**Форма № Н-6.01-у**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

**КУРСОВИЙ ПРОЕКТ**

**(РОБОТА)**

з дисципліни «Організація баз даних»

(назва дисципліни)

на тему: «Розробка інформаційної системи веломайстерні»

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студента | | 2 | | курсу групи | | | | КН-23-1 | |
| напряму підготовки | | | | | | 122 «Комп’ютерні | | | |
| науки» | | | | | | | | | |
|  | | | | | | | | | |
| Полинька І.М. | | | | | | | | | |
| (прізвище та ініціали) | | | | | | | | | |
| Керівник | ст. викл. Сохін Н.Л. | | | | | | | | |
| (посада, вчене звання, | | | | | | | | | |
|  | | | | | | | | | |
| науковий ступінь, прізвище та ініціали) | | | | | | | | | |
|  | | | | | | | | | |
| Національна шкала | | | | |  | | | | |
| Кількість балів: | | |  | | | | Оцінка: ECTS | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  |  |  |  |
|  | (підпис) |  | (прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (прізвище та ініціали) |

м. Кременчук – 2025 р.

|  |
| --- |
| ПОЯСНЮВАЛЬНА ЗАПИСКА |

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

CRUD операції – Create, Read, Update, Delete операції

ПІБ – Прізвище, Ім’я, По-батькові

EF Core – Entity Framework Core

PK – Primary key (Первинний ключ)

FK – Foreign key (Зовнішний ключ)

ЗМІСТ

[ВСТУП 5](#_Toc199448680)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 6](#_Toc199448681)

[1.1 Загальна характеристика предметної області 6](#_Toc199448682)

[1.2 Обгрунтування вибору засобів реалізації 7](#_Toc199448683)

[2 ПОСТАНОВКА ЗАДАЧІ 9](#_Toc199448684)

[2.1 Загальні вимоги 9](#_Toc199448685)

[2.2 Вихідна інформація 10](#_Toc199448686)

[2.3 Вхідна інформація 12](#_Toc199448687)

[3 РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ 15](#_Toc199448688)

[3.1 Побудова інформаційно-логічної моделі даних 15](#_Toc199448689)

[3.2 Визначення функціональних залежностей між реквізитами 16](#_Toc199448690)

[3.3 Фізична модель бази даних та склад таблиць 22](#_Toc199448691)

[3.4 Створення ключів та опис зв’язків 25](#_Toc199448692)

[3.5 Підтримка цілісності даних 27](#_Toc199448693)

[3.6 Забезпечення безпеки даних 28](#_Toc199448694)

[3.7 Типові відображення 29](#_Toc199448695)

[4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 34](#_Toc199448696)

[4.1 Функціональне призначення застосунку 34](#_Toc199448697)

[4.2 Опис програмних модулів 35](#_Toc199448698)

[4.3 Структура інтерфейсу користувача 40](#_Toc199448699)

[4.4 Інструкція користувача 45](#_Toc199448700)

[ВИСНОВКИ 47](#_Toc199448701)

[Додаток А Скрипти створення бази даних та таблиць 49](#_Toc199448702)

[Додаток Б Лістинг програмних файлів 52](#_Toc199448703)

[Додаток В Скрипти створення тригерів 63](#_Toc199448704)

**ВСТУП**

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами. На сьогодні використання баз даних актуальне, тому що дуже оптимізує роботу великих компаній. Вони допомагають систематизувати будь-яку потрібну інформацію та легко її знайти. На даний час є велика кількість інструментів та технологій для створення БД. Вони мають свої особливості та створені для полегшення та прискорення роботи з даними.

Бази даних стали невід'ємною частиною сучасного бізнесу і наукових досліджень. Вони використовуються для зберігання великих обсягів даних, таких, наприклад, як клієнтські записи, фінансова інформація, статистичні дані тощо. Це дозволяє компаніям здійснювати аналіз, прогнозування та приймати обґрунтовані рішення на підставі наявних даних.

Предметна область даної курсової роботи: «Веломайстерня». Це програмне забезпечення для майстерні, в якій співробітники будуть оперувати даними та модифікувати БД за допомогою інтерфейсу, підключенного до сховища.

Основними завданнями є розробка структурованої бази даних, забезпечення можливості зручного та безпечного виконання CRUD операцій для кожної таблиці БД, а також впровадження функціоналу, що дозволяє проводити запити до бази даних. Для взаємодії з базою буде використовуватися ADO.NET, що гарантує швидку та безпечну роботу з даними.

Застосування Windows Forms дозволить створити зручний графічний інтерфейс, що сприяє можливості роботи з БД для кожного працівника, а також реалізувати ефективне управління даними та отримання розширенної інформації через підготовлені запити.

Розроблена система буде актуальним рішенням для автоматизації роботи веломайстерні. Впровадження таких інформаційних систем спрощує доступ до читання, додавання та редагування необхідної інформації для кожного працівника.

**1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

**1.1 Загальна характеристика предметної області**

Веломайстерня є сервісним закладом, основною діяльністю якого є надання послуг з ремонту велосипедів та продаж велосипедних запчастин. Для ефективного управління майстернею та якісного обслуговування клієнтів необхідна комп’ютеризована інформаційна система, яка охоплює всі ключові процеси: від обліку клієнтів і працівників до оформлення ремонтів та продажу деталей.

Комп'ютеризація допомагає забезпечити точний облік, швидке обслуговування та зручне управління ресурсами.

У веломайстерні працюють менеджери та майстри, які мають різні ролі. Усі користувачі можуть додавати, оновлювати та видаляти дані про клієнтів.

Менеджери можуть формувати замовлення товару на продаж, вказуючи клієнта, працівника, що проводив обслуговування клієнта та дату створення замовлення. Після додавання товарів, що були замовлені, з урахуванням кількості працівник отримає суму до сплати для клієнта, а після отримання грошей – підтвердити оплату замовлення.

Майстри можуть формувати записи на обслуговування велосипеду, вказуючи клієнта, працівника, що буде проводити обслуговування, опис проблем та суму сплати за роботу, що можна додати згодом. Після проведення обслуговування та отримання коштів за працю – підтвердити оплату замовлення. вказуючи ПІБ клієнта, тип велосипеда, опис несправності та орієнтовну дату виконання.

Користувачі мають додатковий функціонал для формування звіту та детального відображення даних.

Це дозволяє вести облік роботи веломайстерні, розраховувати прибуток, попит товарів та зберігати дані про роботу з клієнтами.

Основними об'єктами системи є:

* Клієнти – зберігається контактна інформація про клієнтів;
* Співробітники – особи, які працюють у веломайстерні, зберігаються контактні дані, посада та заробітня плата;
* Замовлення – процес, що здійснюють співробітники для обслуговування клієнта та отримання прибутку;
* Запис на ремонт – процес, що здійснюють співробітники для обслуговування велосипеду клієнта та отримання прибутку;
* Товар – облік деталей на продаж..

**1.2 Обгрунтування вибору засобів реалізації**

Для реалізації бази даних комп’ютеризованої системи управління веломайстернеі були обрані наступні засоби: мова програмування C#, Entity Framework Core, Windows Forms та MySQL Server . Нижче наведено обґрунтування цього вибору.

1. Мова програмування C#

C# – це сучасна мова програмування з повною підтримкою об’єктно-орієнтованого підходу, що ідеально підходить для розробки десктопних програм. Вона забезпечує зручну інтеграцію з інструментами Microsoft і дозволяє швидко створювати функціональні та надійні додатки.

**Висока швидкодія:** C# у поєднанні з .NET дає змогу створювати продуктивні програми з швидким доступом до бази даних та користувацького інтерфейсу.

**Надійність:** Система типів, управління пам’яттю та підтримка виключень дозволяє уникнути багатьох типових помилок.

**Актуальність:** Постійні оновлення й підтримка нових фіч дозволяють працювати з сучасними підходами до розробки ПЗ.

1. Entity Framework Core

**EF Core** – це ORM-бібліотека, яка дозволяє взаємодіяти з базою даних через об'єкти, а не через сирі SQL-запити. Це значно спрощує розробку та зменшує ризик помилок.

**Підтримка LINQ:** Дозволяє писати гнучкі та читабельні запити до бази даних.

**Міграції:** Дає змогу легко оновлювати структуру бази даних без втрати даних.

1. Windows Forms

**Windows Forms** – це класична технологія для побудови графічного інтерфейсу Windows-додатків, яка відзначається простотою у використанні та швидкістю розробки.

**Швидка розробка UI:** Інтуїтивно зрозумілий drag-and-drop редактор в Visual Studio дає змогу швидко створювати форми та елементи керування.

**Гнучкість:** Можна легко підключати обробку подій, налаштовувати візуальний стиль і реалізовувати динамічні сценарії взаємодії з користувачем.

**Сумісність:** Працює без проблем на всіх сучасних версіях Windows, що підходить для майстерень, де використовується звичайне офісне обладнання.

4. MySQL Server

**MySQL** – одна з найпопулярніших СУБД з відкритим вихідним кодом, яка була обрана як основа для зберігання даних ІС веломайстерні.

**Широка підтримка:** Працює на більшості платформ, легко інтегрується з C# та EF Core.

**Надійність та масштабованість:** Підходить як для невеликих майстерень, так і для розширення системи в майбутньому.

**Висока швидкість роботи:** Забезпечує швидкий доступ до даних, навіть за великого обсягу запитів та одночасного доступу кількох користувачів.

**2 ПОСТАНОВКА ЗАДАЧІ**

**2.1 Загальні вимоги**

Основні призначення проектованої системи полягають в розробці інформаційної системи для полегшення введення обліку та швидкого доступу до даних.

Функціональні вимоги:

1. Можливість авторизації для користувачів та реєстрації нових для взаємодії з інформацією.
2. Наявність кнопок для виконання CRUD операцій та додаткових дій, передбачених предметною областю.
3. Забезпечення перевірки коректності введення даних користувачами.
4. Виведення повідомлень про помилки у разі некоректного вводу даних.
5. Зручний інтерфейс, що спростить взаємодію з системою.
6. Виведення інформації за запитом користувача.

Нефункціональні вимоги:

1. Система має бути розроблена з використанням Windows Forms для зручного графічного інтерфейсу.
2. Використання MySQL Server для зберігання даних.
3. Взаємодія з базою даних має здійснюватися через SqlClient без використання сторонніх ORM.
4. Код має бути структурований з використанням окремих класів для кожної сутності та підкласів задля роботи запитів (Clients, Workers, тощо; clientsDAO, workersDAO, тощо; FormClients, FormWorkers, тощо).
5. Програма повинна коректно обробляти винятки під час роботи з базою даних та недопускати технічні вибоїни через які можна порушити роботу бази даних.

Інтерфейсні вимоги:

1. Усі меню повинні мати menu strip для зручної роботи з формами та можливістю відтворювати запити у тих місцях, де це передбачено.
2. Усі меню мають підтримувати світлу та темну теми відображення інтерфейсу, мати приємні кольори та шрифти для усіх елементів форм.
3. Головне меню має містити кнопки для переходу до меню таблиць.
4. У кожній вкладці має бути таблиця (DataGrid) для відображення відповідних даних, поля для тексту (TextBox) для роботи з таблицею та допоміжні написи (Label).
5. У кожній вкладці має бути кнопка для завантаження даних із бази та передбачене завантаження даних після кожної дії користувача.

**2.2 Вихідна інформація**

Перелік вихідних повідомлень системи наведено у таблиці 2.1.

Таблиця 2.1 – Вихідні повідомлення системи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування повідомлення | Форма подання повідомлення | Періодичність видачі | Отримувач інформації |
| 1 | Відомості про користувачів | Електронна таблиця | За запитом | Адміністратор |
| 2 | Відомості про клієнтів | Електронна таблиця | За запитом | Менеджер, Майстер |
| 3 | Відомості про робітників | Електронна таблиця | За запитом | Менеджер, Майстер |
| 4 | Список замовлень | Електронна таблиця | За запитом | Менеджер, Майстер |
| 5 | Список деталей замовлення | Електронна таблиця | За запитом | Менеджер, Майстер |
| 6 | Список записів на ремонт | Електронна таблиця | За запитом | Менеджер, Майстер |
| 7 | Відомості про товари | Електронна таблиця | За запитом | Менеджер, Майстер |

* + - 1. Відомості про користувачів:

Містять список усіх користувачів, що мають доступ до системи. Включає логіни, ролі, паролі та унікальні ідентифікатори. Використовується адміністратором для керування доступом до програми.

