевненості, що виправлення не спричинило нових помилок у раніше стабільних компонентах системи.

3.3 Інструкція з експлуатації програмного забезпечення

Ця інструкція надає загальні настанови щодо експлуатації програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго». Вона призначена для офіціантів, менеджерів та адміністраторів системи, охоплюючи ключові аспекти використання для забезпечення ефективної та безперебійної роботи. Детальніші покрокові інструкції для кожної ролі доступні у відповідних посібниках користувача та адміністратора.

1. Загальні положення

Програмне забезпечення «Вікно замовлень офіціанта піцерії Фламінго» є інструментом для автоматизації процесу приймання, обробки та управління замовленнями. Система інтегрована з базою даних і дозволяє ефективно взаємодіяти з інформацією про столи, меню, користувачів та платежі.

2. Запуск та авторизація

Для початку роботи з програмним забезпеченням виконайте наступні дії:

Запуск програми. Двічі клацніть по ярлику «Вікно замовлень офіціанта» на робочому столі або знайдіть її у меню «Пуск».

Діалог входу (LoginDialog). З'явиться вікно авторизації. Введіть свій логін та пароль у відповідні поля.

Вибір ролі. Система автоматично визначить вашу роль (офіціант, менеджер, адміністратор) після успішної авторизації, надаючи доступ до відповідних функціональних можливостей.

Вхід. Натисніть кнопку «Увійти». У разі успішної авторизації ви будете перенаправлені до головного вікна програми (MainWindow) або спеціалізованого вікна офіціанта (WaiterOrderWindow) залежно від вашої ролі.

3. Робота офіціанта

Основні функції для офіціанта зосереджені на швидкому та ефективному обслуговуванні замовлень.

Управління столами (TableMapWidget). На вкладці «Столи» або у спеціалізованому вікні офіціанта відображається інтерактивна карта столів. Кольорове кодування (наприклад, зелений — вільний, червоний — зайнятий) дозволяє швидко оцінити статус столів. Для створення нового замовлення оберіть вільний стіл.

Створення нового замовлення (NewOrderDialog). Після вибору столу відкриється діалог створення нового замовлення. Вкажіть кількість гостей та, за необхідності, додаткову інформацію.

Робота з меню (MenuWidget). Перейдіть на вкладку «Меню» або до відповідного блоку у вікні офіціанта. Використовуйте категорії та пошук для швидкого знаходження страв. Обирайте позиції меню, вказуйте кількість та, для піц, розмір. Додавайте модифікатори та коментарі за необхідності.

Відправлення замовлення на кухню. Після формування замовлення натисніть кнопку «Надіслати замовлення». Статус замовлення зміниться, і воно буде передано на кухню.

Відстеження та зміна статусу замовлення (OrderListWidget, OrderItemWidget). У списку активних замовлень відстежуйте їхній статус. Після отримання інформації від кухні (наприклад, про готовність) або подачі страв клієнту, оновіть статус замовлення через контекстне меню або відповідну кнопку.

озрахунок та оплата (OrderDialog, payments). Для завершення замовлення оберіть його зі списку та перейдіть до детального перегляду. Розрахуйте загальну суму, застосуйте знижки, додайте чайові та оберіть метод оплати (готівка, картка). Після підтвердження платежу, статус замовлення зміниться на «Оплачено».

4. Робота менеджера та адміністратора

Менеджери та адміністратори мають розширені права доступу для управління системою та аналізу даних.

Управління користувачами (UserManagementDialog). Адміністратори можуть додавати, редагувати, блокувати/активувати облікові записи користувачів, а також скидати паролі. Менеджери можуть переглядати інформацію про користувачів та їх активність.

Перегляд журналів дій (UserActivityDialog). Для контролю та аудиту дій користувачів доступний журнал, який дозволяє фільтрувати події за часом, користувачем та типом дії.

Звітність. Система надає детальні звіти про продажі за період, популярність позицій меню, завантаженість столів та продуктивність офіціантів. Використовуйте ці звіти для аналізу та оптимізації роботи піцерії.

5. Завершення роботи

Перед виходом із системи переконайтеся, що всі активні замовлення оброблені, а дані збережені. Для завершення роботи натисніть кнопку «Вихід» або закрийте програму через меню.

6. Технічне обслуговування та підтримка

Дотримання цих інструкцій забезпечить ефективне та безперебійне використання програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» у повсякденній діяльності піцерії.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини кваліфікаційної роботи є виконання здійснення економічних розрахунків, спрямованих на визначення економічної ефективності розробки програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго».

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Для визначення загальної тривалості проведення НДР доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю 4.1.

Таблиця 4.1 - Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1.	Підготовка технічного завдання	кер. проекту	8
2.	Розробка системи класів	програміст	8
3.	Розробка бази даних сайту	програміст	8
4.	Реалізація методів класів	програміст	60
5.	Розробка інтерфейсу користувача	програміст	8
6.	Тестування програмного забезпечення	тестувальник	16
7.	Документування	кер. проекту	8
8.	Здача програми	програміст	4
		кер.проекту	4
	Р а з о м		124

Сумарний час виконання операцій технічного процесу становить 124 години.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Оплата праці - грошовий вираз вартості і ціни робочої сили, який виступає у формі будь-якого заробітку, виплаченого керівником підприємства найманому працівникові за виконану роботу.

Заробітна плата працівника залежить від кінцевих результатів його роботи, регулюється податками і максимальними розмірами не обмежується.

Основна заробітна плата розраховується за формулою 4.1:

$$З_{осн.} = Т_с \cdot К_г, \quad (4.1)$$

де $Т_с$ – тарифна ставка, грн.; $К_г$ – кількість відпрацьованих годин.

Рекомендовані тарифні ставки: керівник проєкту – 350 грн./год., програміст – 250 грн./год., тестувальник – 150 грн./год.

Отже основна заробітна плата становитиме:

- керівник проєкту: $З_{осн1} = 350 \cdot 20 = 7000$ грн.;
- програміст: $З_{осн2} = 250 \cdot 88 = 22000$ грн.;
- тестувальник: $З_{осн3} = 150 \cdot 16 = 2400$ грн.

Сумарна основна заробітна плата становить:

$$З_{осн} = 7000 + 22000 + 2400 = 31400 \text{ грн.}$$

Додаткова заробітна плата становить 10 - 15% від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.}, \quad (4.2)$$

де $К_{допл.}$ – коефіцієнт додаткових виплат працівникам (приймаємо 15%).

Отже додаткова заробітна плата по категоріях працівників становить:

– керівник проєкту: $З_{дод1} = 7000 \cdot 0,15 = 1050$ грн.;

– програміст: $З_{дод2} = 22000 \cdot 0,15 = 3300$ грн.;

– тестувальник: $З_{дод3} = 2400 \cdot 0,15 = 360$ грн.

Сумарна додаткова заробітна плата становить:

$$З_{дод} = 1050 + 3300 + 360 = 4710 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($В_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п} = 31400 + 4710 = 36110 \text{ грн.}$$

Необхідно визначити відрахування на соціальні заходи:

– фонд страхування на випадок безробіття – 1,6 %;

– фонд по тимчасовій втраті працездатності – 1,4 %;

– пенсійний фонд – 33,2 %;

– внески на страхування від нещасного випадку на виробництві та професійного захворювання - 1,4%.

Загальна сума зазначених відрахувань становить 37,6 %.

Отже, сума нарахувань на заробітну плату буде становити згідно формули 4.4:

$$В_{с.з.} = ФОП \cdot 0,376 \quad (4.4)$$

де ФОП – фонд оплати праці, грн.

$$В_{с.з.} = 36110 \cdot 0,376 = 13577,36 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці зведемо у таблицю 4.2.

Таблиця 4.2 - Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додат-кова заробітна плата, грн.	Нарахув. на ФОП, грн.	Нарахування на оплату праці, грн. 6=3+4+5
		Тарифна ставка, грн.	К-сть від-працьов. год.	Фактично нарах. з/пл., грн.			
1	Кер.проекту	350	20	7000	1050	-	-
2	Програміст	250	88	22000	3300	-	-
3	Тестуваль-ник	150	16	2400	360	-	-
Разом				31400	4710	13577,36	49687,36

Отже, загальні витрати на оплату праці становлять 49687,36 грн.

4.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (формула 4.5):

$$M_{Bi} = q_i \cdot p_i \,, \tag{4.5}$$

де q_i – кількість витраченого матеріалу і-го виду; p_i – ціна матеріалу і-го виду.

Звідси, загальні матеріальні витрати можна визначити за формулою 4.6:

$$З_{м.в.} = \sum M_{B\,i} \,, \tag{4.6}$$

Проведені розрахунки занесемо у таблицю 4.3.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна 1-ці, грн.	Загальна сума витрат, грн.
1	Друк листів А3	листів	6	10	60
2	Друк листів А4	листів	100	2	200
3	DVD-диск	шт.	1	15	15
Р а з о м					275

Загальна сума матеріальних витрат на розробку програмного забезпечення становить 275 грн.

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою 4.7:

$$З_е = W \cdot T \cdot S, \quad (4.7)$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії (приймаємо 7 грн).

Для розробки інтернет-магазину використовувався один персональний комп’ютер. Витрати на електроенергію обчислимо взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт (згідно таблиці 4.1) при вартості 1 кВт електроенергії – 7 грн. та потужністю у 0,45 кВт.

$$З_{ве} = 0{,}45 \cdot 124 \cdot 7 = 390{,}60 \text{ грн.}$$

Витрати на електроенергію становлять 390,60 грн.

4.5 Розрахунок суми амортизаційних відрахувань

Комп’ютери та оргтехніка належать до четвертої групи основних фондів.

Мінімально допустимі строки їх використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу 4.8:

$$А = \frac{Б_В \cdot Н_А}{100\%} \cdot T, \quad (4.8)$$

де А – амортизаційні відрахування за звітний період, грн.; $Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.; Т - кількість годин роботи обладнання, год.; $Н_А$ – норма амортизації .

Оскільки для написання програми та її тестування використовується один ПК вартістю 35000 грн., тоді сума амортизаційних відрахувань становить:

$$А = \frac{35000 \cdot 0{,}04}{150} \cdot 124 = 1157{,}33 \text{ грн.}$$

4.6 Обчислення накладних витрат

Накладні витрати - це витрати, не пов'язані безпосередньо з технологічним процесом виготовлення продукції, а утворюються під впливом певних умов роботи по організації, управлінню та обслуговуванню виробництва.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20 – 60 % від суми основної та додаткової заробітної плати працівників, обчислюються за формулою 4.9:

$$Нв = Воп. \cdot 0{,}2 \ldots 0{,}6, \quad (4.9)$$

де $Н_В$ – накладні витрати.

$$Н_В = 36110 \cdot 0{,}2 = 7222 \text{ грн.}$$

4.7 Складання кошторису витрат та визначення собівартості НДР

Кошторис витрат являє собою зведений план усіх витрат підприємства на

майбутній період виробничо-фінансової діяльності.

Результати проведених вище розрахунків зведемо у таблиці 4.4.

Таблиця 4.4 - Кошторис витрат

Зміст витрат	Сума, грн.	В % до загальної суми
Витрати на оплату праці	36110,00	61,47
Відрахування на соціальні заходи	13577,36	23,12
Матеріальні витрати	275,00	0,47
Витрати на електроенергію	390,60	0,67
Амортизаційні відрахування	1157,33	1,97
Накладні витрати	7222,00	12,30
Собівартість	58732,29	100

Собівартість ($С_В$) НДР знайдемо за формулою 4.10:

$$С_В = В_{О.П.} + В_{С.З.} + З_М + З_е + А + Н_в \quad (4.10)$$

Отже, собівартість дорівнює: C_B = 58732,29 грн.

4.8 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою 4.11:

$$Ц = Св \cdot (1+Р_{рен}) \cdot (1+ПДВ), \quad (4.11)$$

де Св – собівартість розробки; $Р_{рен.}$ – рівень рентабельності, (30 %); ПДВ – ставка податку на додану вартість, (20 %).

$$Ц = 58732,29 \cdot (1 + 0,3) \cdot (1 + 0,2) = 91622,37 \text{ грн.}$$

4.9 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Прибуток розраховується за формулою 4.12:

$$П = Ц – С_В \quad (4.12)$$

$$П = 91622,37 – 58732,29 = 32890,08 \text{ грн.}$$

Економічна ефективність (Ер) полягає у відношенні результату виробництва до затрачених ресурсів і розраховується за формулою 4.13:

$$Е_р = П / С_в \quad (4.13) \quad (4.13)$$

де П– прибуток; $С_в$ – собівартість.

$$Ер = 32890,08 / 58732,29 = 0,56$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень $Т_р$ (формула 4.14)

$$Т_р = 1 / Ер \quad (4.14)$$

Допустимим вважається термін окупності до 5 років. В даному випадку:

$$Т_р=1/0,56 = 1,8.$$

Всі дані внесемо в зведену таблицю 4.5 техніко-економічних показників

Таблиця 4.5 - Техніко-економічні показники розробки програмного забезпечення

№ п/п	Показник	Значення
1.	Собівартість, грн.	58732,29
2.	Плановий прибуток, грн.	32890,08
3.	Ціна, грн.	91622,37
4.	Економічна ефективність, грн.	0,56
5.	Термін окупності, рік	1,8

Загальна вартість розробки програмного забезпечення становить 91622,37 грн. Зважаючи на високі показники економічної ефективності - 0,56, кошти, вкладені в розробку програмного забезпечення окупляться за 1,8 року.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 Розрахунок системи штучного освітлення для приміщення, де здійснюється розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»

Ефективне та належне освітлення є одним із ключових чинників, що визначають комфорт, продуктивність і, найголовніше, безпеку праці у будь-якому виробничому чи офісному приміщенні. Особливого значення це набуває в приміщеннях, де здійснюється розробка програмного забезпечення, оскільки робота програміста пов'язана з високим зоровим навантаженням і тривалим перебуванням за монітором. Недостатнє, надмірне або неправильно організоване освітлення може призвести до швидкої втоми очей, головного болю, зниження концентрації уваги та загального самопочуття, що, в свою чергу, негативно позначається на якості виконуваної роботи та може спричинити розвиток професійних захворювань.

Залежно від джерела світла виробниче освітлення може бути: природним, що створюється прямими сонячними променями та розсіяним світлом небосхилу; штучним, що створюється електричними джерелами світла та суміщеним, при якому недостатнє за нормами природне освітлення доповнюється штучним.

Штучне освітлення може бути загальним та комбінованим. Загальним називають освітлення, при якому світильники розміщуються у верхній зоні приміщення (не нижче 2,5м над підлогою) рівномірно (загальне рівномірне освітлення) або з врахуванням розташування робочих місць (загальне локалізоване освітлення). Комбіноване освітлення складається із загального та місцевого. Його доцільно застосовувати при роботах високої точності, а також, якщо необхідно створити певний або змінний, в процесі роботи, напрямок світла. Місцеве освітлення створюється світильниками, що концентрують світловий потік безпосередньо на робочих місцях. Застосування лише місцевого освітлення не допускається з огляду на небезпеку виробничого травматизму та професійних

захворювань.

За функціональним призначенням штучне освітлення поділяється на робоче, аварійне, евакуаційне, охоронне, чергове.

Штучне освітлення передбачається у всіх виробничих та побутових приміщеннях, де недостатньо природного світла, а також для освітлення приміщень в темний період доби. При організації штучного освітлення необхідно забезпечити сприятливі гігієнічні умови для зорової роботи і одночасно враховувати економічні показники.

Найменша освітленість робочих поверхонь у виробничих приміщеннях визначається, в основному, характеристикою зорової роботи.

В якості джерел штучного освітлення широко використовують лампи розжарювання та газорозрядні лампи.

Лампи розжарювання відносяться до теплових джерел світла. Під дією електричного струму нитка розжарювання (вольфрамовий дріт) нагрівається до високої температури і випромінює потік променевої енергії. Ці лампи характеризуються простотою конструкції та виготовлення, відносно низькою вартістю, зручністю експлуатації, широким діапазоном напруг та потужностей.

Поряд з перевагами їм притаманні і суттєві недоліки - велика яскравість (засліплююча дія); низька світлова віддача (7—20 лм/Вт); відносно малий термін експлуатації (2,5 тис. год); переважання жовто-червоних променів в порівняні з природним світлом; висока температура нагрівання до 140°С і вище що робить їх пожежонебезпечними.

Лампи розжарювання використовують, як правило, для місцевого освітлення, а також освітлення приміщень з тимчасовим перебуванням людей.

Газорозрядні лампи внаслідок електричного розряду в середовищі інертних газів і парів металу та явища люмінесценції випромінюють світа оптичного діапазону спектру.

Основною перевагою газорозрядних ламп є їх економічність.

Світлова віддача цих ламп становить 40—100 лм/Вт, що в 3 рази перевищує світлову віддачу ламп розжарювання. Термі експлуатації — до 10 тис. год., а

температура нагрівання (люмінесцентні) — 30-60 °С. Окрім того, газорозрядні ламп забезпечують світловий потік практично будь-якого спектра, шляхом підбирання відповідним чином інертних газів, парів металу, люмінофору. Так, за спектральним складом видимого світла розрізняють люмінесцентні лампи: денного світла (ЛД), денного світла з покращеною передачею кольорів (ЛДЦ), холодного білого (ЛХБ), теплого білого (ЛТБ) та білого (ЛБ) кольорів.

Основним недоліком газорозрядних ламп є пульсація світлового потоку, що може зумовити виникнення стробоскопічного ефекту, котрі полягає у спотворенні зорового сприйняття об'єктів, що рухаються обертаються. До недоліків цих ламп можна віднести також складність схеми включення, шум дроселів, значний час між включенням та запалюванням ламп, відносна дороговизна.

При проектуванні штучного освітлення необхідно вирішити наступне: вибрати систему освітлення, тип джерела світла, тип світильників, визначити розташування світлових приладів, виконати розрахунки штучного освітлення та визначити потужності світильників та ламп [3, с. 50-54].

Розраховуємо систему загального рівномірного освітлення люмінесцентними лампами для приміщення офісного приміщення «open space» (або «відкритий простір), в якому виконуються зорові роботи дуже високої точності ($II_г$), мінімальне освітлення якого становить Е = 300лк. При виконанні при розрахунку освітлення необхідно використовувати ДБН В.2.5-28:2018 для загальних вимог та ДСТУ EN 12464-1:2016 для специфічних вимог до освітлення внутрішніх робочих місць, особливо тих, що обладнані візуальними дисплейними терміналами. Ці нормативні документи визначають вимоги до нормованої освітленості, якісних показників освітлення (рівномірність, показник осліпленості, коефіцієнт пульсації), а також до вибору типів світильників та їх розташування. Правильний розрахунок дозволить створити оптимальні світлові умови, які сприятимуть збереженню зору працівників, підвищенню їхньої працездатності та забезпеченню відповідності робочих місць нормам охорони праці.

Як світлові пристрої приймаємо світильники типу ЛПО01 (з двома

лампами). Як джерело світла при штучному освітленні застосовуватимуться люмінесцентні лампи типу ЛБ-40.

Розрахунок виконаємо методом коефіцієнта використання світлового потоку.

Загальне освітлення має бути виконане у вигляді суцільних або переривчастих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників.

Розміри приміщення «відкритого простору», де встановлено 10 ПК та 2 мережевих принтери: довжина а=14,90м. (прийнята усереднено для спрощення розрахунків), ширина b=7,70м, висота Н=3м. Приміщення має такі показники: коефіцієнт відбиття $\rho_{стелі}$=70%, $\rho_{стін}$=50%.

Висота робочих поверхонь (столів) h_p=0,75м. Оскільки світильники кріпляться до стелі, то їх висота над підлогою майже рівна висоті приміщення h_0=3м, що не суперечить вимогам ДБН В.2.5-28-2006, відповідно до яких h_{0min}=2,6-4 м.

Скориставшись формулою 5.1 визначимо висоту світильника над робочою поверхнею:

$$h = h_0 - h_p = 3 - 0{,}75 = 2{,}25 \text{ м} \qquad (5.1)$$

Показник приміщення *i* за формулою 5.2 становить:

$$i = \frac{a \cdot b}{h(a+b)} = \frac{14{,}90 \cdot 7{,}70}{2{,}25(14{,}90+7{,}70)} = 2{,}26 \qquad (5.2)$$

При *i* = 2,25 (*i* =2,26 немає), $\rho_{стелі}$= 70 %, $\rho_{стін}$= 50% для світильника ЛПО01 коефіцієнт використання дорівнює η =0,63. Ці дані отримано згідно таблиці коефіцієнтів використання світлового потоку світильників з люмінісцентними лампами.

Визначимо необхідну кількість світильників, для забезпечення необхідної

нормованої освітленості робочих поверхонь, якщо відомо, що в кожному світильнику встановлено по дві лампи ЛБ-40, а світловий потік однієї такої лампи становить $Ф_Л = 3200$ лм:

$$N = \frac{ESK_з Z}{2Ф_з} = \frac{300 \cdot 114{,}73 \cdot 1{,}5 \cdot 1{,}12}{2{,}25 \cdot 3200} = 14{,}3 \qquad (5.3)$$

Приймаємо 15 світильників, які для забезпечення рівномірності освітлення розташовуємо в 3 ряди, симетрично до стін. Оскільки довжина світильника становить 1м, то між ним будуть рівномірні проміжки. Розміщення світильників наведено на рисунку 5.1

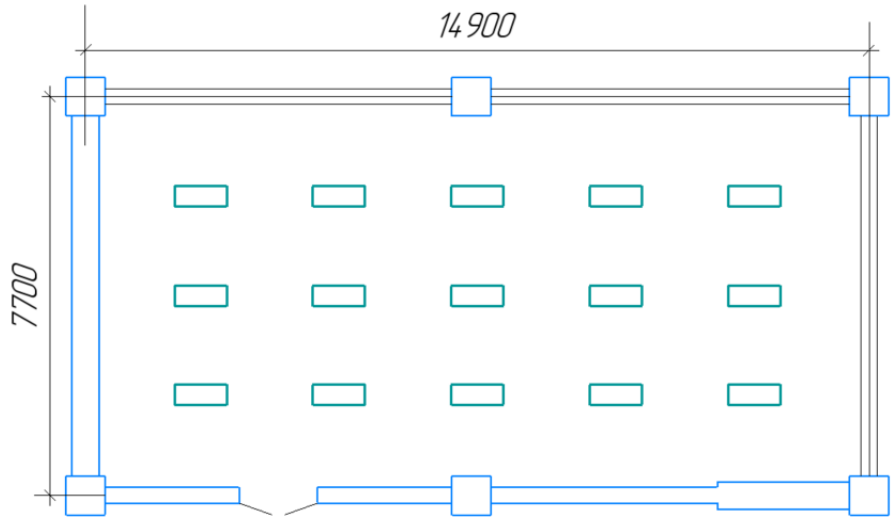


Рисунок 5.1 - Схема розміщення світильників

5.2 Організація робочого місця програміста

Правильна організація робочого місця програміста має вирішальне значення не лише для підвищення продуктивності, але й для збереження здоров'я та запобігання професійним захворюванням. Це комплексний підхід, що охоплює ергономіку, освітлення, мікроклімат, психологічний комфорт та навіть звички самого працівника, при цьому важливо враховувати вимоги, встановлені Державними будівельними нормами (ДБН).

Ергономіка та правильне положення тіла.