Отримувач: Адміністратор;

Форма: Електронна таблиця;

Періодичність: За запитом.

* + - 1. Відомості про клієнтів:

Містить контактні дані клієнтів, їх ПІБ та номери телефонів. Необхідна для зворотного зв’язку, оформлення замовлень та запису на ремонт.

Отримувач: Менеджер, Майстер;

Форма: Електронна таблиця;

Періодичність: За запитом.

* + - 1. Відомості про робітників:

Містить інформацію про персонал майстерні: ПІБ, посади, номери телефонів і розміри заробітної плати. Забезпечує можливість менеджменту працівників.

Отримувач: Менеджер, Майстер;

Форма: Електронна таблиця;

Періодичність: За запитом.

* + - 1. Список замовлень:

Відображає деталі оформлених замовлень: дати, суми оплати, залучених клієнтів і працівників. Дозволяє контролювати процес обслуговування.

Отримувач: Менеджер, Майстер;

Форма: Електронна таблиця;

Періодичність: За запитом.

* + - 1. Список деталей замовлення:

Містить перелік товарів у кожному замовленні з вказанням кількості. Необхідний для обліку витрачених матеріалів і контролю запасів.

Отримувач: Менеджер, Майстер;

Форма: Електронна таблиця;

Періодичність: За запитом.

1. Список записів на ремонт:

Містить дані про звернення клієнтів на ремонт, включаючи дату запису, причину звернення, залучених працівників та оплату. Дозволяє керувати чергою ремонту та навантаженням майстрів.

Отримувач: Менеджер, Майстер;

Форма: Електронна таблиця;

Періодичність: За запитом.

1. Відомості про товари:

Містить інформацію про деталі, доступні для продажу або використання в ремонтах, включаючи назву та ціну. Забезпечує облік товарів і контроль за складськими залишками.

Отримувач: Менеджер, Майстер;

Форма: Електронна таблиця;

Періодичність: За запитом.

**2.3 Вхідна інформація**

Відомості про користувачів:

Містить дані про всіх користувачів системи, що мають доступ до програми.

1. Номер користувача (user\_id) – унікальний ідентифікатор користувача.
2. Логін (login) – логін користувача для входу в систему.
3. Пароль (password) – пароль користувача.
4. Роль (role) – роль користувача в системі ("Administrator", "Manager", "Master").

Відомості про клієнтів:

Зберігає інформацію про клієнтів, які звертаються до веломайстерні.

1. Номер клієнта (client\_id) – унікальний ідентифікатор клієнта.
2. ПІБ (pib) – повне ім’я клієнта (прізвище, ім’я, по батькові).
3. Номер телефону (phone\_number) – контактний номер телефону клієнта.

Відомості про працівників:

Містить дані про працівників веломайстерні.

1. Номер робітника (worker\_id) – унікальний ідентифікатор робітника.
2. ПІБ (pib) – ПІБ робітника.
3. Посада (position) – посада ("майстер", "менеджер", "адміністратор").
4. Номер телефону (phone\_number) – номер телефону робітника.
5. Заробітня плата (salary) – місячна заробітна плата робітника.

Список замовлень:

Включає загальні відомості про замовлення клієнтів.

1. Номер замовлення (order\_id) – унікальний номер замовлення.
2. Номер клієнта (client\_id) – унікальний ідентифікатор клієнта.
3. Номер робітника (worker\_id) – унікальний ідентифікатор робітника.
4. Дата замовлення (order\_date) – дата оформлення замовлення.
5. Сума оплати (payment\_amount) – сума, яку потрібно сплатити за замовлення.
6. Дата оплати (payment\_date) – фактична дата оплати.

Список деталей замовлень:

Пов’язана з таблицею "Замовлення" та описує, які товари входять до складу кожного замовлення.

1. Номер замовлення (order\_id) – унікальний номер замовлення.
2. Номер товару (detail\_id) – унікальний ідентифікатор товару.
3. Кількість товару (quantity\_details) – кількість одиниць певного товару в конкретному замовленні.

Список записів на ремонт:

Відображає звернення клієнтів на обслуговування велосипедів.

1. Номер запису на ремонт (sign\_up\_repairs\_id) – унікальний номер запису на ремонт.
2. Номер робітника (worker\_id) – унікальний ідентифікатор робітника.
3. Номер клієнта (client\_id) – унікальний ідентифікатор клієнта.
4. Причина ремонту (repair\_reason) – опис причини звернення.
5. Дата запису (sign\_up\_date) – дата, коли клієнт записався на ремонт.
6. Сума оплати (payment\_amount) – вартість ремонту.
7. Дата оплати (payment\_date) – дата оплати за ремонтні послуги.

Відомості про товар:

Містить дані про товари, які продаються в майстерні.

1. Номер товару (detail\_id) – унікальний ідентифікатор товару.
2. Найменування (name) – найменування товару.
3. Ціна (price) – ціна за одиницю товару.

**3 РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ**

**3.1 Побудова інформаційно-логічної моделі даних**

Основне призначення інформаційно-логічної моделі полягає в тому, щоб наочно відобразити дані як сукупність інформаційних об'єктів та зв'язків між ними. Ця модель відображає дані, призначені для зберігання в базі даних.

Інформаційний об'єкт складається з логічно зв'язаних атрибутів, що відображають якісні та кількісні характеристики наочної області.

Аналізуючи предметну область, можна виділити наступні інформаційні об'єкти, які представляють собою ключові складові цієї області (сутності): «Клієнти», «Співробітники», «Замовлення», «Запис на ремонт», «Товари».

Специфікація атрибутів сутностей наведена у таблиці 3.1.

Таблиця 3.1 – Специфікація атрибутів сутностей предметної області

|  |  |  |
| --- | --- | --- |
| Сутність | Атрибут | Тип атрибута |
| Клієнти | Номер клієнта | Числовий |
| ПІБ | Текстовий |
| Номер телефону | Текстовий |
| Співробітники | Номер співробітника | Числовий |
| ПІБ | Текстовий |
| Посада | Текстовий |
| Номер телефону | Текстовий |
| Заробітня плата | Числовий |
| Замовлення | Номер замовлення | Числовий |
| FK Номер клієнта | Числовий |

Продовження таблиці 3.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
|  | FK Номер товару | Числовий |
| FK Номер співробітника | Числовий |
| Дата замовлення | Датовий |
| Сума оплати | Числовий |
| Дата оплати | Датовий |
| Запис на ремонт | Номер запису на ремонт | Числовий |
| FK Номер клієнта | Числовий |
| FK Номер співробітника | Числовий |
| Причина ремонту | Текстовий |
| Дата запису | Датовий |
| Сума оплати | Числовий |
| Дата оплати | Датовий |
| Товари | Номер товару | Числовий |
| Назва товару | Текстовий |
| Ціна товару | Числовий |

**3.2 Визначення функціональних залежностей між реквізитами**

Для визначення того, як різні реквізити пов'язані між собою, ми будемо розглядати бінарні зв'язки між об'єктами та встановлювати їх властивості:

– В одного співробітника може бути багато замовлень: 1:N;

– В одного співробітника може бути багато записів на ремонт: 1:N;

– В одного клієнта може бути багато замовленнь: 1:N;

– В одного клієнта може бути багато записів на ремонт 1:N;

– Багато товарів можуть бути у багатьох замовленнях: M:N.

На підставі проведеного аналізу будуємо ER-діаграму, подану на рисунку 3.1.

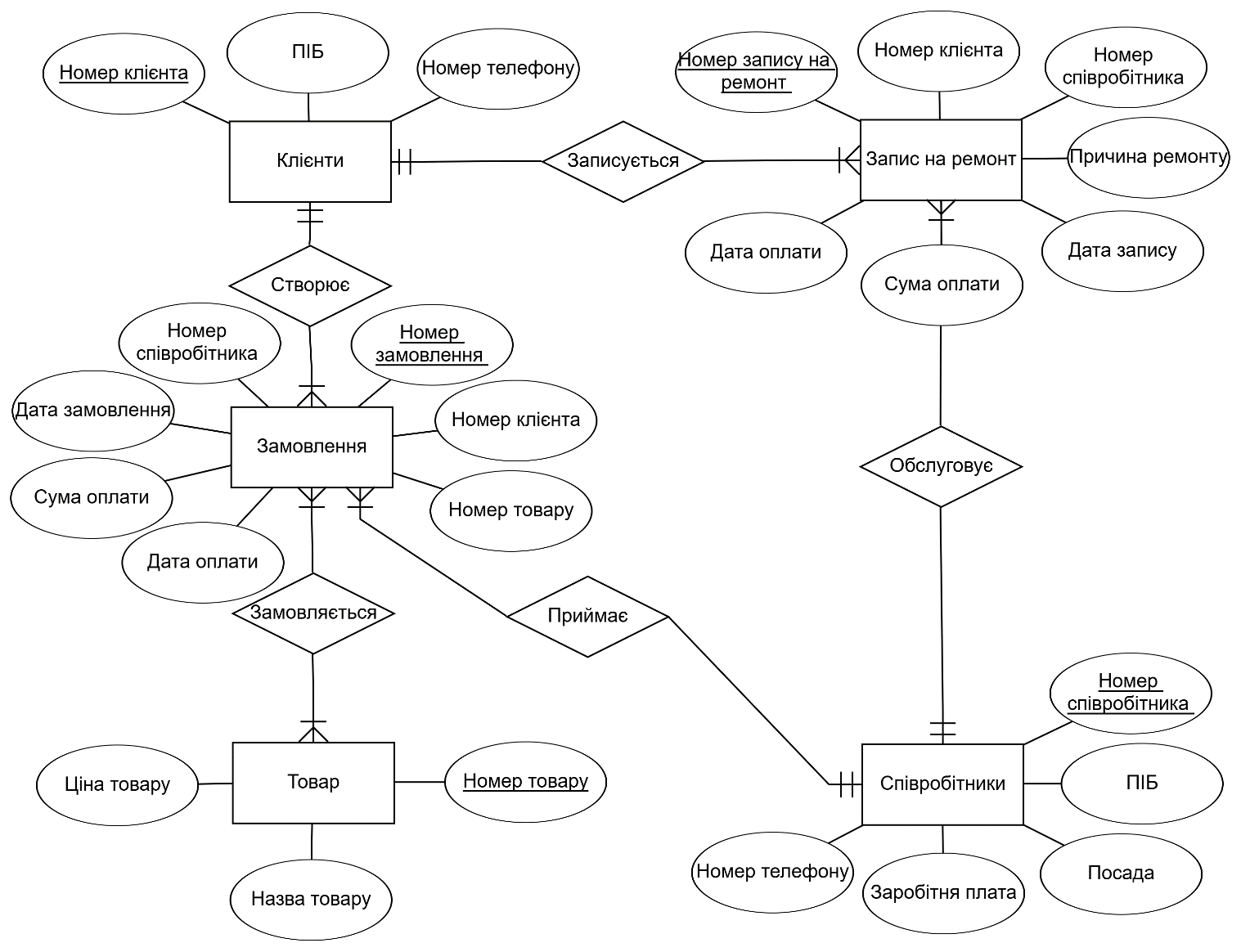


Рисунок 3.1 – ER-діаграма предметної області

Щоб скласти інформаційно-логічну модель предметної області, складемо словник даних, в якому запишемо специфікації об’єктів, зв’язків та атрибутів. Словник даних зведемо в таблиці 3.2.