Центральним елементом будь-якого робочого місця є стіл та крісло. Крісло повинно бути ергономічним, з регульованою висотою сидіння, спинки та підлокітників, забезпечуючи адекватну підтримку для попереку. Згідно з вимогами ДСТУ EN 12464-1:2016, які стосуються умов праці з комп'ютерами, спина програміста має бути прямою, стопи повністю стояти на підлозі або на спеціальній підставці. Важливо, щоб стегна були паралельні підлозі, а коліна зігнуті під кутом приблизно 90 градусів.

Стіл повинен бути достатньо великим, щоб розмістити все необхідне обладнання, залишаючи при цьому простір для комфортної роботи. Оптимальна висота столу дозволяє ліктям знаходитись під кутом 90 градусів, коли руки лежать на клавіатурі. Дедалі популярнішими стають столи з регульованою висотою, що дають можливість працювати як сидячи, так і стоячи, запобігаючи застою крові та розвантажуючи хребет (див. рив. 5.2).

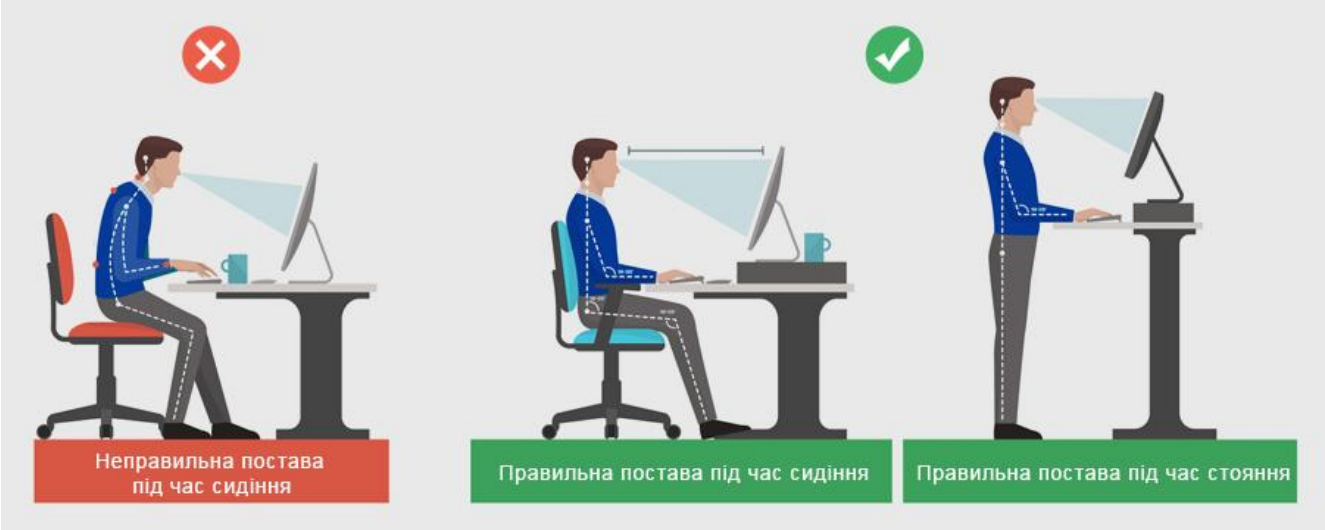


Рисунок 5.2 – Не правильна і правильні постави під час роботи за комп’ютером

Монітор слід розташовувати безпосередньо перед програмістом, на відстані витягнутої руки (приблизно 50-70 см). Згідно з ДСТУ EN 12464-1:2016, верхній край екрану повинен знаходитись на рівні очей або трохи нижче, щоб уникнути напруги шиї та очей. Якщо використовується кілька моніторів, їх слід розмістити

таким чином, щоб основний екран знаходився по центру, а додаткові – по боках під невеликим кутом. Важливо також зменшити відблиски на екрані, розташувавши його перпендикулярно до вікон та інших джерел світла.

Клавіатура та миша повинні бути зручними та дозволяти природне положення кистей рук. Рекомендується використовувати ергономічні моделі, які мінімізують напругу зап'ястя. Руки повинні вільно лежати на клавіатурі, не згинаючись у зап'ястях.

Освітлення та мікроклімат.

Освітлення відіграє ключову роль у запобіганні втоми очей. Згідно з ДБН та ДСТУ EN 12464-1:2016, які регламентують норми природного та штучного освітлення для приміщень, ідеальне освітлення – це комбінація природного та штучного світла. Природне світло має бути достатнім, але не сліпучим. Вікна повинні мати жалюзі або штори для регулювання інтенсивності світла. Штучне освітлення повинно бути рівномірним, без мерехтіння, і розташовуватися таким чином, щоб не створювати відблисків на екрані. Рекомендується використовувати джерела світла з регульованою яскравістю та колірною температурою. Загальне освітлення приміщення має бути доповнене місцевим освітленням, наприклад, настільною лампою, яка дозволить програмісту підлаштовувати освітлення під конкретні завдання.

Мікроклімат на робочому місці також впливає на самопочуття та продуктивність. Оптимальна температура повітря повинна бути в межах 20-24 градусів Цельсія, вологість – 40-60%. Важливо забезпечити регулярне провітрювання приміщення. Сухе повітря може призвести до сухості очей, тому в деяких випадках доцільно використовувати зволожувачі повітря. Системи вентиляції та кондиціонування повинні відповідати вимогам ДБН щодо забезпечення належного повітрообміну.

Організація простору та порядок.

Робоче місце програміста повинно бути організованим і чистим. Зайві предмети відволікають увагу і створюють безлад. Кабелі повинні бути акуратно укладені та зафіксовані, щоб уникнути плутанини та потенційних небезпек, а

також відповідати вимогам пожежної безпеки, які також частково регламентуються ДБН. Наявність достатнього місця для зберігання документів та особистих речей сприяє порядку.

Регулярні перерви та активність.

Навіть ідеально організоване робоче місце не може замінити регулярні перерви. Згідно з трудовим законодавством та рекомендаціями ДСТУ EN 12464-1:2016 для працівників, які працюють з комп'ютерами, програмістам, які проводять багато часу в сидячому положенні, необхідно робити короткі перерви кожні 45-60 хвилин. Під час цих перерв рекомендується встати, розім'ятись, зробити кілька простих вправ для очей та шиї, пройтись. Це допомагає запобігти м'язово-скелетним розладам, покращує кровообіг та зменшує напругу очей.

Психологічний комфорт.

Крім фізичних аспектів, важливо враховувати і психологічний комфорт. Робоче місце повинно бути вільним від зайвого шуму, який може відволікати та створювати стрес. Допустимі рівні шуму в офісних приміщеннях також регламентуються відповідними ДБН та ДСТУ EN 12464-1:2016. Використання звукоізолюючих матеріалів або навушників з шумозаглушенням може бути корисним. Деякі програмісти знаходять комфорт у персоналізації свого робочого простору, додаючи рослини, фотографії або предмети, які створюють приємну атмосферу, що сприяє загальному добробуту.

Дотримання цих рекомендацій щодо організації робочого місця програміста, з урахуванням вимог Державних будівельних норм та Державних санітарних норм та правил України, дозволяє створити умови, які сприяють не лише ефективній та продуктивній праці, але й збереженню здоров'я та добробуту фахівця. Це інвестиція в довгострокову кар'єру та якість життя.

ВИСНОВКИ

Виконана розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» стала значним кроком до автоматизації та оптимізації операційних процесів у закладі. Завдяки реалізації комплексного рішення, система забезпечує ефективне управління циклом замовлень, від його створення до фінального розрахунку. Це досягнуто за рахунок впровадження інтуїтивно зрозумілого інтерфейсу користувача, що включає головне вікно, спеціалізоване вікно офіціанта, інтерактивну карту столів та повнофункціональний каталог меню.

Система успішно інтегрована з надійною базою даних на PostgreSQL, що забезпечує цілісність та збереження даних, а також підтримує розширений workflow статусів замовлень. Застосовані архітектурні принципи, такі як MVC та Repository Pattern, у поєднанні з принципами SOLID, гарантують високу якість коду, його гнучкість, розширюваність та легкість у подальшій підтримці та модифікації.

Впровадження програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго» дозволить піцерії досягти таких техніко-економічних переваг, як скорочення часу обслуговування клієнтів, мінімізація кількості помилок у замовленнях, оптимізація робочого часу персоналу та підвищення загальної пропускної здатності закладу. Керівництво отримуватиме доступ до детальної аналітики, що сприятиме прийняттю обґрунтованих управлінських рішень та підвищенню рентабельності.

Таким чином, розроблене програмне забезпечення є потужним інструментом для модернізації та підвищення ефективності діяльності піцерії «Фламінго», забезпечуючи значну конкурентну перевагу на ринку ресторанного бізнесу.

ПЕРЕЛІК ПОСИЛАНЬ

1) Автоматизуй свій заклад разом з R-KeeperUa. Автоматизуй свій заклад разом з R-KeeperUa. URL: https://rkeeper.com.ua/?gad_source=1 (дата звернення: 10.06.2025).

2) Poster POS. URL: https://joinposter.com/ua (дата звернення: 10.06.2025).

3) Waiterio. URL: https://www.waiterio.com/uk/ (дата звернення: 11.06.2025).

4) Автоматизація кафе, барів та ресторанів: програма (софт) обліку кафе та ресторану | Knaipa-Service. Knaipa Service. URL: https://knaipa-service.pro/ (дата звернення: 11.06.2025).

5) Буч Г., Об'єктно-орієнтований аналіз і проєктування з прикладами застосувань на C++ : навч. посібник. Харків : Вид-во Харківського Технічного Університету, 2017. 600 с.

6) Седжвік Р., Алгоритми на C++. Фундаментальні алгоритми і структури даних : навч. посібник. Київ : Вид-во Київського Університету, 2020. 450 с.

7) Топп У., Форд У., Структури даних в C++ : навч. посібник. Київ : Вид-во Київського Політехнічного Інституту, 2019. 380 с.

8) Гамма Е., Хелм Р., Джонсон Р., Вліссидес Дж., Прийоми об'єктно-орієнтованого проєктування. Шаблони проєктування : навч. посібник. Дніпро : Вид-во Дніпровського Університету, 2016. 520 с.

9) Мейерс С., Ефективне використання C++ : навч. посібник. Одеса : Вид-во Одеського Політехнічного Університету, 2015. 450 с.

10) Страуструп Б., Мова програмування C++. Спеціальне видання : навч. посібник. Львів : Вид-во Львівського Університету, 2018. 720 с.

11) SQLite Home Page. SQLite Home Page. URL: https://sqlite.org/ (дата звернення: 01.05.2025).

12) Embedded Software Development Tools & Cross Platform IDE | Qt Creator. Qt | Tools for Each Stage of Software Development Lifecycle. URL:

https://www.qt.io/product/development-tools (дата звернення: 10.05.2025).

13) Home Page | EBSCO. EBSCO Information Services, Inc. URL: https://www.ebsco.com/ (дата звернення: 14.04.2025)..

14) ProQuest | Better research, better learning, better insights. ProQuest | Better research, better learning, better insights. URL: https://www.proquest.com/ (дата звернення: 15.04.2025).

15) JSTOR. JSTOR Home. URL: https://www.jstor.org/ (дата звернення: 16.04.2025).

ДОДАТКИ

Додаток А. Лістинг файлу «PizzeriaWaiter.pro»

```
# PizzeriaWaiter.pro
# Qt project file for PizzeriaWaiter - Pizza restaurant order management system

QT += core widgets sql printsupport network
CONFIG += c++17

TARGET = PizzeriaWaiter
TEMPLATE = app

# Версія проекту
VERSION = 1.0.0
QMAKE_TARGET_PRODUCT = "Pizzeria Waiter"
QMAKE_TARGET_DESCRIPTION = "Pizza restaurant order management system with authentication"
QMAKE_TARGET_COPYRIGHT = "Copyright (c) 2025 Pizzeria Team"

# Конфігурація збірки
CONFIG += warn_on
CONFIG += qt
CONFIG += thread

# Включення підтримки великих файлів
CONFIG += largefile

# Опціональна підтримка spdlog (розкоментувати якщо встановлено)
# CONFIG += spdlog
# spdlog {
#     DEFINES += USE_SPDLOG
#     LIBS += -lspdlog
```

```
# }

# Заголовкові файли
HEADERS += \
    src/models/User.h \
    src/models/Order.h \
    src/models/OrderItem.h \
    src/models/MenuItem.h \
    src/models/Table.h \
    src/database/DatabaseManager.h \
    src/database/repositories/BaseRepository.h \
    src/database/repositories/UserRepository.h \
    src/database/repositories/OrderRepository.h \
    src/database/repositories/MenuRepository.h \
    src/database/repositories/TableRepository.h \
    src/services/AuthService.h \
    src/services/OrderService.h \
    src/services/UserActionLogger.h \
    src/services/PrintService.h \
    src/ui/MainWindow.h \
    src/ui/dialogs/LoginDialog.h \
    src/ui/dialogs/UserManagementDialog.h \
    src/ui/dialogs/UserEditDialog.h \
    src/ui/dialogs/UserActivityDialog.h \
    src/ui/dialogs/NewOrderDialog.h \
    src/ui/dialogs/OrderDialog.h \
    src/ui/widgets/TableMapWidget.h \
    src/ui/widgets/MenuWidget.h \
    src/ui/widgets/OrderListWidget.h \
    src/utils/Logger.h

# Файли реалізації
SOURCES += \
    src/main.cpp \
```

```
    src/models/User.cpp \
    src/models/Order.cpp \
    src/models/OrderItem.cpp \
    src/models/MenuItem.cpp \
    src/models/Table.cpp \
    src/database/DatabaseManager.cpp \
    src/database/repositories/BaseRepository.cpp \
    src/database/repositories/UserRepository.cpp \
    src/database/repositories/OrderRepository.cpp \
    src/database/repositories/MenuRepository.cpp \
    src/database/repositories/TableRepository.cpp \
    src/services/AuthService.cpp \
    src/services/OrderService.cpp \
    src/services/UserActionLogger.cpp \
    src/services/PrintService.cpp \
    src/ui/MainWindow.cpp \
    src/ui/dialogs/LoginDialog.cpp \
    src/ui/dialogs/UserManagementDialog.cpp \
    src/ui/dialogs/UserEditDialog.cpp \
    src/ui/dialogs/UserActivityDialog.cpp \
    src/ui/dialogs/NewOrderDialog.cpp \
    src/ui/dialogs/OrderDialog.cpp \
    src/ui/widgets/TableMapWidget.cpp \
    src/ui/widgets/MenuWidget.cpp \
    src/ui/widgets/OrderListWidget.cpp \
    src/utils/Logger.cpp

# Ресурсні файли
RESOURCES += \
    src/resources/app_resources.qrc

# UI файли (якщо використовуються)
# FORMS += \
#     src/ui/MainWindow.ui \
```

```
#     src/ui/dialogs/LoginDialog.ui

# Директорії для пошуку заголовкових файлів
INCLUDEPATH += \
    src \
    src/models \
    src/database \
    src/database/repositories \
    src/services \
    src/ui \
    src/ui/dialogs \
    src/ui/widgets \
    src/utils

# Платформо-специфічні налаштування
win32 {
    # Windows конфігурація
    CONFIG += windows
    RC_FILE = packaging/app.rc

    # PostgreSQL для Windows
    INCLUDEPATH += "C:/Program Files/PostgreSQL/16/include"
    LIBS += -L"C:/Program Files/PostgreSQL/16/lib" -lpq

    # Налаштування для MinGW
    mingw {
        QMAKE_CXXFLAGS += -Wall -Wextra
    }
}

# Конфігурація для різних типів збірки
CONFIG(debug, debug|release) {
    DEFINES += DEBUG_BUILD
    DEFINES += QT_QML_DEBUG
```

```
    DEFINES += QT_DEBUG
    TARGET = $${TARGET}_debug
}

CONFIG(release, debug|release) {
    # Release конфігурація
    DEFINES += RELEASE_BUILD
    DEFINES += QT_NO_DEBUG_OUTPUT
}
# Додаткові визначення препроцесора
DEFINES += \
    QT_DEPRECATED_WARNINGS \
    QT_DISABLE_DEPRECATED_BEFORE=0x060000

# Встановлення
isEmpty(PREFIX) {
    unix:!macx {
        PREFIX = /usr/local
    }
    win32 {
        PREFIX = "C:/Program Files/PizzeriaWaiter"
    }
    macx {
        PREFIX = /Applications
    }
}

target.path = $$PREFIX/bin
INSTALLS += target
# Очищення
QMAKE_CLEAN += \
    $$TARGET \
    Makefile \
    object_script.*
```

```
# Повідомлення при конфігурації
message("Configuring PizzeriaWaiter v$$VERSION")
message("Target: $$TARGET")
message("Qt version: $$QT_VERSION")
message("Build type: $$CONFIG")

# Перевірка мінімальної версії Qt
lessThan(QT_MAJOR_VERSION, 6) {
    error("PizzeriaWaiter requires Qt 6.0 or higher. Current version:
$$QT_VERSION")
}

# Додавання custom targets
QMAKE_EXTRA_TARGETS += docs
docs.commands = echo "Generating documentation..."

QMAKE_EXTRA_TARGETS += clean-all
clean-all.commands = rm -rf build* && rm -rf *.pro.user*

# Share all project output files by directories
MOC_DIR = moc
RCC_DIR = rcc
UI_DIR = ui
unix:OBJECTS_DIR = unix
win32:OBJECTS_DIR = win32
macx:OBJECTS_DIR = mac

# If release-buid -> run windeploy applications, that will collect all the
dlls
CONFIG(release, debug|release) {
    win32:QMAKE_POST_LINK = $$(QTDIR)/bin/windeployqt $$OUT_PWD/release
}

RC_ICONS = $$PWD/app-icon.ico
```

Додаток Б. Лістинг файлу «main.cpp»

```
#include <QApplication>
#include <QMessageBox>
#include <QDir>
#include <QStandardPaths>
#include <QSplashScreen>
#include <QPixmap>
#include <QPainter>
#include <QTimer>
#include <AuthService.h>
#include "ui/MainWindow.h"
#include "database/DatabaseManager.h"
#include "utils/Logger.h"

void showSplashScreen(QApplication &app) {
    QPixmap pixmap(400, 200);
    pixmap.fill(Qt::white);

    QPainter painter(&pixmap);
    painter.setRenderHint(QPainter::Antialiasing);

    QLinearGradient gradient(0, 0, 400, 200);
    gradient.setColorAt(0, QColor("#e3f2fd"));
    gradient.setColorAt(1, QColor("#1976d2"));
    painter.fillRect(pixmap.rect(), gradient);

    painter.setFont(QFont("Arial", 36, QFont::Bold));
    painter.setPen(Qt::white);
    painter.drawText(pixmap.rect(), Qt::AlignCenter, "🍕");

    painter.setFont(QFont("Arial", 18, QFont::Bold));
    painter.drawText(QRect(0, 120, 400, 30), Qt::AlignCenter, "Pizzeria
Waiter");
```

```
    painter.setFont(QFont("Arial", 12));
    painter.drawText(QRect(0, 150, 400, 20), Qt::AlignCenter, "v1.0 з
авторизацією");

    painter.setFont(QFont("Arial", 10));
    painter.drawText(QRect(0, 175, 400, 15), Qt::AlignCenter,
"Завантаження...");

    QSplashScreen splash(pixmap);
    splash.show();
    app.processEvents();

    QTimer::singleShot(2000, &splash, &QSplashScreen::close);
    QTimer timer;
    timer.setSingleShot(true);
    QEventLoop loop;
    QObject::connect(&timer, &QTimer::timeout, &loop, &QEventLoop::quit);
    timer.start(2000);
    loop.exec();
}

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    app.setApplicationName("PizzeriaWaiter");
    app.setApplicationVersion("1.0.0");
    app.setOrganizationName("Pizzeria");
    app.setApplicationDisplayName("Pizzeria Waiter - Система замовлень");

    showSplashScreen(app);

#ifdef USE_SPDLOG
    QString                          logPath                          =
```

```
QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
    QDir().mkpath(logPath);
    Logger::initialize(logPath + "/pizzeria.log");
#else
    Logger::initialize(); // Use Qt logging
#endif

    Logger::info("=== Pizzeria Waiter Starting ===");
    Logger::info(QString("Version: %1").arg(app.applicationVersion()));
    Logger::info(QString("Qt Version: %1").arg(QT_VERSION_STR));

    DatabaseManager& db = DatabaseManager::instance();

    // SQLite
    QString host = "localhost";
    QString database = "pizzeria_waiter.db";
    QString username = "";
    QString password = "";

    // Postgresql
    // QString database = "pizzeria_waiter";
    // QString username = "pizzeria_user";
    // QString password = "pizzeria_pass";
    int port = 5432;

    QString envHost = qgetenv("DB_HOST");
    QString envDatabase = qgetenv("DB_NAME");
    QString envUsername = qgetenv("DB_USER");
    QString envPassword = qgetenv("DB_PASS");
    QString envPort = qgetenv("DB_PORT");

    if (!envHost.isEmpty()) host = envHost;
    if (!envDatabase.isEmpty()) database = envDatabase;
    if (!envUsername.isEmpty()) username = envUsername;
```

```
    if (!envPassword.isEmpty()) password = envPassword;
    // if (!envPort.isEmpty()) port = envPort.toInt();

    Logger::info(QString("Connecting to database: %1@%2:%3/%4").arg(username, host).arg(port).arg(database));

    // if (!db.initialize(host, database, username, password, port)) {

    if (!db.initialize("", database, "", "", 0)) {
        QString error = QString("Помилка підключення до бази даних:\n\n%1\n\n"
                                "Переконайтеся що:\n"
                                "• PostgreSQL запущений\n"
                                "• База даних '%2' існує\n"
                                "• Користувач '%3' має доступ\n"
                                "• Параметри підключення правильні\n\n"
                                "Для налаштування БД запустіть:\n"
                                "./scripts/setup_database.sh")
                            .arg(db.lastError(), database, username);

        QMessageBox::critical(nullptr, "Помилка бази даних", error);
        Logger::error(QString("Database connection failed: %1").arg(db.lastError()));
        return -1;
    }

    Logger::info("Database connected successfully");

    MainWindow window;

    app.setWindowIcon(QIcon(":/icons/app-icon.png"));

    window.show();
```

```
    Logger::info("Application started successfully");

    int result = app.exec();

    Logger::info("=== Pizzeria Waiter Shutting Down ===");
    return result;
}
```

Додаток В. Лістинг файлу «DatabaseManager.h»

```
#pragma once
#include <QObject>
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>

class DatabaseManager : public QObject {
    Q_OBJECT
public:
    static DatabaseManager& instance();

    bool initialize(const QString &host, const QString &database,
                    const QString &username, const QString &password,
                    int port = 5432);
    QSqlDatabase& database();
    bool isConnected() const;

    QString lastError() const;
    QSqlQuery createQuery();
signals:
    void connectionStatusChanged(bool connected);
    void errorOccurred(const QString &error);
private:
    explicit DatabaseManager(QObject *parent = nullptr);
    ~DatabaseManager();
    bool createTables();
    bool loadSchemaFromFile(const QString &filename);
    QSqlDatabase m_database;
    bool m_connected = false;
    QString m_lastError;
    Q_DISABLE_COPY(DatabaseManager)
};
```