Таблиця 3.2 – Словник даних

|  |  |  |
| --- | --- | --- |
| 1.Специфікація об’єктів | | |
| Сутність | Атрибут | Тип атрибута |
| Клієнти | Номер клієнта | Числовий |
| ПІБ | Текстовий |
| Номер телефону | Текстовий |
| Співробітники | Номер співробітника | Числовий |
| ПІБ | Текстовий |
| Посада | Текстовий |
|  | Номер телефону | Текстовий |
| Заробітня плата | Числовий |
| Замовлення | Номер замовлення | Числовий |
| FK Номер клієнта | Числовий |
| FK Номер співробітника | Числовий |
| Дата замовлення | Датовий |
| Сума оплати | Числовий |
| Дата оплати | Датовий |
| Замовлення\_Товар | FK Номер замовлення | Числовий |
| FK Номер товару | Числовий |
| Кількість товару | Числовий |
| Запис на ремонт | Номер запису на ремонт | Числовий |
| FK Номер клієнта | Числовий |
| FK Номер співробітника | Числовий |
| Причина ремонту | Текстовий |

Продовження таблиці 3.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | | 2 | | 3 | |
|  | | Дата запису | | Датовий | |
| Сума оплати | | Числовий | |
| Дата оплати | | Датовий | |
| Товари | | Номер товару | | Числовий | |
| Назва товару | | Текстовий | |
| Ціна товару | | Числовий | |
| 2. Специфікація зв’язків | | | | | |
| Зв’язок | | | | | Показник кардинальності |
| Співробітники – Замовлення | | | | | 1:N |
| Співробітники – Запис на ремонт | | | | | 1:N |
| Клієнти – Замовлення | | | | | 1:N |
| Клієнти – Запис на ремонт | | | | | 1:N |
| Товар – Замовлення\_Товар | | | | | 1:N |
| Замовлення – Замовлення\_Товар | | | | | 1:N |
| 3.Специфікація атрибутів | | | | | |
| Об’єкт | Атрибут | | Тип даних та формат | | Обмеження |
| Клієнти | Номер клієнта | | INT | | Унікальний, не пустий |
| ПІБ | | VARCHAR | | Не пустий |
| Номер телефону | | VARCHAR | | Не пустий |
| Співробітники | Номер співробітника | | INT | | Унікальний, не пустий |

Продовження таблиці 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | |
|  | ПІБ | VARCHAR | Не пустий | |
| Посада | VARCHAR | Не пустий | |
| Номер телефону | VARCHAR | Не пустий | |
| Заробітня плата | DECIMAL | Не пустий | |
| Замовлення | Номер замовлення | INT | Унікальний, не пустий | |
| Номер клієнта | INT | Унікальний, не пустий | |
| Номер співробітника | INT | Унікальний, не пустий | |
| Дата замовлення | DATETIME | Не пустий | |
| Сума оплати | DECIMAL |  | |
|  | Дата оплати | DATETIME | |  |
| Замовлення\_Товар | Номер замовлення | INT | | Унікальний, не пустий |
| Номер товару | INT | | Унікальний, не пустий |
| Кількість товару | INT | | Не пустий |
| Запис на ремонт | Номер запису на ремонт | INT | | Унікальний, не пустий |
| Номер клієнта | INT | | Унікальний, не пустий |

Продовження таблиці 3.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
|  | Номер співробітника | INT | Унікальний, не пустий |
| Причина ремонту | VARCHAR | Не пустий |
| Дата запису | DATETIME | Не пустий |
| Сума оплати | DECIMAL |  |
| Дата оплати | DATETIME |  |
| Товар | Номер товару | INT | Унікальний, не пустий |
| Назва товару | VARCHAR | Не пустий |
| Ціна товару | DECIMAL | Не пустий |

На підставі цих даних складемо інформаційно-логічну модель предметної області, зображену на рисунку 3.2.

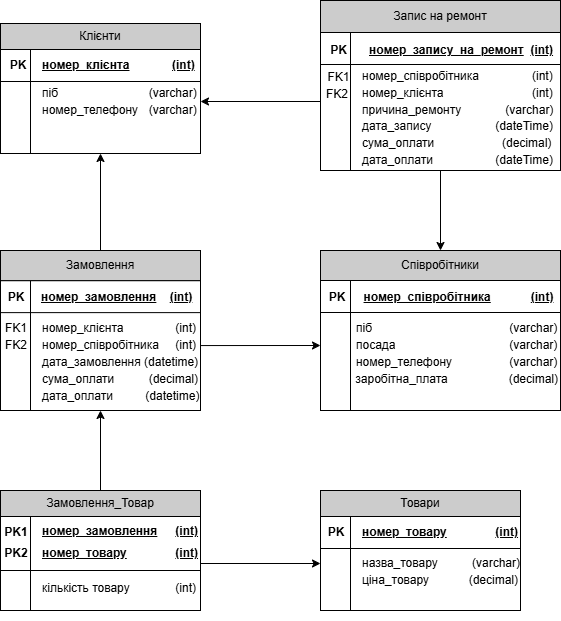


Рисунок 3.2 – Інформаційно-логічна модель предметної області

**3.3 Фізична модель бази даних та склад таблиць**

Для збереження даних, що використовуються у додатку, була створена база даних з назвою «bicycle-repair-shop-kursova-polynko-kn-23-1», у якій є 7 таблиць (таблицю Users додано для зберігання інформації про користувачів застосунку). Фізичну структуру БД наведено на рисунку 3.3.

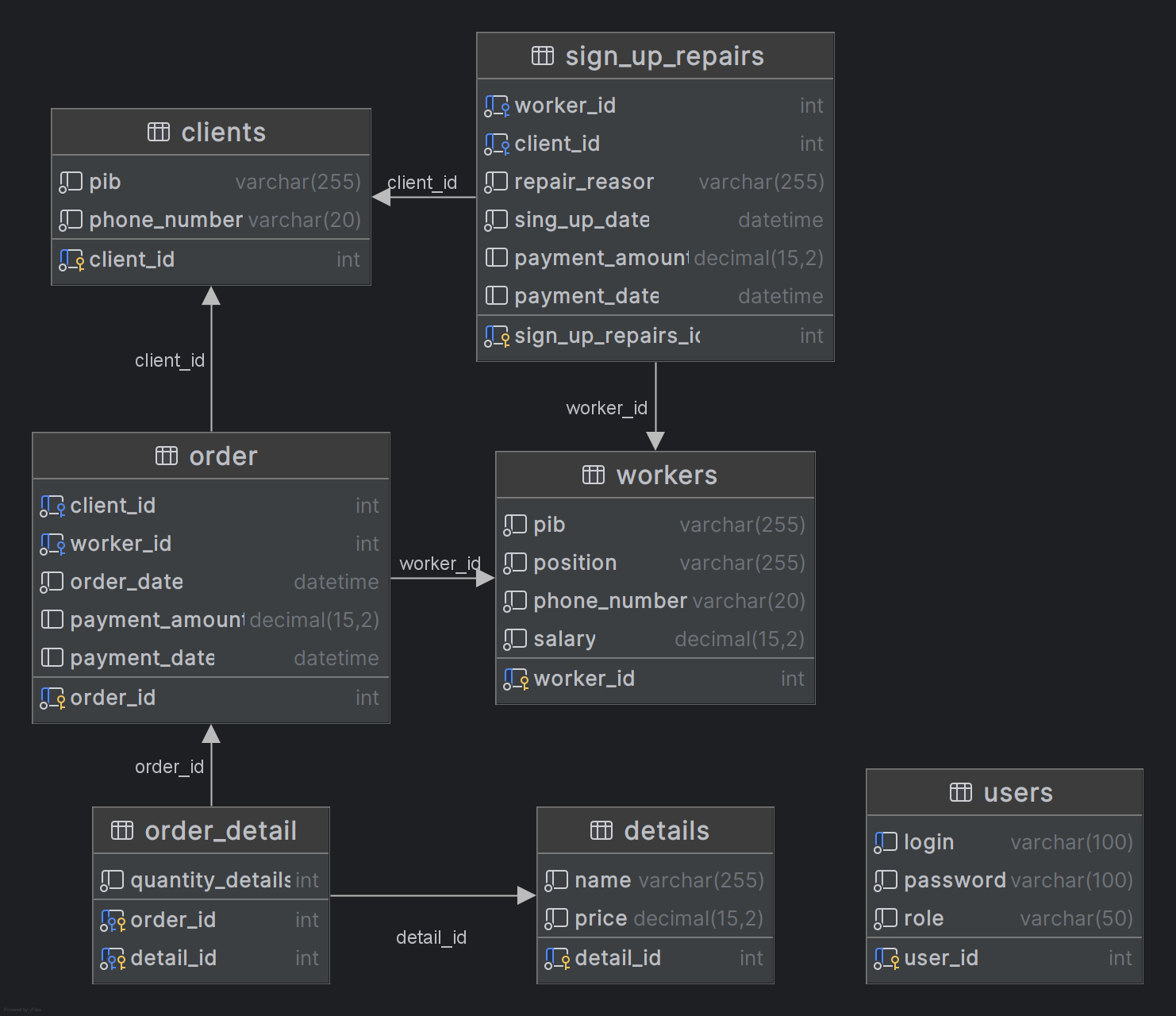


Рисунок 3.3 – Фізична модель бази даних

Таблиця 3.3 призначена для зберігання даних про клієнтів, які замовляють товари та записуються ремонт. Вона містить контактну інформацію про людей, що звертаються до веломайстерні.

Таблиця 3.3 – Таблиця «clients»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | client\_id | INT | Універсальний номер |
| 2 | pib | VARCHAR(255) | ПІБ клієнтів |
| 3 | phone\_number | VARCHAR(20) | Номер телефону |

Таблиця 3.4 призначена для зберігання даних про співробітників, які створюють замовлення та записи на ремонт, обслуговують велосипеди та клієнтів. Вона містить містить контактну інформацію про людей, що працюють у веломайстерні, їх посаду та заробітню плату.

Таблиця 3.4 – Таблиця «workers»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | worker\_id | INT | Універсальний номер |
| 2 | pib | VARCHAR(255) | ПІБ клієнтів |
| 3 | position | VARCHAR(255) | Посада |
| 4 | phone\_number | VARCHAR(20) | Номер телефону |
| 5 | salary | DECIMAL(15,2) | Заробітня плата |

Таблиця 3.5 призначена для зберігання даних про товари, які продаються клієнтам. Вона містить містить інформацію про товар, їх найменування та ціну.

Таблиця 3.5 – Таблиця «details»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | detail\_id | INT | Універсальний номер |
| 2 | name | VARCHAR(255) | Найменування |
| 3 | price | DECIMAL(15,2) | Ціна |

Таблиця 3.6 призначена для зберігання даних про користувачів, які взаємодіють з інтерфейсом та базою даних. Вона містить містить інформацію про користувачів, їх логін, пароль та роль.

Таблиця 3.6 – Таблиця «users»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | order\_id | INT | Універсальний номер |
| 2 | login | VARCHAR(100) | Логін |
| 3 | password | VARCHAR(100) | Пароль |
| 3 | role | VARCHAR(50) | Роль доступу |

Таблиця 3.7 призначена для зберігання даних про замовлення, які містять інформацію про клієнтів, що замовляють товари, співробітників, що оброблють замовлення, дату замовлення, суму до сплати та дату сплати замовлення

Таблиця 3.7 – Таблиця «order»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | user\_id | INT | Універсальний номер |
| 2 | client\_id | INT | Універсальний зовнішний номер |
| 3 | worker\_id | INT | Універсальний зовнішний номер |
| 4 | order\_date | DATETIME | Дата замовлення |
| 5 | payment\_amount | DECIMAL(15,2) | Сума оплати |
| 6 | payment\_date | DATETIME | Дата оплати |

Таблиця 3.8 призначена для зберігання даних про деталі замовлення, які містять інформацію про товари, які замовив клієнт у певному замовленні та їх кількість.