Додаток Г. Лістинг файлу «DatabaseManager.cpp»

```
#include "DatabaseManager.h"
#include <QSqlQuery>
#include <QSqlError>
#include <QFile>
#include <QTextStream>
#include <QDebug>
#include <QStandardPaths>
#include <QDir>

DatabaseManager& DatabaseManager::instance() {
    static DatabaseManager instance;
    return instance;
}

DatabaseManager::DatabaseManager(QObject *parent)
    : QObject(parent)
{
    m_database = QSqlDatabase::addDatabase("QSQLITE");
}

DatabaseManager::~DatabaseManager() {
    if (m_database.isOpen()) {
        m_database.close();
    }
}

bool DatabaseManager::initialize(const QString &host, const QString
&database,
                                 const QString &username, const QString
&password, int port) {
    m_database.setHostName(host);
    m_database.setDatabaseName(database);
```

```
    m_database.setUserName(username);
    m_database.setPassword(password);
    m_database.setPort(port);

    if (!m_database.open()) {
        m_lastError = m_database.lastError().text();
        qDebug() << "Database connection failed:" << m_lastError;
        emit errorOccurred(m_lastError);
        return false;
    }

    m_connected = true;
    emit connectionStatusChanged(true);

    // Create tables if they don't exist
    if (!createTables()) {
        m_lastError = "Failed to create database tables";
        return false;
    }

    qDebug() << "Database connected successfully";
    return true;
}

QSqlDatabase& DatabaseManager::database() {
    return m_database;
}

bool DatabaseManager::isConnected() const {
    return m_connected && m_database.isOpen();
}

QString DatabaseManager::lastError() const {
    return m_lastError;
```

```
}

QSqlQuery DatabaseManager::createQuery() {
    return QSqlQuery(m_database);
}

bool DatabaseManager::createTables() {
    QString schemaPath;
    if (m_database.driverName() == "QSQLITE") {
        schemaPath = ":/database/schema_sqlite.sql";
    } else {
        schemaPath = ":/database/schema.sql";
    }
    return loadSchemaFromFile(schemaPath);
}

bool DatabaseManager::loadSchemaFromFile(const QString &filename) {
    QFile file(filename);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        m_lastError = QString("Cannot open schema file: %1").arg(filename);
        return false;
    }

    QTextStream in(&file);
    in.setEncoding(QStringConverter::Utf8);
    QString schema = in.readAll();

    // Split by semicolon and execute each statement
    QStringList statements = schema.split(';', Qt::SkipEmptyParts);

    for (const QString &statement : statements) {
        QString trimmed = statement.trimmed();
        if (trimmed.isEmpty() || trimmed.startsWith("--")) {
            continue;
```

```
        }

        QSqlQuery query(m_database);
        if (!query.exec(trimmed)) {
            m_lastError = QString("Failed to execute SQL: %1 - %2")
                          .arg(query.lastError().text())
                          .arg(trimmed);
            qDebug() << m_lastError;
            return false;
        }
    }

    return true;
}
```

Додаток Д. Лістинг файлу «MainWindow.h»

```
#pragma once
#include "UserActionLogger.h"
#include <QMainWindow>
#include <memory>

QT_BEGIN_NAMESPACE
class QTabWidget;
class QStatusBar;
class QMenuBar;
class QLabel;
class QAction;
QT_END_NAMESPACE

class MenuWidget;
class TableMapWidget;
class OrderListWidget;
class OrderService;
class AuthService;
class User;
class Order;
class MenuItem;

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

protected:
    void closeEvent(QCloseEvent *event) override;
```

```
private slots:
    // Authentication slots
    void onUserLoggedIn(std::shared_ptr<User> user);
    void onUserLoggedOut();
    void onLoginRequired();
    void onLogoutTriggered();
    void onChangePasswordTriggered();

    // User management slots
    void onManageUsersTriggered();
    void onViewActivityTriggered();

    // Existing slots
    void onTableSelected(int tableNumber);
    void onMenuItemSelected(std::shared_ptr<MenuItem> item, int sizeId, int
quantity);
    void onOrderSelected(int orderId);
    void onNewOrder();
    void onRefreshOrders();
    void showAbout();

private:
    void setupUI();
    void setupMenuBar();
    void setupStatusBar();
    void setupConnections();
    void setupAuthActions();

    void updateUIForUser();
    void enableUI(bool enabled);
    void updateUserInfo();
    void updateCurrentOrderDisplay();
    void addItemToCurrentOrder(std::shared_ptr<MenuItem> item, int sizeId,
int quantity);
```

```
    bool showLoginDialog();

    // UI Components
    QTabWidget *m_centralTabs = nullptr;
    MenuWidget *m_menuWidget = nullptr;
    TableMapWidget *m_tableMapWidget = nullptr;
    OrderListWidget *m_orderListWidget = nullptr;
    QLabel *m_currentOrderLabel = nullptr;
    QLabel *m_userInfoLabel = nullptr;

    // Menu actions
    QAction *m_loginAction = nullptr;
    QAction *m_logoutAction = nullptr;
    QAction *m_changePasswordAction = nullptr;
    QAction *m_manageUsersAction = nullptr;
    QAction *m_viewActivityAction = nullptr;

    // Services
    std::shared_ptr<OrderService> m_orderService;
    std::shared_ptr<AuthService> m_authService;
    std::shared_ptr<UserActionLogger> m_userActionLogger;

    // Current state
    int m_currentTableNumber = 0;
    std::shared_ptr<Order> m_currentOrder;
};
```

Додаток Е. Лістинг файлу «MainWindow.cpp»

```
#include "MainWindow.h"
#include "widgets/TableMapWidget.h"
#include "widgets/MenuWidget.h"
#include "widgets/OrderListWidget.h"
#include "dialogs/LoginDialog.h"
#include "dialogs/UserManagementDialog.h"
#include "dialogs/UserActivityDialog.h"
#include "../services/OrderService.h"
#include "../services/AuthService.h"
#include "../services/UserActionLogger.h"
#include "../models/Order.h"
#include "../models/MenuItem.h"
#include "../models/User.h"
#include "../utils/Logger.h"
#include <QTabWidget>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QMenuBar>
#include <QStatusBar>
#include <QLabel>
#include <QMessageBox>
#include <QApplication>
#include <QCloseEvent>
#include <QInputDialog>
#include <QTimer>

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    // Ініціалізація сервісів
    m_authService = std::make_shared<AuthService>(this);
    m_userActionLogger  =  std::make_shared<UserActionLogger>(m_authService,
this);
    m_orderService = std::make_shared<OrderService>(m_authService, this);
```

```
    setupUI();
    setupMenuBar();
    setupStatusBar();
    setupConnections();

    // Спочатку заблокувати UI
    enableUI(false);

    // Показати діалог входу через невеликий проміжок часу
    QTimer::singleShot(100, this, &MainWindow::onLoginRequired);

    Logger::info("MainWindow created with authentication system");
}

MainWindow::~MainWindow() {
    Logger::info("MainWindow destroyed");
}

void MainWindow::setupUI() {
    // Central widget with tabs
    m_centralTabs = new QTabWidget(this);
    setCentralWidget(m_centralTabs);

    // Table map widget
    m_tableMapWidget = new TableMapWidget(this);
    m_centralTabs->addTab(m_tableMapWidget, "🪑 Столи");

    // Menu widget
    m_menuWidget = new MenuWidget(this);
    m_centralTabs->addTab(m_menuWidget, "🍕 Меню");

    // Order list widget
    m_orderListWidget = new OrderListWidget(m_authService, this);
```

```
    m_centralTabs->addTab(m_orderListWidget, "📋 Замовлення");

    // Current order info in status bar
    m_currentOrderLabel = new QLabel("Замовлення не вибрано", this);
    m_currentOrderLabel->setStyleSheet("padding: 4px; background-color: #f0f0f0; "
                                       "border: 1px solid #ccc; border-radius: 4px;");

    // User info label
    m_userInfoLabel = new QLabel("Не авторизовано", this);
    m_userInfoLabel->setStyleSheet("padding: 4px; background-color: #e3f2fd; "
                                   "border: 1px solid #1976d2; border-radius: 4px; "
                                   "color: #1976d2; font-weight: bold;");

    setWindowTitle("Pizzeria Waiter v1.0");
    resize(1200, 800);

    // Apply modern styling
    setStyleSheet(
        "QMainWindow { background-color: #fafafa; }"
        "QTabWidget::pane { border: 1px solid #ccc; background-color: white; }"
        "QTabBar::tab { "
        "  background-color: #e0e0e0; "
        "  border: 1px solid #ccc; "
        "  padding: 8px 16px; "
        "  margin-right: 2px; "
        "}"
        "QTabBar::tab:selected { "
        "  background-color: white; "
        "  border-bottom: none; "
```

```
        "}"
        "QTabBar::tab:hover { background-color: #f0f0f0; }"
    );
}

void MainWindow::setupMenuBar() {
    // Меню "Файл"
    QMenu *fileMenu = menuBar()->addMenu("&Файл");

    m_loginAction = fileMenu->addAction("&Увійти");
    m_loginAction->setShortcut(QKeySequence("Ctrl+L"));
    m_loginAction->setIcon(QIcon(":/icons/login.png"));
    connect(m_loginAction,            &QAction::triggered,            this,
&MainWindow::onLoginRequired);

    m_logoutAction = fileMenu->addAction("&Вийти з системи");
    m_logoutAction->setShortcut(QKeySequence("Ctrl+Shift+L"));
    m_logoutAction->setIcon(QIcon(":/icons/logout.png"));
    connect(m_logoutAction,            &QAction::triggered,            this,
&MainWindow::onLogoutTriggered);

    fileMenu->addSeparator();

    QAction *newOrderAction = fileMenu->addAction("&Нове замовлення");
    newOrderAction->setShortcut(QKeySequence::New);
    newOrderAction->setIcon(QIcon(":/icons/add.png"));
    connect(newOrderAction,            &QAction::triggered,            this,
&MainWindow::onNewOrder);

    fileMenu->addSeparator();

    QAction *exitAction = fileMenu->addAction("&Вихід");
    exitAction->setShortcut(QKeySequence::Quit);
    exitAction->setIcon(QIcon(":/icons/exit.png"));
```

```
    connect(exitAction, &QAction::triggered, this, &MainWindow::close);

    // Меню "Замовлення"
    QMenu *orderMenu = menuBar()->addMenu("&Замовлення");

    QAction *refreshAction = orderMenu->addAction("&Оновити");
    refreshAction->setShortcut(QKeySequence::Refresh);
    refreshAction->setIcon(QIcon(":/icons/refresh.png"));
    connect(refreshAction,            &QAction::triggered,            this,
&MainWindow::onRefreshOrders);

    // Меню "Користувачі"
    QMenu *userMenu = menuBar()->addMenu("&Користувачі");

    m_changePasswordAction = userMenu->addAction("&Змінити пароль");
    m_changePasswordAction->setIcon(QIcon(":/icons/key.png"));
    connect(m_changePasswordAction,       &QAction::triggered,       this,
&MainWindow::onChangePasswordTriggered);

    userMenu->addSeparator();

    m_manageUsersAction = userMenu->addAction("&Управління користувачами");
    m_manageUsersAction->setIcon(QIcon(":/icons/users.png"));
    connect(m_manageUsersAction,         &QAction::triggered,         this,
&MainWindow::onManageUsersTriggered);

    m_viewActivityAction = userMenu->addAction("&Журнал дій");
    m_viewActivityAction->setIcon(QIcon(":/icons/log.png"));
    connect(m_viewActivityAction,        &QAction::triggered,        this,
&MainWindow::onViewActivityTriggered);

    // Меню "Довідка"
    QMenu *helpMenu = menuBar()->addMenu("&Довідка");
```

```
    QAction *aboutAction = helpMenu->addAction("&Про програму");
    aboutAction->setIcon(QIcon(":/icons/info.png"));
    connect(aboutAction,             &QAction::triggered,             this,
&MainWindow::showAbout);

    setupAuthActions();
}


void MainWindow::setupStatusBar() {
    statusBar()->addWidget(m_userInfoLabel);
    statusBar()->addPermanentWidget(m_currentOrderLabel);
}

void MainWindow::setupConnections() {
    // Підключення сигналів авторизації
    connect(m_authService.get(), &AuthService::userLoggedIn,
            this, &MainWindow::onUserLoggedIn);
    connect(m_authService.get(), &AuthService::userLoggedOut,
            this, &MainWindow::onUserLoggedOut);

    // Підключення сигналів логування
    connect(m_userActionLogger.get(), &UserActionLogger::actionLogged,
            this, [this](UserAction action, const QString &details) {
                Q_UNUSED(action)
                statusBar()->showMessage(details, 2000);
            });

    // Інші підключення...
    connect(m_tableMapWidget, &TableMapWidget::tableSelected,
            this, &MainWindow::onTableSelected);
    connect(m_menuWidget, &MenuWidget::itemSelected,
            this, &MainWindow::onMenuItemSelected);
    connect(m_orderListWidget, &OrderListWidget::orderSelected,
```

```
            this, &MainWindow::onOrderSelected);

    // Автоматичне оновлення інформації користувача
    QTimer *updateTimer = new QTimer(this);
    connect(updateTimer,              &QTimer::timeout,              this,
&MainWindow::updateUserInfo);
    updateTimer->start(30000); // Кожні 30 секунд
}

void MainWindow::setupAuthActions() {
    updateUIForUser();
}

void MainWindow::onUserLoggedIn(std::shared_ptr<User> user) {
    // Логування входу
    m_userActionLogger->logLogin(user->username());

    // Увімкнення UI
    enableUI(true);
    updateUIForUser();
    updateUserInfo();

    // Привітальне повідомлення
    QString welcomeMessage = QString("Вітаємо, %1! Роль: %2")
            .arg(user->fullName(), user->roleString());
    statusBar()->showMessage(welcomeMessage, 5000);

    Logger::info(QString("User interface enabled for: %1 (%2)")
                         .arg(user->username(), user->roleString()));
}

void MainWindow::onUserLoggedOut() {
    // Логування виходу
    if (m_authService->currentUser()) {
```

```
        m_userActionLogger->logLogout(m_authService->currentUser()-
>username());
    }

    // Вимкнення UI
    enableUI(false);
    updateUIForUser();
    updateUserInfo();

    // Скидання поточного стану
    m_currentOrder.reset();
    m_currentTableNumber = 0;
    updateCurrentOrderDisplay();

    statusBar()->showMessage("Сесію завершено", 2000);
}

void MainWindow::onLoginRequired() {
    if (!showLoginDialog()) {
        // User cancelled login, close application
        QApplication::quit();
    }
}

void MainWindow::onLogoutTriggered() {
    int ret = QMessageBox::question(this, "Підтвердження",
                                    "Дійсно вийти з системи?",
                                    QMessageBox::Yes | QMessageBox::No);

    if (ret == QMessageBox::Yes) {
        m_authService->logout();

        // Show login dialog again
        QTimer::singleShot(500, this, &MainWindow::onLoginRequired);
```

```
    }
}

void MainWindow::onChangePasswordTriggered() {
    bool ok;
    QString oldPassword = QInputDialog::getText(this, "Зміна пароля",
                                                "Поточний пароль:",
QLineEdit::Password,
                                                "", &ok);
    if (!ok || oldPassword.isEmpty()) return;

    QString newPassword = QInputDialog::getText(this, "Зміна пароля",
                                                "Новий пароль:",
QLineEdit::Password,
                                                "", &ok);
    if (!ok || newPassword.isEmpty()) return;

    QString confirmPassword = QInputDialog::getText(this, "Зміна пароля",
                                                    "Підтвердити новий
пароль:", QLineEdit::Password,
                                                    "", &ok);
    if (!ok || confirmPassword.isEmpty()) return;

    if (newPassword != confirmPassword) {
        QMessageBox::warning(this, "Помилка", "Паролі не співпадають");
        return;
    }

    if (newPassword.length() < 4) {
        QMessageBox::warning(this, "Помилка", "Пароль повинен містити
мінімум 4 символи");
        return;
    }
```

```
    if (m_authService->changePassword(oldPassword, newPassword)) {
        QMessageBox::information(this, "Успіх", "Пароль успішно змінено");
    } else {
        QMessageBox::warning(this, "Помилка", "Не вдалося змінити пароль.
Перевірте поточний пароль.");
    }
}

void MainWindow::onManageUsersTriggered() {
    if (!m_authService->canManageUsers()) {
        m_userActionLogger->logAccessDenied("user_management",
"Insufficient permissions");
        QMessageBox::warning(this, "Доступ заборонено",
                             "У вас немає прав для управління
користувачами");
        return;
    }

    UserManagementDialog dialog(m_authService, this);
    dialog.exec();
}

void MainWindow::onViewActivityTriggered() {
    if (!m_authService->canManageUsers()) {
        m_userActionLogger->logAccessDenied("activity_log",   "Insufficient
permissions");
        QMessageBox::warning(this, "Доступ заборонено",
                             "У вас немає прав для перегляду журналу дій");
        return;
    }

    UserActivityDialog dialog(m_authService, m_userActionLogger, this);
    dialog.exec();
}
```

```
bool MainWindow::showLoginDialog() {
    LoginDialog dialog(m_authService, this);
    return dialog.exec() == QDialog::Accepted;
}

void MainWindow::enableUI(bool enabled) {
    m_centralTabs->setEnabled(enabled);

    // Enable/disable menu actions based on authentication
    m_loginAction->setVisible(!enabled);
    m_logoutAction->setVisible(enabled);
    m_changePasswordAction->setEnabled(enabled);
}

void MainWindow::updateUIForUser() {
    if (!m_authService->isLoggedIn()) {
        m_manageUsersAction->setVisible(false);
        return;
    }

    auto user = m_authService->currentUser();
    if (!user) return;

    // Show/hide actions based on user permissions
    m_manageUsersAction->setVisible(user->canManageUsers());
    m_manageUsersAction->setEnabled(user->canManageUsers());
}

void MainWindow::updateUserInfo() {
    if (!m_authService->isLoggedIn()) {
        m_userInfoLabel->setText("Не авторизовано");
        m_userInfoLabel->setStyleSheet("padding:   4px;   background-color:
#ffebee; "
```

```
                                        "border: 1px solid #f44336; border-
radius: 4px; "
                                        "color: #f44336; font-weight:
bold;");
        return;
    }

    QString sessionInfo = m_authService->sessionInfo();
    m_userInfoLabel->setText(sessionInfo);
    m_userInfoLabel->setStyleSheet("padding: 4px; background-color:
#e3f2fd; "
                                   "border: 1px solid #1976d2; border-
radius: 4px; "
                                   "color: #1976d2; font-weight: bold;");
}

void MainWindow::onTableSelected(int tableNumber) {
    if (!m_authService->isLoggedIn()) {
        m_userActionLogger->logAccessDenied("table_selection", "User not
logged in");
        return;
    }

    m_currentTableNumber = tableNumber;

    // Шукаємо існуюче замовлення в статусі "чернетка"
    auto orders = m_orderService->getOrdersByTable(tableNumber);
    m_currentOrder = nullptr;

    for (auto &order : orders) {
        if (order->status() == OrderStatus::Draft) {
            m_currentOrder = order;
            break;
        }
```

```
    }

    // Якщо немає чернетки, створюємо нове замовлення
    if (!m_currentOrder) {
        m_currentOrder = m_orderService->createOrder(tableNumber);
        if (m_currentOrder) {
            m_userActionLogger->logOrderCreated(m_currentOrder->id(),
tableNumber);
        }
    }

    updateCurrentOrderDisplay();

    // Перемикаємось на меню для додавання позицій
    m_centralTabs->setCurrentWidget(m_menuWidget);

    Logger::info(QString("Table %1 selected by %2, order ID: %3")
                         .arg(tableNumber)
                         .arg(m_authService->currentUsername())
                         .arg(m_currentOrder ? m_currentOrder->id() : 0));
}

void  MainWindow::onMenuItemSelected(std::shared_ptr<MenuItem>  item,  int
sizeId, int quantity) {
    if (!m_authService->isLoggedIn()) return;

    if (!m_currentOrder) {
        QMessageBox::information(this, "Увага", "Спочатку виберіть стіл");
        return;
    }

    addItemToCurrentOrder(item, sizeId, quantity);
}
```

```
void MainWindow::addItemToCurrentOrder(std::shared_ptr<MenuItem> item, int sizeId, int quantity) {
    if (!m_currentOrder || !item) return;

    bool success = m_orderService->addItemToOrder(m_currentOrder->id(), item, quantity, sizeId);

    if (success) {
        // Перезавантажуємо замовлення для отримання оновлених даних
        m_currentOrder = m_orderService->getOrder(m_currentOrder->id());
        updateCurrentOrderDisplay();

        // Логування додавання позиції
        QString details = QString("Added %1x %2 to order %3")
                .arg(quantity)
                .arg(item->name())
                .arg(m_currentOrder->id());
        m_userActionLogger->logOrderUpdated(m_currentOrder->id(), details);

        statusBar()->showMessage(QString("Додано: %1").arg(item->name()), 3000);
    } else {
        m_userActionLogger->logError(QString("Failed to add item %1 to order").arg(item->name()));
        QMessageBox::warning(this, "Помилка", "Не вдалося додати позицію до замовлення");
    }
}

void MainWindow::updateCurrentOrderDisplay() {
    if (!m_currentOrder) {
        m_currentOrderLabel->setText("Замовлення не вибрано");
        return;
    }
```

```
    QString text = QString("Стіл %1 | Позицій: %2 | Сума: ₴%.2f")
                   .arg(m_currentOrder->tableId())
                   .arg(m_currentOrder->itemCount())
                   .arg(m_currentOrder->totalAmount());

    m_currentOrderLabel->setText(text);
}

void MainWindow::onOrderSelected(int orderId) {
    auto order = m_orderService->getOrder(orderId);
    if (order) {
        QString summary = m_orderService->getOrderSummary(order);
        QMessageBox::information(this, "Деталі замовлення", summary);
    }
}

void MainWindow::onNewOrder() {
    if (!m_authService->isLoggedIn()) return;

    if (m_currentTableNumber == 0) {
        QMessageBox::information(this, "Увага", "Спочатку виберіть стіл");
        return;
    }

    // Create new order
    m_currentOrder = m_orderService->createOrder(m_currentTableNumber);
    updateCurrentOrderDisplay();

    m_centralTabs->setCurrentWidget(m_menuWidget);
}

void MainWindow::onRefreshOrders() {
    if (!m_authService->isLoggedIn()) return;
```