Таблиця 3.8 – Таблиця «order\_detail»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | order\_id | INT | Універсальний зовнішний номер |
| 2 | detail\_id | INT | Універсальний зовнішний номер |
| 3 | quantity\_details | INT | Кількість замовлених товарів |

Таблиця 3.9 призначена для зберігання даних про замовлення, які містять інформацію про клієнтів, що замовляють товари, співробітників, що оброблють замовлення, дату замовлення, суму до сплати та дату сплати замовлення

Таблиця 3.9 – Таблиця «sign\_up\_repairs»

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поля | Найменування поля | Тип даних | Призначення поля |
| 1 | sign\_up\_repairs\_id | INT | Універсальний номер |
| 2 | worker\_id | INT | Універсальний зовнішний номер |
| 3 | client\_id | INT | Універсальний зовнішний номер |
| 4 | repair\_reason | VARCHAR(255) | Інформація про причину звернення |
| 5 | sign\_up\_date | DATETIME | Дата запису |
| 6 | payment\_amount | DECIMAL(15,2) | Сума оплати |
| 7 | payment\_date | DATETIME | Дата оплати |

**3.4 Створення ключів та опис зв’язків**

В реляційній моделі бази даних ключові елементи, такі як первинні ключі (PRIMARY KEY) та зовнішні ключі (FOREIGN KEY), відіграють вирішальну роль у забезпеченні цілісності та логічної структури даних.

Первинний ключ – це стовпець або набір стовпців, що унікально ідентифікує кожен рядок у таблиці. Ці ключі є основою для забезпечення унікальності даних та їх взаємозв'язків у реляційних базах даних. У базі даних для бару первинні ключі встановлені для наступних таблиць:

* + Клієнти: client\_id;
  + Робітники: worker\_id;
  + Користувачі: user\_id;
  + Товари: detail\_id;
  + Замовлення: order\_id;
  + Запис на ремонт: sign\_up\_repairs\_id;
  + Деталі замовлення: order\_id, detail\_id.

Ці ключі мають властивість AUTO INCREMENT, що гарантує автоматичне присвоєння унікального значення при додаванні нових записів.

Зовнішні ключі використовуються для забезпечення зв'язків між таблицями, дозволяючи рядкам однієї таблиці посилатися на рядки іншої. Наприклад, поля order\_id та detail\_id у таблиці "Деталі замовлення" є зовнішніми ключами, що посилаються на первинний ключ у таблицях "Замовлення" та «Товари»

В таблиці 3.10 наведені зовнішні ключі з усіх таблиць.

Кожен з цих ключів має властивість NOT NULL, що запобігає введенню порожніх значень. Обмеження PRIMARY KEY гарантує унікальність даних у відповідних полях. Зовнішні ключі забезпечують логічну узгодженість між таблицями, що дозволяє здійснювати ефективні запити та підтримує цілісність даних під час оновлень і видалень.

Ці зв’язки не лише відображають логіку предметної області, а й забезпечують **референційну цілісність** даних. При оновленні або видаленні записів із пов’язаних таблиць зовнішні ключі запобігають виникненню «висячих» посилань, тобто таких, що ведуть до неіснуючих записів. У випадку Supabase, для зовнішніх ключів можуть застосовуватися політики ON UPDATE та ON DELETE (наприклад, CASCADE, RESTRICT або SET NULL), які додатково регулюють поведінку при змінах.

Таблиця 3.10 – Зовнішні ключі БД.

|  |  |  |
| --- | --- | --- |
| Таблиця | Зовнішний ключ | Посилальна таблиця |
| Замовлення | order\_clients\_client\_id\_fk | Клієнти |
| order\_workers\_worker\_id\_fk | Робітники |
| Деталі замовлення | order\_detail\_details\_detail\_id\_fk | Товари |
| order\_detail\_order\_order\_id\_fk | Замовлення |
| Запис на ремонт | sign\_up\_repairs\_clients\_client\_id\_fk | Клієнти |
| sign\_up\_repairs\_workers\_worker\_id\_fk | Робітники |

Ці ключі та зв'язки є фундаментальними для структури бази даних і грають важливу роль у забезпеченні надійності, ефективності та доступності даних.

**3.5 Підтримка цілісності даних**

Для підвищення цілісності даних у базі впроваджено систему тригерів, що автоматизують контроль за правильністю змін у таблицях. Зокрема:

* after\_order\_detail\_insert та after\_order\_detail\_update – автоматично перераховують загальну суму замовлення при додаванні або зміні деталей у таблиці order\_detail (Додаток В);
* before\_order\_delete – забезпечує каскадне видалення пов’язаних записів із таблиці order\_detail при видаленні замовлення (Додаток В);
* before\_order\_insert та before\_order\_update – перевіряють, щоб у поле worker\_id замовлення потрапляли лише працівники з посадою Manager (Додаток В);
* before\_sign\_up\_repairs\_insert та before\_sign\_up\_repairs\_update – аналогічно обмежують внесення у таблицю sign\_up\_repairs, дозволяючи вказати лише майстрів із посадою Master (Додаток В).

Ці тригери виконують роль автоматизованих перевірок та обмежень, що унеможливлюють порушення бізнес-логіки на рівні бази даних.

Таким чином, впроваджене поєднання аутентифікації, рольової моделі та тригерів забезпечує високий рівень контролю над доступом і змінами в системі. Це дозволяє не лише зберігати конфіденційність і цілісність даних, а й забезпечує автоматичний захист від помилок користувача або навмисного втручання. У перспективі структура безпеки може бути розширена за рахунок реалізації додаткових рівнів доступу, системи журналювання дій користувачів та двофакторної аутентифікації.

**3.6 Забезпечення безпеки даних**

Забезпечення безпеки даних є критично важливим елементом функціонування інформаційної системи комп’ютерного клубу, особливо враховуючи доступ до бази даних через мережевий інтерфейс. У системі реалізовано низку заходів, спрямованих на захист даних від несанкціонованого доступу та порушення цілісності.

Центральною складовою системи безпеки є таблиця users, яка зберігає облікові записи користувачів, їх логіни, паролі та рівень доступу. Ролі користувачів реалізовано через поле role, яке визначає права доступу до інтерфейсу бази даних. На основі цієї інформації формується рольова модель, яка дозволяє розмежовувати повноваження:

* Адміністратор має повний доступ до всіх функцій системи, включаючи основне меню і модифікацію таблиці користувачів;
* Менеджер та майстер обмежені лише роботою з основним меню особистої інформації, оформленням замовлень та доступом до загальнодоступного контенту.

Таке розмежування дозволяє запобігти неконтрольованому доступу до критичних функцій і зберігати конфіденційність даних.

**3.7 Типові відображення**

1. Розширені деталі замовлення (рис. 3.4)

Подання expanded\_order\_detail дозволяє переглядати детальну інформацію про виконані замовлення, включно з прізвищем працівника та клієнта, датою замовлення, назвою товару, кількістю одиниць та загальною вартістю. Така агрегація спрощує аналіз продажів.

CREATE VIEW expanded\_order\_detail AS

SELECT od.order\_id, w.pib AS worker\_pib, c.pib AS client\_pib, o.order\_date, d.name AS detail\_name, SUM(od.quantity\_details) AS quantity,

SUM(d.price \* od.quantity\_details) AS total\_price

FROM order\_detail od

JOIN `order` o ON od.order\_id = o.order\_id

JOIN workers w ON o.worker\_id = w.worker\_id

JOIN clients c ON o.client\_id = c.client\_id

JOIN details d ON od.detail\_id = d.detail\_id

WHERE o.payment\_date IS NOT NULL

GROUP BY od.order\_id, w.pib, c.pib, o.order\_date, d.name;

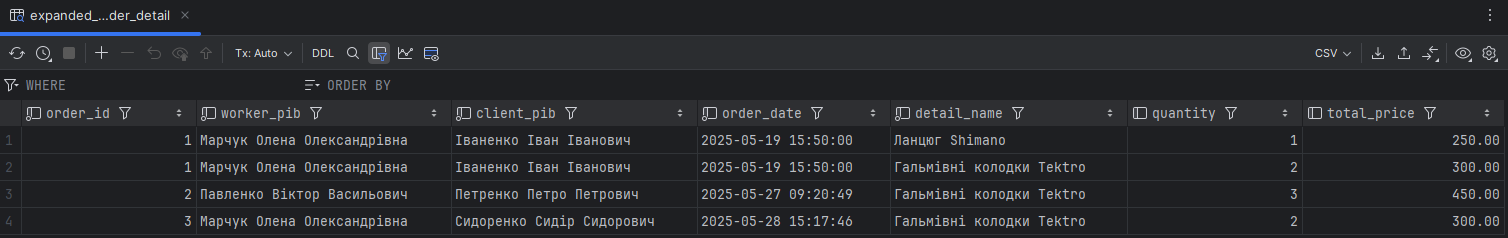


Рисунок 3.4 – Розширені дані про замовлення

1. Розширені записи на ремонт (рис. 3.5)

У expanded\_sign\_up\_repairs зібрано дані про завершені ремонти – імена працівника та клієнта, дату запису та суму оплати. Це дозволяє швидко формувати звіти по наданих ремонтних послугах.

CREATE VIEW expanded\_sign\_up\_repairs AS

SELECT sur.sign\_up\_repairs\_id, w.pib AS worker\_pib, c.pib AS client\_pib,

sur.sign\_up\_date, sur.payment\_amount

FROM sign\_up\_repairs sur

JOIN workers w ON sur.worker\_id = w.worker\_id

JOIN clients c ON sur.client\_id = c.client\_id

WHERE sur.payment\_date IS NOT NULL;

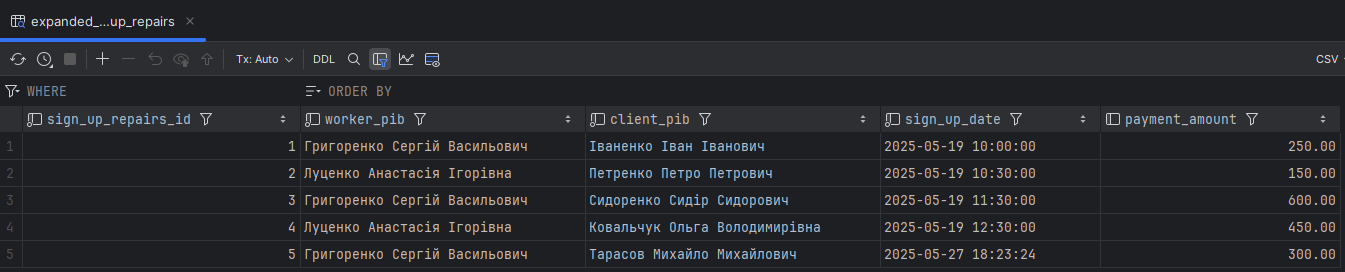


Рисунок 3.5 – Розширені дані про записи на ремонт

1. Продажі товарів за поточний місяць (рис. 3.6)

Подання monthly\_details\_sales є звітом продажу деталей і відображає кількість проданих одиниць кожного товару за поточний календарний місяць, з урахуванням лише оплачених замовлень.

CREATE VIEW monthly\_details\_sales AS

SELECT

d.detail\_id,

d.name,

d.price,

COALESCE(SUM(od.quantity\_details), 0) AS total\_sold

FROM details d

LEFT JOIN order\_detail od ON d.detail\_id = od.detail\_id

LEFT JOIN `order` o ON od.order\_id = o.order\_id

AND o.payment\_date IS NOT NULL

AND MONTH(o.payment\_date) = MONTH(CURRENT\_DATE())

AND YEAR(o.payment\_date) = YEAR(CURRENT\_DATE())

GROUP BY d.detail\_id, d.name, d.price;

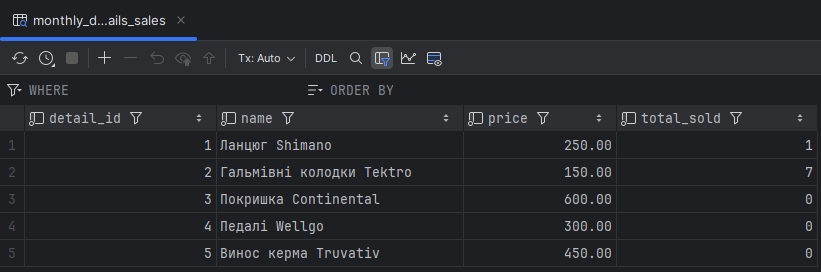


Рисунок 3.6 – Звіт продажу деталей за останній місяць

1. Місячний прибуток працівників (рис. 3.7)

monthly\_worker\_profit є фінансовим звітом і об’єднує інформацію про зароблені суми кожним працівником як через замовлення, так і через ремонти. Це корисно для оцінки ефективності роботи персоналу.

CREATE VIEW monthly\_worker\_profit AS

SELECT

w.worker\_id,

w.pib AS worker\_pib,

w.position,

COALESCE(order\_profit.total\_order\_profit, 0) +

COALESCE(repair\_profit.total\_repair\_profit, 0) AS profit

FROM workers w

LEFT JOIN (

SELECT

o.worker\_id,

SUM(o.payment\_amount) AS total\_order\_profit

FROM `order` o

WHERE o.payment\_date IS NOT NULL

AND MONTH(o.payment\_date) = MONTH(CURRENT\_DATE())

AND YEAR(o.payment\_date) = YEAR(CURRENT\_DATE())

GROUP BY o.worker\_id

) AS order\_profit ON w.worker\_id = order\_profit.worker\_id

LEFT JOIN (

SELECT

sur.worker\_id,

SUM(sur.payment\_amount) AS total\_repair\_profit

FROM sign\_up\_repairs sur

WHERE sur.payment\_date IS NOT NULL

AND MONTH(sur.payment\_date) = MONTH(CURRENT\_DATE())

AND YEAR(sur.payment\_date) = YEAR(CURRENT\_DATE())

GROUP BY sur.worker\_id

) AS repair\_profit ON w.worker\_id = repair\_profit.worker\_id;

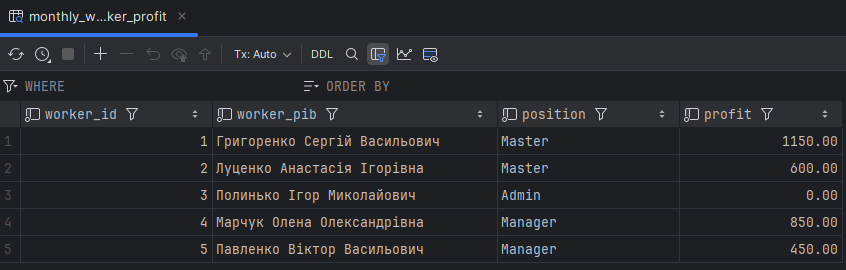


Рисунок 3.7 – Звіт прибутку від співробітників

1. Попит на товари (рис. 3.8)

product\_demand показує загальну кількість проданих одиниць кожного товару за весь період, відсортовану за популярністю. Дозволяє виявити найбільш ходові позиції.

CREATE VIEW product\_demand AS

SELECT d.detail\_id, d.name, d.price,

COALESCE(SUM(od.quantity\_details), 0) AS total\_sold

FROM details d

LEFT JOIN order\_detail od ON d.detail\_id = od.detail\_id

GROUP BY d.detail\_id, d.name, d.price

ORDER BY total\_sold DESC;

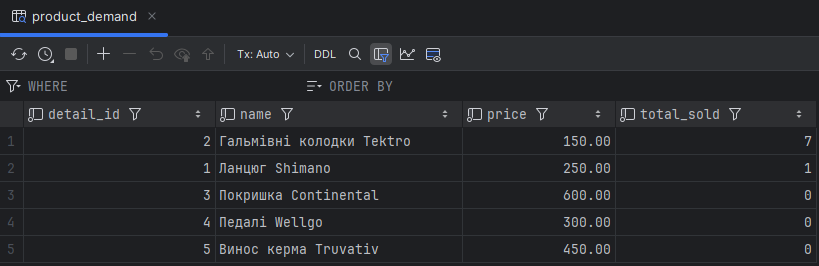


Рисунок 3.8 – Попит товарів

**4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**4.1 Функціональне призначення застосунку**

Кожна форма з текстовими полями передбачає можливість очищення усіх полів через кнопку «Очистити» у menuStrip. Також через menuStrip можна покинути форму та запустити додатковий запит в місцях, де це передбачено.

На формах з таблицею передбачено автоматичне занесення даних з поля DataGridView у відповідні поля (TextBox, ComboBox, DateTimePicker, CheckBox). Усі таблиці при запуску форми виводять останні додані поля.

Весь застосунок має зображення з функцією кліку для зміни візуальної теми форм.

Після запуску програми відкривається вікно авторизації, в якому користувач може:

* Ввести логін та пароль.
* Змінити конфігурацію підключення до бази даних.

При спробі входу програма порівнює дані, введені користувачем та при знаходженні користувача визначає роль.

Якщо користувач виявляється адміністратором, то він переходить у меню, де може перейти до таблиці користувачів або увійти у головне меню.

Інші користувачі відразу переходять у основне меню. З основного меню ми маємо доступ до усіх інших таблиць бази даних:

* Форма «Клієнти» містить список усіх клієнтів сервісного центру. Вона дозволяє додавати нових клієнтів, редагувати або видаляти існуючих. Для кожного клієнта вказуються прізвище, ім’я, по батькові, контактні дані та інша необхідна інформація.
* Форма «Замовлення» відображає перелік усіх здійснених замовлень, із зазначенням дати, клієнта, працівника, відповідального за виконання, та суми оплати. З форми можна додавати нові замовлення, змінювати існуючі або переглядати деталі кожного з них.
* Форма «Деталі замовлення» відкривається тільки під час додавання або оновлення запису у таблиці замовлень і дозволяє вказати перелік обраних товарів (деталей), їх кількість, а також автоматично обчислює загальну вартість замовлення. Вона є допоміжною і не використовується окремо.
* Форма «Запис на ремонт» дозволяє фіксувати заявки на проведення ремонтних робіт. Тут вказуються клієнт, працівник, дата запису, сума оплати, а також фіксується дата фактичної оплати, якщо ремонт завершено. Форма спрощує керування поточними ремонтами.
* Форма «Товари» містить перелік усіх доступних деталей, які можуть бути використані у замовленнях. Для кожного товару зазначено назву, ціну та інші характеристики. З форми можна додавати нові товари або змінювати наявні.
* Форма «Працівники» дає змогу керувати списком співробітників. Вказуються їхні ПІБ, посада, контактні дані та інша інформація. Також форму можна використовувати для аналітики щодо прибутковості працівника через пов’язані подання.

Усі меню таблиць передбачають виконання CRUD операцій та пошуку за полями. Меню з таблицями, що містять зовнішні ключі пропонують випадаючі списки для таких полів зі зручним вибором потрібного значення. Наприклад, у поле номеру клієнта в таблиці order ComboBox виводить ПІБ усіх клієнтів з таблиці clients, а при додаванні нового поля – визначає номер клієнта за обраним ПІБ і додає цей номер у таблицю order. Сортування таких списків виконані за спаданням, щоб першочергово отримувати нещодавно доданих клієнтів.

**4.2 Опис програмних модулів**

Структура програмного забезпечення є поєднанням усіх застосованих модулів програми, що утворюють працюючу систему. Кожний модуль повинен реалізовувати функції програми, що вирішують основні завдання проєкту.

Застосунок створений за допомою модулю Windows Forms на базі мови C#.

На рисунку 4.1 зображено оглядач рішень програмного застосунку з файлами форм. В таблиці 4.1 представлено опис модулів форм проекту курсової роботи.

Директорії View було розділено чином єрархії корстувачів (Адміністратор та клієнт). Було додано директорію для вікон помилок та підтверджнь. В директорії адміністратора знаходяться сторінки для керування, перегляду статистики ти редагування даних. Директорія користувача містить вікна та сторінки, які він може переглядати та з якими може взаємодіяти.

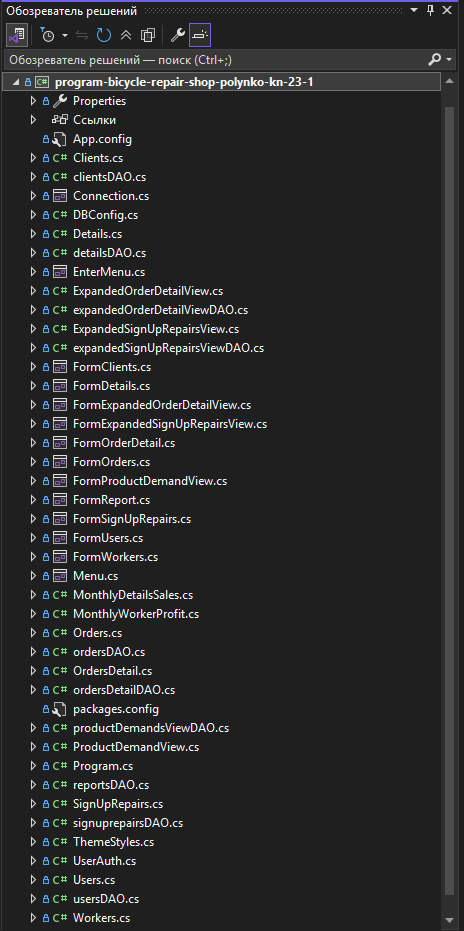


Рисунок 4.1 – Оглядач рішень

Таблиця 4.1 – Призначення модулів форм програми

|  |  |
| --- | --- |
| Назва модуля | Призначення модуля |
| **EnterMenu.cs** | Форма входу в застосунок (авторизація користувача). |
| **Connection.cs** | Клас для керування з’єднанням з базою даних. |
| **FormUsers.cs** | Форма для керування користувачами (перегляд, додавання, редагування, видалення). |
| **Menu.cs** | Головна форма-меню застосунку. |
| **FormReport.cs** | Форма для перегляду та формування звітів. |
| **FormClients.cs** | Форма для керування клієнтами. |
| **FormOrders.cs** | Форма для керування замовленнями. |
| **FormExpandedOrderDetailView.cs** | Форма для розширеного перегляду деталей замовлення. |
| **FormOrderDetail.cs** | Форма для перегляду й редагування деталей конкретного замовлення. |
| **FormSignUpRepairs.cs** | Форма для запису на ремонт. |
| **FormExpandedSignUpRepairsView.cs** | Форма для розширеного перегляду записів на ремонт. |
| **FormDetails.cs** | Форма для керування деталями (запчастинами). |
| **FormProductDemandView.cs** | Форма для перегляду попиту на товари/деталі. |
| **FormWorkers.cs** | Форма для керування працівниками. |

В таблиці 4.2 представлено опис модулів(класів) проекту курсової роботи

Таблиця 4.2 – Файли моделі, класи керування та конфігурації.

|  |  |  |
| --- | --- | --- |
| Назва модуля | Призначення модуля | |
| **Program.cs** | Точка входу в застосунок (Main). | |
| **ThemeStyles.cs** | Клас для керування темами оформлення застосунку (світла/темна тема). | |
| **DBConfig.cs** | Клас для збереження та завантаження конфігурації підключення до бази даних. | |
| **UserAuth.cs** | Клас для автентифікації користувача. | |
| **Users.cs** | | Модель користувача (властивості користувача). |
| **MonthlyWorkerProfit.cs** | | Модель для відображення/розрахунку місячного прибутку працівника. |
| **MonthlyDetailsSales.cs** | | Модель для відображення/розрахунку місячних продажів деталей. |
| **Clients.cs** | | Модель клієнта. |
| **Orders.cs** | | Модель замовлення. |
| **ExpandedOrderDetailView.cs** | | Модель для розширеного подання деталей замовлення. |
| **OrdersDetail.cs** | | Модель деталі замовлення (зв’язок між замовленням і деталлю). |
| **SignUpRepairs.cs** | | Модель запису на ремонт. |
| **ExpandedSignUpRepairsView.cs** | | Модель для розширеного подання записів на ремонт. |
| **Details.cs** | | Модель деталі (запчастини). |
| **ProductDemandView.cs** | | Модель для подання попиту на товари/деталі. |
| **Workers.cs** | | Модель працівника. |

В таблиці 4.3 представлено опис файлів класів доступів до даних БД, що містять відтворення запитів до БД із залученням даних, введених користувачем на формах.