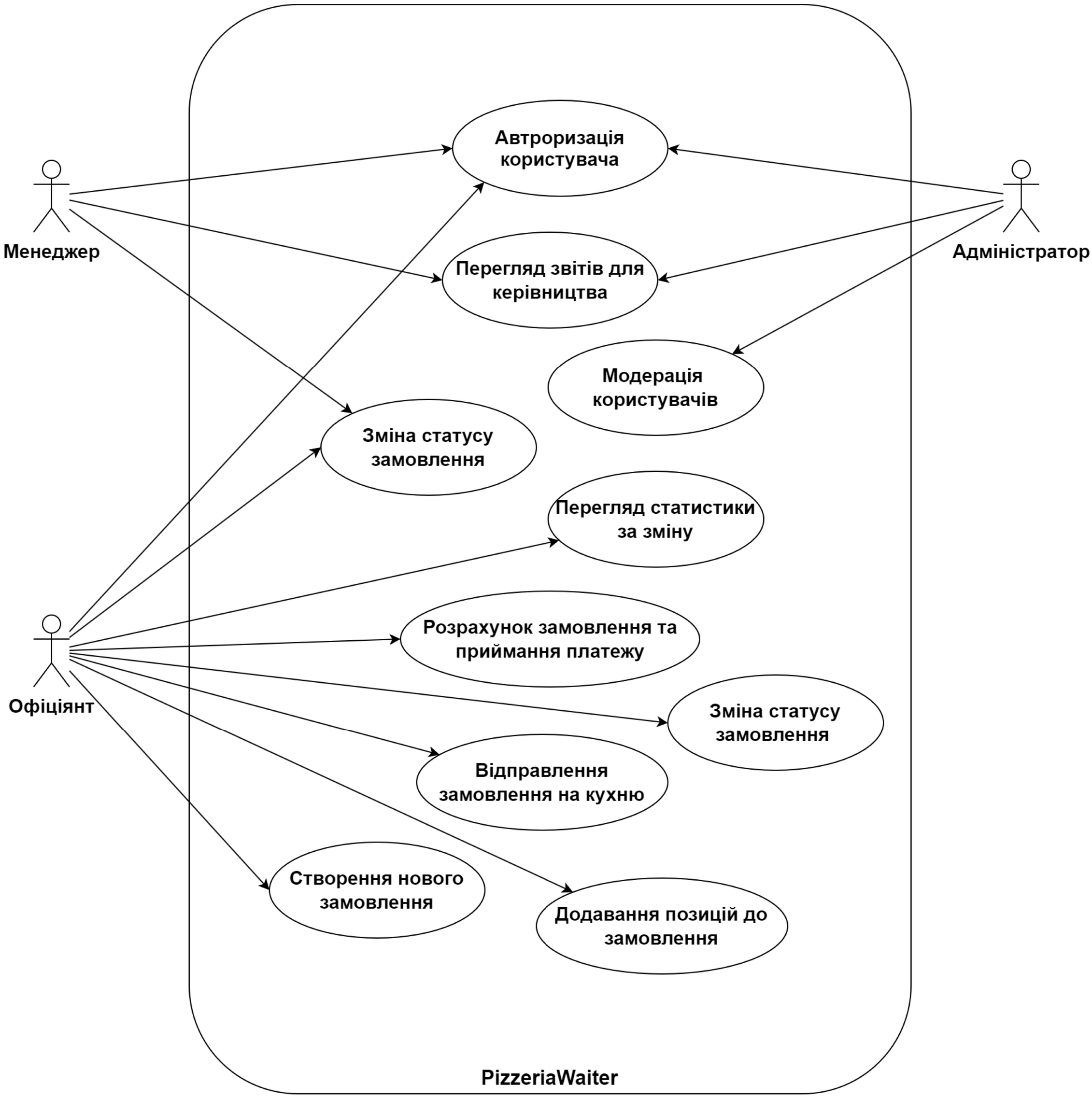
```
    m_orderListWidget->refreshOrders();
    statusBar()->showMessage("Замовлення оновлено", 2000);
}

void MainWindow::showAbout() {
    QMessageBox::about(this, "Про програму",
                       "<h3>Pizzeria Waiter v1.0</h3>"
                       "<p>Система управління замовленнями піцерії з
авторизацією</p>"
                       "<p>Розроблено на Qt6 + PostgreSQL</p>"
                       "<p><b>Основні функції:</b></p>"
                       "<ul>"
                       "<li>Система користувачів та ролей</li>"
                       "<li>Управління столами</li>"
                       "<li>Каталог меню з різними розмірами піц</li>"
                       "<li>Створення та відстеження замовлень</li>"
                       "<li>Система статусів замовлень</li>"
                       "<li>Друк чеків</li>"
                       "</ul>");
}

void MainWindow::closeEvent(QCloseEvent *event) {
    if (m_authService->isLoggedIn()) {
        int ret = QMessageBox::question(this, "Підтвердження",
                                        "Дійсно закрити програму?",
                                        QMessageBox::Yes |
QMessageBox::No);

        if (ret == QMessageBox::Yes) {
            // Логування закриття програми
            m_userActionLogger->logAction(UserAction::Logout,  "Application
closing");
            m_authService->logout();
```

```
            Logger::info("Application closing by user request");
            event->accept();
        } else {
            event->ignore();
        }
    } else {
        Logger::info("Application closing (no active session)");
        event->accept();
    }
}
```



Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Вальчишин О.І.		
Перев.		Пиж В.С.		
Т.контр.				
Рецензент				
Н.контр.		Приймак В.А.		
Затв.				

2025.КРБ.123.602.04.00.00 ДВ

Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»

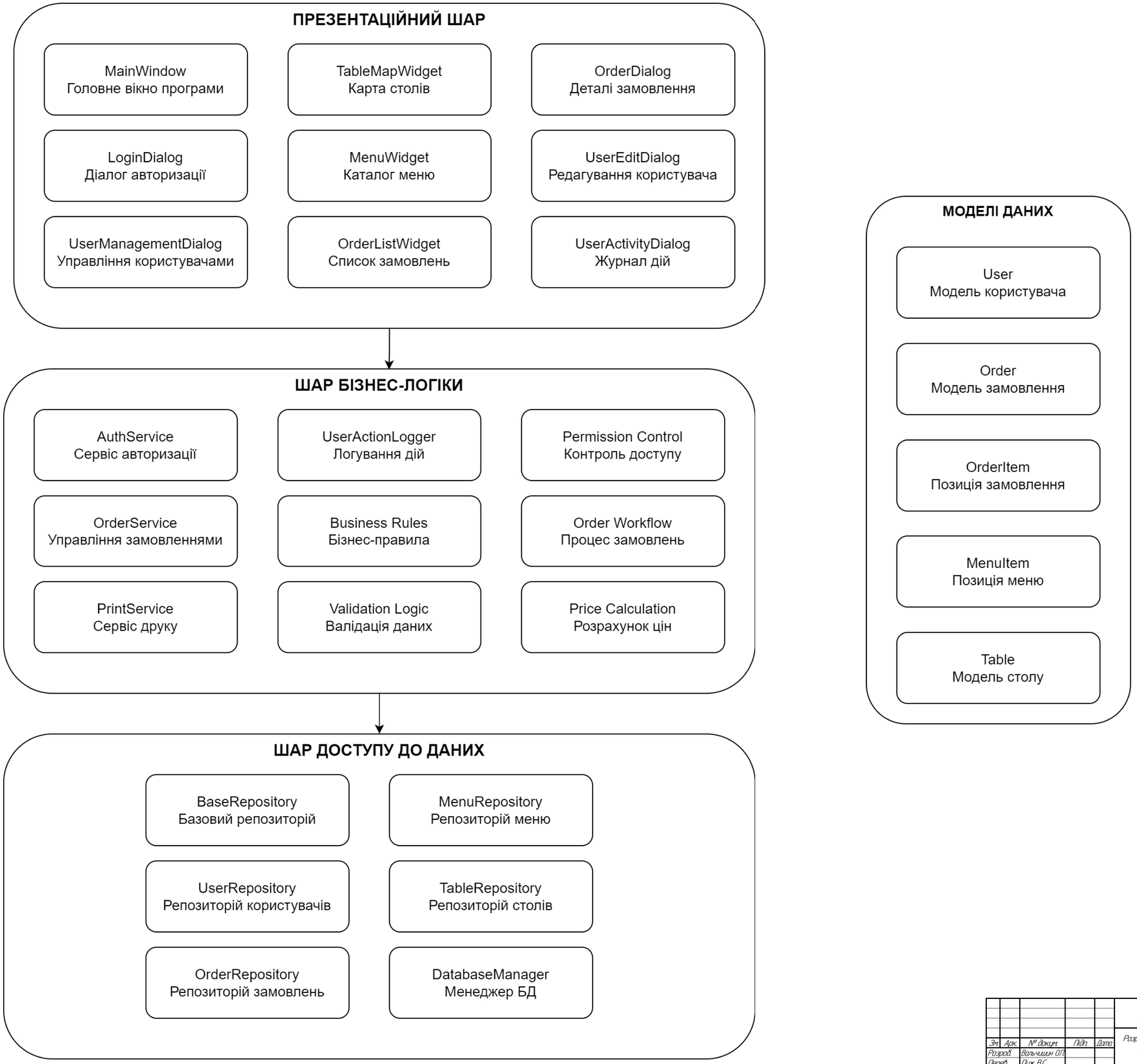
UML-діаграма варіантів використання

Літ.	Маса	Масштаб
Аркуш	Аркушів 1	

ВСП ТФК ТНТУ КІБ-602 м. Тернопіль

Копіював

Формат А1



Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Вольчишин О.П.		
Перев.		Пиж В.С.		
Т.контр.				
Рецензент				
Н.контр.		Пришмак В.А.		
Затв.				

2025.КРБ.123.602.04.00.00 СС

Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»

Схема структурна програмного забезпечення

Літ.	Маса	Масштаб
Аркуш	Аркушів 1	

ВСП ТФК ТНТУ КІБ-602 м. Тернопіль

Копіював

Формат А1

Перш. застос.

Справ. №

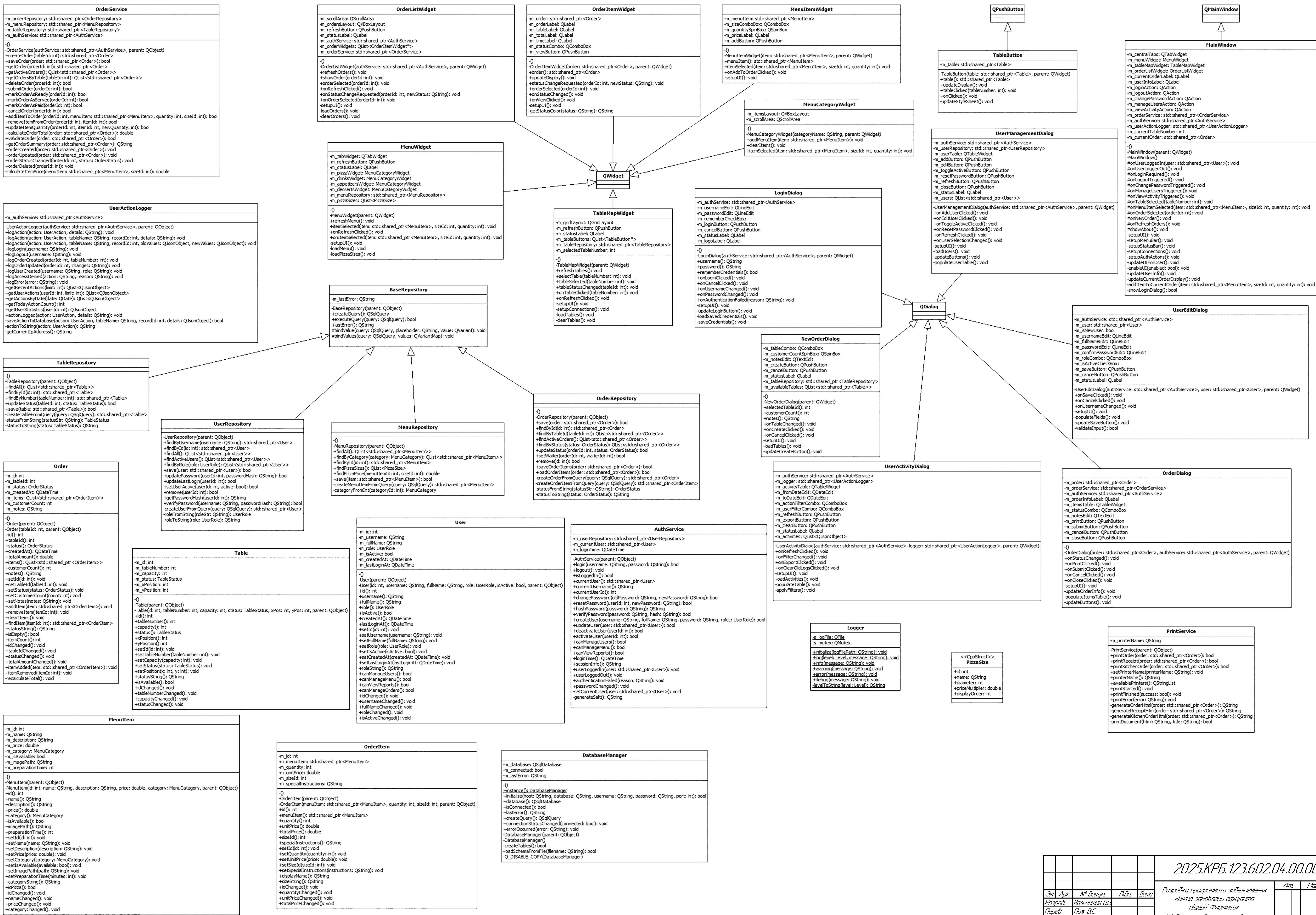
Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № оригіл.



Перв. застос.

Справ. №

Підп. і дата

Інв. № дубл.

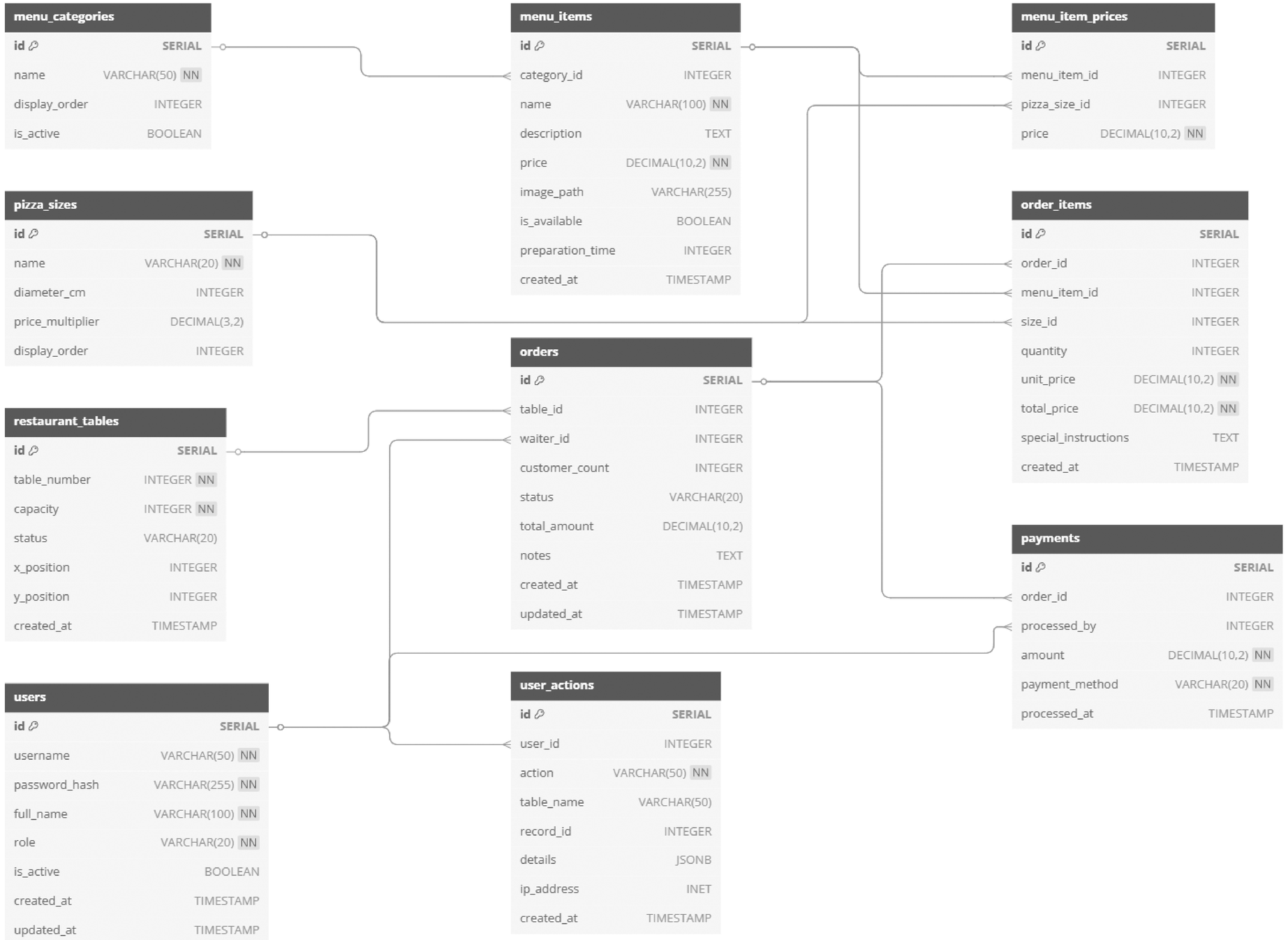
Взам. інв. №

Підп. і дата

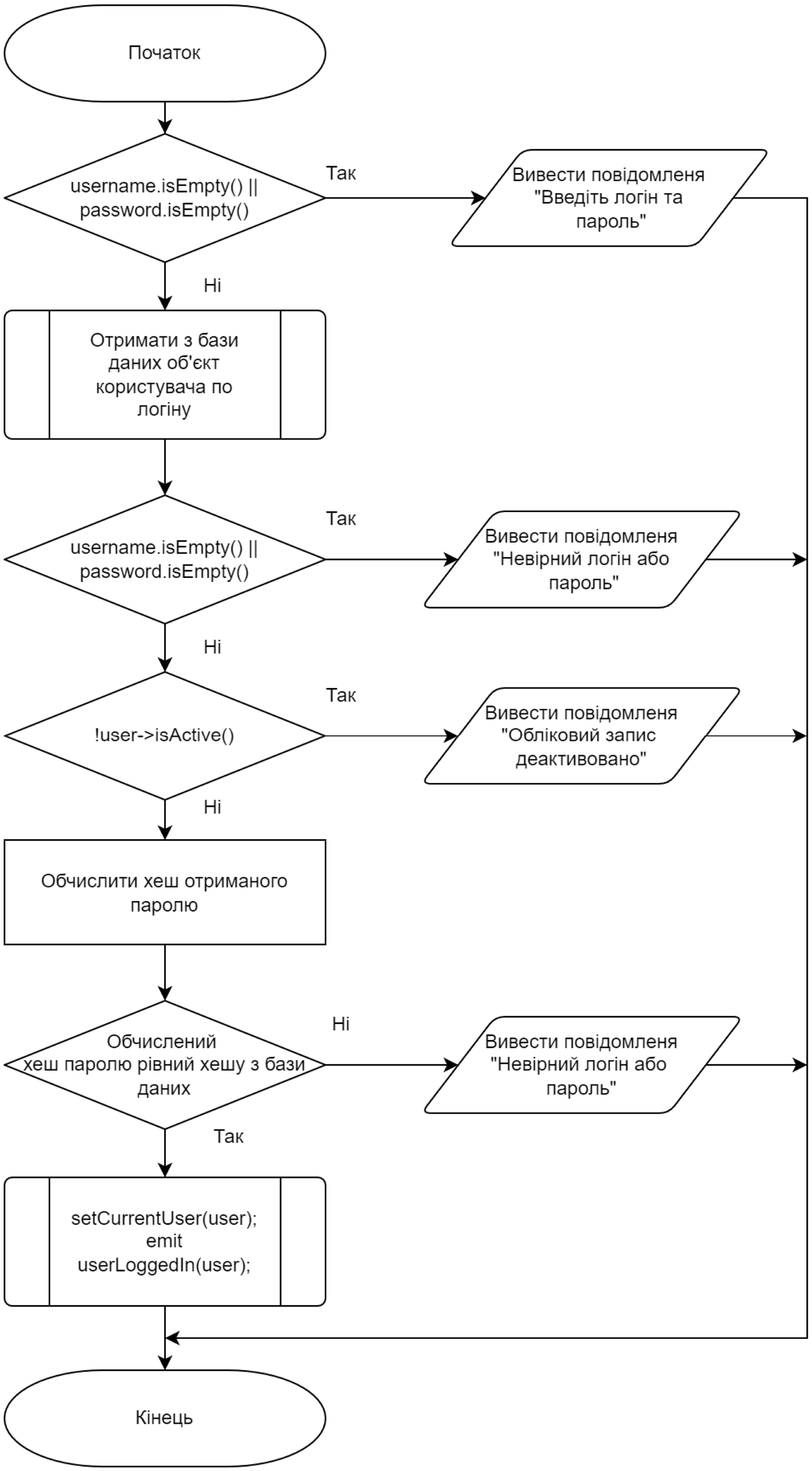
Інв. № ориг.

					2025.КРБ.123.602.04.00.00 ДК			
Зм.	Арк.	№ докум.	Підп.	Дата				
Розроб.	Вольчишин О.П.				Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»	Літ.	Маса	Масштаб
Перев.	Пиж В.С.							
Т.контр.					UML-діаграма класів програмного забезпечення	Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КІб-602		
Н.контр.	Приймак В.А.							
Затв.						м. Тернопіль		

Копіював   Формат A1


menu_categories
id SERIAL
name VARCHAR(50) NN
display_order INTEGER
is_active BOOLEAN
menu_items
id SERIAL
category_id INTEGER
name VARCHAR(100) NN
description TEXT
price DECIMAL(10,2) NN
image_path VARCHAR(255)
is_available BOOLEAN
preparation_time INTEGER
created_at TIMESTAMP
menu_item_prices
id SERIAL
menu_item_id INTEGER
pizza_size_id INTEGER
price DECIMAL(10,2) NN
pizza_sizes
id SERIAL
name VARCHAR(20) NN
diameter_cm INTEGER
price_multiplier DECIMAL(3,2)
display_order INTEGER
order_items
id SERIAL
order_id INTEGER
menu_item_id INTEGER
size_id INTEGER
quantity INTEGER
unit_price DECIMAL(10,2) NN
total_price DECIMAL(10,2) NN
special_instructions TEXT
created_at TIMESTAMP
orders
id SERIAL
table_id INTEGER
waiter_id INTEGER
customer_count INTEGER
status VARCHAR(20)
total_amount DECIMAL(10,2)
notes TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
restaurant_tables
id SERIAL
table_number INTEGER NN
capacity INTEGER NN
status VARCHAR(20)
x_position INTEGER
y_position INTEGER
created_at TIMESTAMP
payments
id SERIAL
order_id INTEGER
processed_by INTEGER
amount DECIMAL(10,2) NN
payment_method VARCHAR(20) NN
processed_at TIMESTAMP
users
id SERIAL
username VARCHAR(50) NN
password_hash VARCHAR(255) NN
full_name VARCHAR(100) NN
role VARCHAR(20) NN
is_active BOOLEAN
created_at TIMESTAMP
updated_at TIMESTAMP
last_login_at TIMESTAMP
user_actions
id SERIAL
user_id INTEGER
action VARCHAR(50) NN
table_name VARCHAR(50)
record_id INTEGER
details JSONB
ip_address INET
created_at TIMESTAMP

Блок-схема алгоритму роботи функції AuthService::login(QString &username, QString &password)



Зм.	Арк.	№ докум.	Підп.	Дата	2025.КРБ.123.602.04.00.00 БС
Розроб.		Вальчишин О.П.			Розробка програмного забезпечення «Вікно замовлень офіціанта піцерії Фламінго»
Перев.		Пиж В.С.			Блок-схема алгоритму роботи функції AuthService::login
Т.контр.					Літ. / Маса / Масштаб
Рецензент					Аркуш / Аркушів 1
Н.контр.		Приймак В.А.			ВСП ТФК ТНТУ КІБ-602 м. Тернопіль
Затв.					

Перш. застос. | Справ. № | Підп. і дата | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № оригі.

Копіював | Формат А1

Таблиця техніко-економічних показників

№ п/п	Показник	Одиниці вимірювання	Значення
1	Мова програмування	–	C++
2	Фреймворк	–	Qt v6
3	База даних	–	PostgreSQL
4	Середовище розробки	–	Qt Creator
5	Захист	–	Засобами ОС
6	Собівартість	грн.	58732,29
7	Плановий прибуток	грн.	32890,08
8	Ціна	грн.	91622,37
9	Економічна ефективність	грн.	0,56
10	Термін окупності	рік	1,8