Таблиця 4.3 – Файли класів доступів до даних БД.

|  |  |
| --- | --- |
| Назва модуля | Призначення модуля |
| **usersDAO.cs** | Клас доступу до даних користувачів (CRUD-операції для Users). |
| **reportsDAO.cs** | Клас доступу до даних звітів (формування та отримання звітів). |
| **clientsDAO.cs** | Клас доступу до даних клієнтів. |
| **ordersDAO.cs** | Клас доступу до даних замовлень. |
| **expandedOrderDetailViewDAO.cs** | Клас доступу до даних розширеного подання деталей замовлення. |
| **ordersDetailDAO.cs** | Клас доступу до даних деталей замовлення. |
| **signuprepairsDAO.cs** | Клас доступу до даних записів на ремонт. |
| **expandedSignUpRepairsViewDAO.cs** | Клас доступу до даних розширеного подання записів на ремонт. |
| **detailsDAO.cs** | Клас доступу до даних деталей (запчастин). |
| **productDemandsViewDAO.cs** | Клас доступу до даних подання попиту на товари/деталі. |
| **workersDAO.cs** | Клас доступу до даних працівників (CRUD-операції для Workers). |

Було створено 14 вікон, до них було створено 12 моделей таблиць, 11 класів доступів до даних і додаткові 4 класи для роботи стилей, підключення до БД, відтворення форм і перевірку входу користувача, що показано на рисунку 4.1. Всі класи моделей вигляду наслідують клас ViewModel, який реалізує інтерфейс INotifyPropertyChanged, а також зберігає шляхи до файлів, з зоною доступності protected.

Підключення до БД виконується через ConnectionString, що зчитує дані з dbconfig.json файлу. На формі Connection адміністратор може змінювати відповідні поля конфігурації, щоб застосунок з’єднувався з сервером на любому комп’ютері. Також будь-який користувач, що не увійшов у систему може перейти на форму Connection і змінити конфігурацію для початку роботи, але в такому випадку користувач не має доступу до іншого функціоналу форми, що призначений для адміністратора.

**4.3 Структура інтерфейсу користувача**

Діалог з користувачем система буде здійснювати з допомогою багаторівневого меню, структура якого наведена на рисунку 4.1.

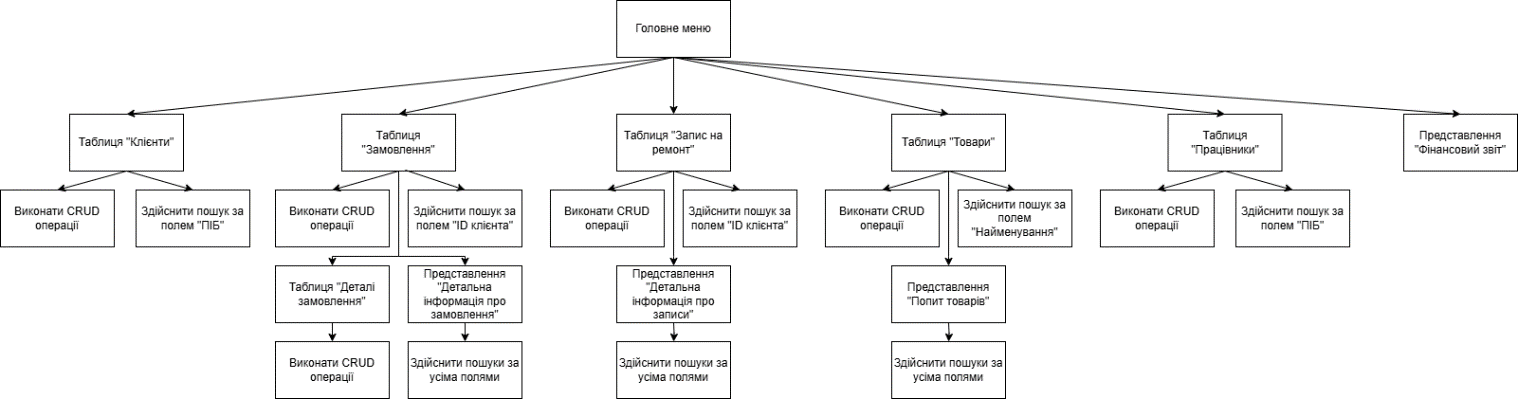


Рисунок 4.1 – Структура головного меню

На рисунку 4.2 зображений узагальнений алгоритм роботи додатку.

При розробці програми створено зручний інтерфейс користувача. Опишемо основні вікна програми:

На рисунку 4.3 зображений інтерфейс стартового вікна входу EnterMenu.

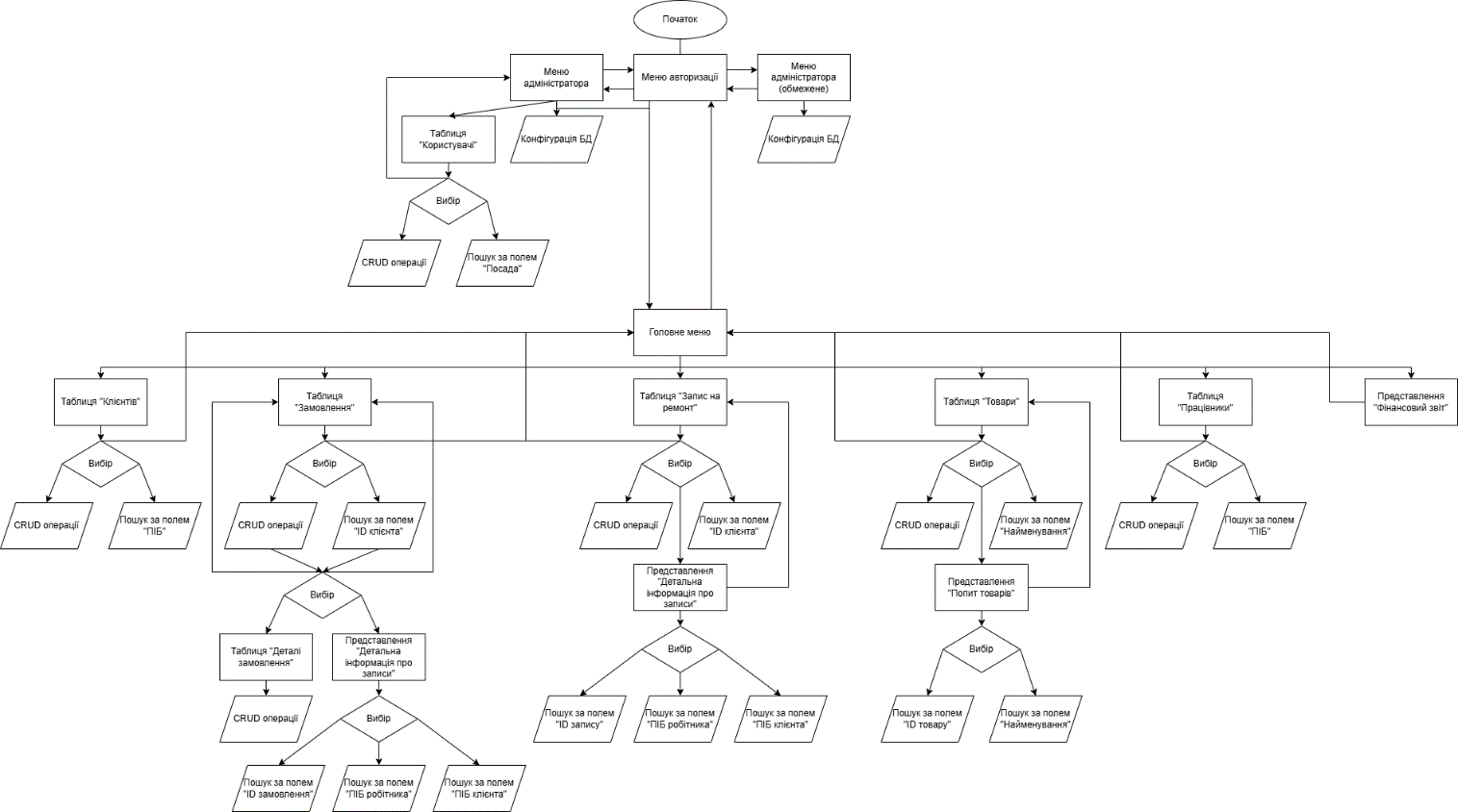


Рисунок 4.2 – Узагальнений алгоритм роботи додатку

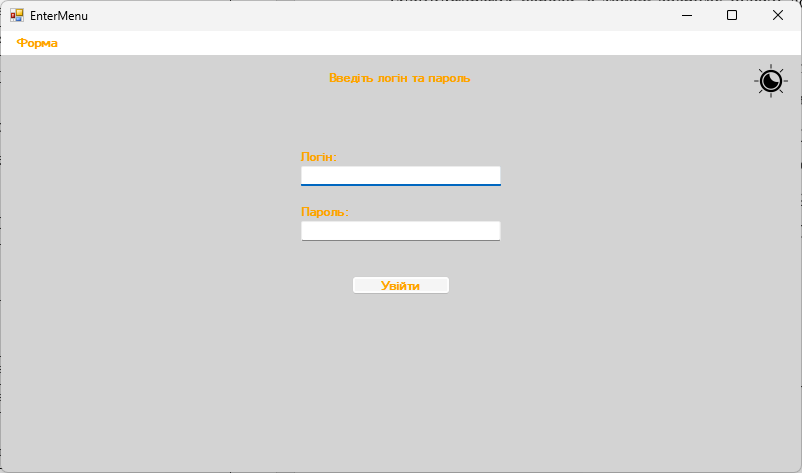


Рисунок 4.3 – Інтерфейс форми «EnterMenu»

На формі бачимо:

* Поле введення логіну;
* Поле введення паролю;
* Кнопка входу;
* Зображення перемикання графічної теми форм;
* MenuStrip «Форма».

При введенні даних у поля Password і Login користувач натискає кнопку «Увійти», яка переносить його на головну сторінку додатка. Якщо входить під даними адміністратора, то на головну сторінку адміністратора.

На рисунку 4.4 зображений вигляд відкритого MenuStrip на формі.

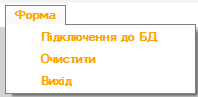


Рисунок 4.4 – MenuStrip форми

На рисунку 4.5 зображений приклад вигляду темної теми форми.

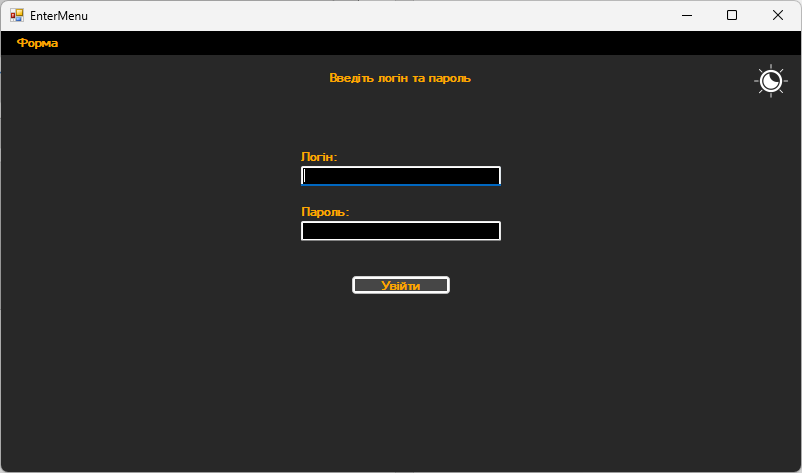


Рисунок 4.5 – Темна тема

На рисунку 4.6 зображена форма адміністратора.

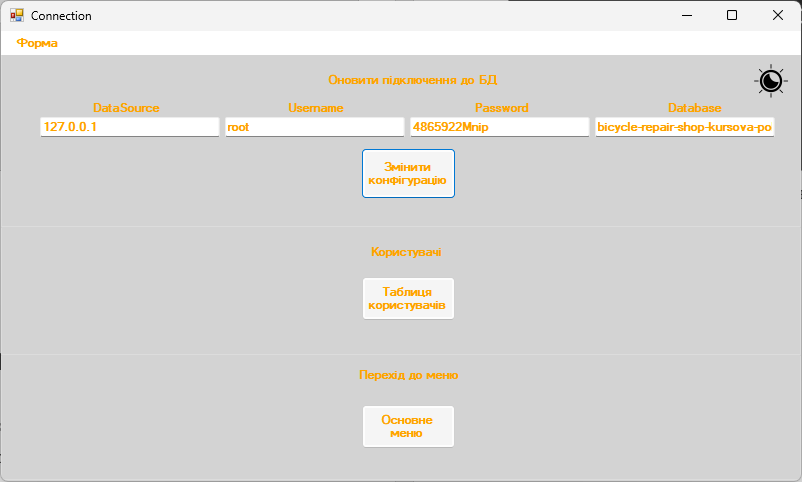


Рисунок 4.6 – Форма «Connection»

На формі бачимо:

* Зображення перемикання графічної теми форм;
* Поле зміни DataSource до БД;
* Поле зміни Username до БД;
* Поле зміни Password до БД;
* Поле зміни Database до БД;
* Кнопка зміни конфігурації;
* Кнопка переходу то таблиці користувачів;
* Кнопка переходу до основого меню.

При зміні конфігурації перезаписуються дані до .json файлу. При наступному включенні форми дані з файлу автоматично підтягуються до форми для зручного редагування та перегляду конфігурації.

На рисунку 4.7 зображена одна із форм таблиць програмного застосунку.

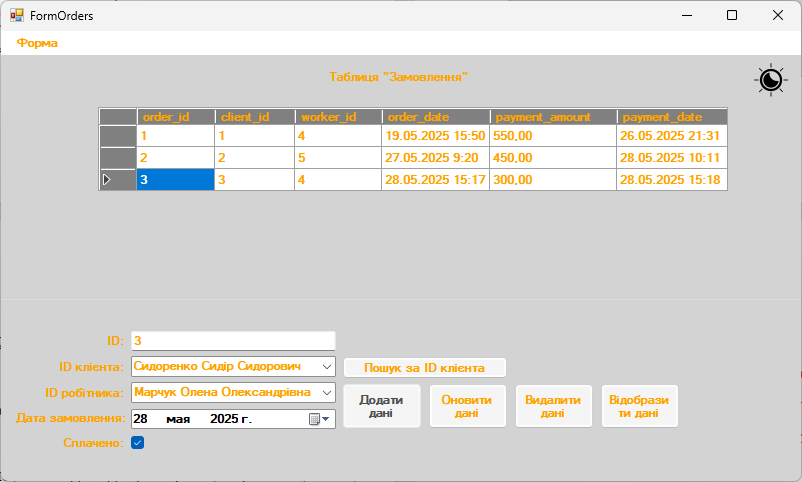


Рисунок 4.7 – Форма «FormOrder»

На формі бачимо:

* Текстові поля, даталогічні поля та чекбокси для виконання CRUD операцій з БД;
* Текстове поле номеру замовлення;
* Випадаючий список клієнтів;
* Випадаючий список співробітників;
* Кнопка додавання даних;
* Кнопка оновлення даних;
* Кнопка видалення даних;
* Кнопка пошуку за полем «ID клієнта»;
* Кнопка відображення даних;
* У menuStrip додана кнопка переходу на представлення «Детальніше про замовлення».

Кнопка додавання даних також викликає форму «Деталі замовлення» для вказання товару, що замовляє клієнт у замовленні

Дані з полів переносяться до текстових полів при натисненні на відповідне поле у DataGridView. Для зовнішніх ключів створені ComboBox, що зчитують номера та видають ПІБ за цими номерами. При додаванні або оновленні полів по обраним ПІБ відбираються відповідні номери і додаються у таблицю. міні конфігурації перезаписуються дані до .json файлу. При наступному включенні форми дані з файлу автоматично підтягуються до форми для зручного редагування та перегляду конфігурації.

На рисунку 4.8 зображене одна із представлень програмного застосунку.

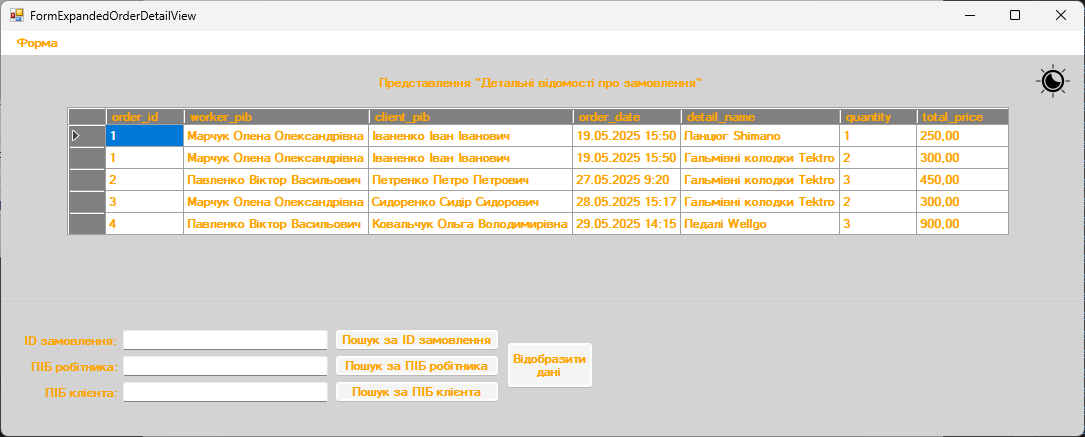


Рисунок 4.8 – Форма «FormExpandedOrderDetailView»

На формі бачимо:

* Текстове поле номеру замовлення;
* Поле ПІБ співробітника;
* Поле ПІБ клієнта;
* Кнопка пошуку за полем «ID замовлення»;
* Кнопка пошуку за полем «ПІБ робітника»;
* Кнопка пошуку за полем «ПІБ клієнта»;
* Кнопка відображення даних.

Форми: «FormClients», «FormDetails», «FormExpandedSignUpRepairsView», «FormOrderDetail», «FormProductDemandView», «FormReport», «FormSignUpRepairs», «FormUsers», «FormWorkers», «Menu» аналогічні вище описаним формам.

**4.4 Інструкція користувача**

Ця інструкція користувача призначена для усіх співробітників веломайстерні, що взаємодіють з інформаційною системою.

Для початку роботи з програмним застосунком користувач має мати встановлений MySQL Server та знаходитися у одній LAN мережі із хостом. Хост має додатково завантажити інтерфейс до роботи з MySQL Server (або працювати з консолі) та відтворити дамп бази даних веломайстерні.

Після встановлення програмного забезпечення його можна запустити. Відкриється меню входу (рис. 4.3). Перед спробою увійти потрібно налаштувати конфігурацію підключення до бази даних.

Через menuStrip форми (рис. 4.4) потрібно перейти у налаштування підключення до БД та вказати власні параметри підключення, зазвичай достатньо змінити до root користувача MySQL Server, що визначався користувачем при встановленні серверу.

Адміністратор має сформувати користувачів у БД, вказати логіни і паролі, а також ролі для усіх співробітників.

Користувачу потрібно ввести свої дані у полях меню входу задля переходу до основого меню. Адміністратор має можливість взаємодіяти з таблицею користувачів через інтерфейс.

В основному меню користувач може перейти до будь-якої таблиці та почати працювати з даними БД.

При додавані даних всі форми передбачають непотрібність вводу поля з первинним ключем, тобто поля з ID, що являються PK для таблиці використовуються для для функцій оновлення та видалення даних (а також пошуку, коли це передбачено формою).

Кожна форма має menuStrip, в якому є функції очищення усіх полів форми, повернення назад, а також виведення різноманітних запитів там, де це передбачено формою.

При натисненні на значок зміни палітри кольорів теми користувач може змінювати темну зі світлої на темної задля особистого комфорту (рис. 4.5). Вигляд значку зміни палітри кольорів зображено на рисунку 4.9.



Рисунок 4.9 – Значок зміни палітри кольорів теми

**ВИСНОВКИ**

У результаті розробки програмного забезпечення для автоматизації роботи веломайстерні було реалізовано настільний застосунок на платформі Windows Forms із підключенням до серверної бази даних MySQL. Основною метою стало спрощення доступу до інформації, підвищення точності обліку та забезпечення зручності для щоденного використання співробітниками майстерні.

Система дозволяє працювати з таблицями клієнтів, працівників, замовлень та послуг у зручному графічному інтерфейсі. Було реалізовано авторизацію з розмежуванням прав доступу: адміністратор має розширені можливості щодо керування користувачами, тоді як звичайні співробітники можуть виконувати операції згідно з призначеними їм ролями. Процес додавання, редагування та видалення даних максимально спрощений — поля з первинними ключами автоматично генеруються на рівні СУБД, що знижує ризик помилок при ручному введенні.

Завдяки використанню menuStrip інтерфейс програми є інтуїтивно зрозумілим: передбачені можливості очищення полів, повернення до попереднього вікна, формування запитів тощо. Наявність перемикача темної та світлої тем дозволяє користувачам адаптувати застосунок до власних уподобань, що позитивно впливає на зручність та ергономіку.

Інформаційна система повністю готова до впровадження в локальній мережі майстерні. Вона забезпечує збереження даних на сервері, можливість резервного копіювання та масштабування у разі зростання обсягу даних. Таким чином, поставлені завдання було виконано успішно, а отримане рішення відповідає практичним вимогам реального бізнесу.

**СПИСОК ЛІТЕРАТУРИ**

1. Завадський І.О., Основи баз даних: навчальний посібник. – Київ: 2011. - 192 с.
2. Сегеда І.В., Проектування та використання баз даних. - Київ: КПІ ім. Ігоря Сікорського, 2021. – 49с.
3. Омельчук Л. Л. Об’єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник. Київ: 2021. 265 с.
4. Алхімова С. М. Обєктно-орієнтоване програмування: підручник. У 2-х ч. Ч. 2: Обєктно-орієнтований підхід до розробки програмного забезпечення. Київ: КГП ім. Ігоря Сікорського, Вид-во «Політехніка», 2019. 192 с.
5. Настенко Д. В., Нестерко А. Б. Об’єктно-орієнтоване програмування. Частина 2: Основи об’єктно-орієнтованого програмування на мові C#: навчальний посібник. Київ: НТУУ «КПІ», 2016. 84 с.
6. Coronel C., Morris S. Database Systems: Design, Implementation, & Management. - Cengage Learning; 13th edition, 2018. - 816 с.
7. Garcia-Molina H., Ullman J., Widom J., Database Systems: The Complete Book - Pearson; 2nd edition, 2008. - 1248 с.
8. C#: підручник. URL: <https://w3schoolsua.github.io/cs/index.html> #gsc.tab=0
9. Документація по C#. URL: https://learn.microsoft.com/en- us/dotnet/csharp/
10. Флеш О.В., Кравченко М.В. Створення інформаційних систем з використанням Windows Forms: навчальний посібник. – Київ: НАУ, 2017. – 112 с.
11. Тимченко В.І. Технології розробки програмних продуктів у середовищі Microsoft Visual Studio. – Київ: КНЕУ, 2020. – 196 с.
12. Perryman L. Programming Microsoft Windows Forms. – Microsoft Press, 2005. – 1056 с.

**Додаток А  
Скрипти створення бази даних та таблиць**

create schema 'bicycle-repair-shop-kursova-polynko-kn-23-1';

use `bicycle-repair-shop-kursova-polynko-kn-23-1`;

create table clients

(

client\_id int auto\_increment

primary key,

pib varchar(255) not null,

phone\_number varchar(20) not null

);

create table details

(

detail\_id int auto\_increment

primary key,

name varchar(255) not null,

price decimal(15, 2) not null

);

create table users

(

user\_id int auto\_increment

primary key,

login varchar(100) not null,

password varchar(100) not null,

role varchar(50) not null,

constraint login

unique (login)

);

create table workers

(

worker\_id int auto\_increment

primary key,

pib varchar(255) not null,

position varchar(255) not null,

phone\_number varchar(20) not null,

salary decimal(15, 2) not null

);

create table `order`

(

order\_id int auto\_increment

primary key,

client\_id int not null,

worker\_id int not null,

order\_date datetime not null,

payment\_amount decimal(15, 2) null,

payment\_date datetime null,

constraint order\_clients\_client\_id\_fk

foreign key (client\_id) references clients (client\_id),

constraint order\_workers\_worker\_id\_fk

foreign key (worker\_id) references workers (worker\_id)

);

create table order\_detail

(

order\_id int not null,

detail\_id int not null,

quantity\_details int not null,

primary key (order\_id, detail\_id),

constraint order\_detail\_details\_detail\_id\_fk

foreign key (detail\_id) references details (detail\_id),

constraint order\_detail\_order\_order\_id\_fk

foreign key (order\_id) references `order` (order\_id)

);

create table sign\_up\_repairs

(

sign\_up\_repairs\_id int auto\_increment

primary key,

worker\_id int not null,

client\_id int not null,

repair\_reason varchar(255) not null,

sign\_up\_date datetime not null,

payment\_amount decimal(15, 2) null,

payment\_date datetime null,

constraint sign\_up\_repairs\_clients\_client\_id\_fk

foreign key (client\_id) references clients (client\_id),

constraint sign\_up\_repairs\_workers\_worker\_id\_fk

foreign key (worker\_id) references workers (worker\_id)

);

**Додаток Б  
Лістинг програмних файлів**

1. Файл DBConfig.cs:

public class DBConfig

{

public string DataSource { get; set; }

public string Username { get; set; }

public string Password { get; set; }

public string Database { get; set; }

private static string dataSource;

private static string username;

private static string password;

private static string database;

public static string LoadConfig()

{

string jsonString = File.ReadAllText("dbconfig.json");

var config = JsonSerializer.Deserialize<DBConfig>(jsonString);

dataSource = config.DataSource;

username = config.Username;

password = config.Password;

database = config.Database;

return $"datasource={dataSource};username={username};password={password};database={database}";

}

public string GetConnectionString()

{

return $"datasource={DataSource};username={Username};password={Password};database={Database}";

}

}

2. Файл UserAuth.cs:

public static class UserAuth

{

public static string ConnectionString { get; set; }

public static Users Authenticate(string username, string password)

{

using (var conn = new MySqlConnection(ConnectionString))

{

conn.Open();

string query = "SELECT login, password, role FROM users WHERE login = @login AND password = @password LIMIT 1";

using (var cmd = new MySqlCommand(query, conn))

{

cmd.Parameters.AddWithValue("@login", username);

cmd.Parameters.AddWithValue("@password", password);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Users

{

login = reader.GetString("login"),

password = reader.GetString("password"),

role = reader.GetString("role")

};

}

}

}

}

return null;

}

}

3. Файл Order.cs:

public class Orders

{

public int order\_id { get; set; }

public int client\_id { get; set; }

public int worker\_id { get; set; }

public DateTime order\_date { get; set; }

public Decimal? payment\_amount { get; set; }

public DateTime? payment\_date { get; set; }

}

4. Файл ordersDAO.cs:

public class ordersDAO

{

public List<Orders> orders = new List<Orders>();

public List<Orders> getALLOrders()

{

List<Orders> orders = new List<Orders>();

MySqlConnection connection = new MySqlConnection(DBConfig.LoadConfig());

connection.Open();

MySqlCommand command = new MySqlCommand("select \* from `order`;", connection);

using (MySqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Orders o = new Orders

{

order\_id = reader.GetInt32(0),

client\_id = reader.GetInt32(1),

worker\_id = reader.GetInt32(2),

order\_date = reader.GetDateTime(3),

payment\_amount = reader.IsDBNull(4) ? (decimal?)null : reader.GetDecimal(4),

payment\_date = reader.IsDBNull(5) ? (DateTime?)null : reader.GetDateTime(5),

};

orders.Add(o);

}

}

connection.Close();

return orders;

}

internal int addOrder(Orders order)

{

MySqlConnection connection = new MySqlConnection(DBConfig.LoadConfig());

connection.Open();

MySqlCommand command = new MySqlCommand();

command.CommandText = "insert into `order` (client\_id, worker\_id, order\_date) values(@client\_id, @worker\_id, @order\_date);";

command.Parameters.AddWithValue("@client\_id", order.client\_id);

command.Parameters.AddWithValue("@worker\_id", order.worker\_id);

command.Parameters.AddWithValue("@order\_date", order.order\_date);

command.Connection = connection;

int result = command.ExecuteNonQuery();

connection.Close();

return result;

}

internal int deleteOrder(Orders order)

{

MySqlConnection connection = new MySqlConnection(DBConfig.LoadConfig());

connection.Open();

MySqlCommand command = new MySqlCommand();

command.CommandText = "delete from `order` where (order\_id = @order\_id);";

command.Parameters.AddWithValue("@order\_id", order.order\_id);

command.Connection = connection;

int result = command.ExecuteNonQuery();

connection.Close();

return result;

}

internal int updateOrder(Orders order)

{

MySqlConnection connection = new MySqlConnection(DBConfig.LoadConfig());

connection.Open();

MySqlCommand command = new MySqlCommand();

command.CommandText = "update `order` set client\_id = @client\_id, worker\_id = @worker\_id, payment\_date = @payment\_date WHERE (order\_id = @order\_id);";

command.Parameters.AddWithValue("@order\_id", order.order\_id);

command.Parameters.AddWithValue("@client\_id", order.client\_id);

command.Parameters.AddWithValue("@worker\_id", order.worker\_id);

command.Parameters.AddWithValue("@payment\_date", order.payment\_date);

command.Connection = connection;

int result = command.ExecuteNonQuery();

connection.Close();

return result;

}

public List<Orders> getSearchOrders(Orders order)

{

List<Orders> orders = new List<Orders>();

MySqlConnection connection = new MySqlConnection(DBConfig.LoadConfig());

connection.Open();

MySqlCommand command = new MySqlCommand();

command.CommandText = "select \* from `order` where client\_id = @client\_id;";

command.Parameters.AddWithValue("@client\_id", order.client\_id);

command.Connection = connection;

using (MySqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Orders o = new Orders

{

order\_id = reader.GetInt32(0),

client\_id = reader.GetInt32(1),

worker\_id = reader.GetInt32(2),

order\_date = reader.GetDateTime(3),

payment\_amount = reader.IsDBNull(4) ? (decimal?)null : reader.GetDecimal(4),

payment\_date = reader.IsDBNull(5) ? (DateTime?)null : reader.GetDateTime(5),

};

orders.Add(o);

}

}

connection.Close();

return orders;

}

public int getLastOrderId()

{

int lastId = -1;

using (MySqlConnection connection = new MySqlConnection(DBConfig.LoadConfig()))

{

connection.Open();

string query = "SELECT MAX(order\_id) FROM `order`";

using (MySqlCommand command = new MySqlCommand(query, connection))

{

var result = command.ExecuteScalar();

if (result != DBNull.Value)

{

lastId = Convert.ToInt32(result);

}

}

}

return lastId;

}

}

4. Файл ThemeStyles.cs:

using System;

using System.Drawing;

using System.Windows.Forms;

namespace program\_bicycle\_repair\_shop\_polynko\_kn\_23\_1

{

public static class ThemeStyles

{

public static bool isDarkTheme = false;

private static readonly Color LightBackground = Color.LightGray;

private static readonly Color LightTextColor = Color.Orange;

private static readonly Color LightButtonColor = Color.WhiteSmoke;

private static readonly Color LightButtonTextColor = Color.Orange;

private static readonly Color LightMenuBackground = Color.White;

private static readonly Color LightMenuTextColor = Color.Orange;

private static readonly Color LightTextBoxColor = Color.White;

private static readonly Color LightTextBoxTextColor = Color.Orange;

private static readonly Color LightLabelTextColor = Color.Orange;

private static readonly Color LightComboBoxTextColor = Color.Orange;

private static readonly Color LightComboBoxColor = Color.White;

private static readonly Color LightGroupBoxTextColor = Color.Orange;

private static readonly Color LightDataGridViewBackColor = Color.White;

private static readonly Color LightDataGridViewTextColor = Color.Orange;

private static readonly Color LightDataGridViewBackGroundColor = Color.LightGray;

private static readonly Color LightDataGridViewRowHeadersColor = Color.Gray;

private static readonly Color LightDataGridViewColumnHeadersColor = Color.Gray;

private static readonly Color LightRichTextBoxColor = Color.White;

private static readonly Color LightRichTextBoxTextColor = Color.Orange;

private static readonly Color DarkBackground = Color.FromArgb(40, 40, 40);

private static readonly Color DarkTextColor = Color.Orange;

private static readonly Color DarkButtonColor = Color.FromArgb(70, 70, 70);

private static readonly Color DarkButtonTextColor = Color.Orange;

private static readonly Color DarkMenuBackground = Color.Black;

private static readonly Color DarkMenuTextColor = Color.Orange;

private static readonly Color DarkTextBoxColor = Color.Black;

private static readonly Color DarkTextBoxTextColor = Color.Orange;

private static readonly Color DarkLabelTextColor = Color.Orange;

private static readonly Color DarkComboBoxTextColor = Color.Orange;

private static readonly Color DarkComboBoxColor = Color.Black;

private static readonly Color DarkGroupBoxTextColor = Color.Orange;

private static readonly Color DarkDataGridViewBackColor = Color.FromArgb(70, 70, 70);

private static readonly Color DarkDataGridViewTextColor = Color.Orange;

private static readonly Color DarkDataGridViewBackGroundColor = Color.FromArgb(40, 40, 40);

private static readonly Color DarkDataGridViewRowHeadersColor = Color.Black;

private static readonly Color DarkDataGridViewColumnHeadersColor = Color.Black;

private static readonly Color DarkRichTextBoxColor = Color.Black;

private static readonly Color DarkRichTextBoxTextColor = Color.Orange;

private static readonly Font BoldFont = new Font("Microsoft Sans Serif", 8, FontStyle.Bold);

public static void ApplyTheme(Control parent)

{

Color backgroundColor = isDarkTheme ? DarkBackground : LightBackground;

Color textColor = isDarkTheme ? DarkTextColor : LightTextColor;

Color buttonColor = isDarkTheme ? DarkButtonColor : LightButtonColor;

Color buttonTextColor = isDarkTheme ? DarkButtonTextColor : LightButtonTextColor;

Color menuBackground = isDarkTheme ? DarkMenuBackground : LightMenuBackground;

Color menuTextColor = isDarkTheme ? DarkMenuTextColor : LightMenuTextColor;

Color textBoxColor = isDarkTheme ? DarkTextBoxColor : LightTextBoxColor;

Color textBoxTextColor = isDarkTheme ? DarkTextBoxTextColor : LightTextBoxTextColor;

Color labelTextColor = isDarkTheme ? DarkLabelTextColor : LightLabelTextColor;

Color comboboxTextColor = isDarkTheme ? DarkComboBoxTextColor : LightComboBoxTextColor;

Color comboboxColor = isDarkTheme ? DarkComboBoxColor : LightComboBoxColor;

Color groupBoxTextColor = isDarkTheme ? DarkGroupBoxTextColor : LightGroupBoxTextColor;

Color dataGridViewBackColor = isDarkTheme ? DarkDataGridViewBackColor : LightDataGridViewBackColor;

Color dataGridViewTextColor = isDarkTheme ? DarkDataGridViewTextColor : LightDataGridViewTextColor;

Color dataGridViewBackGroundColor = isDarkTheme ? DarkDataGridViewBackGroundColor : LightDataGridViewBackGroundColor;

Color dataGridViewRowHeadersColor = isDarkTheme ? DarkDataGridViewRowHeadersColor : LightDataGridViewRowHeadersColor;

Color dataGridViewColumnHeadersColor = isDarkTheme ? DarkDataGridViewColumnHeadersColor : LightDataGridViewColumnHeadersColor;

Color richTextBoxColor = isDarkTheme ? DarkRichTextBoxColor : LightRichTextBoxColor;

Color richTextBoxTextColor = isDarkTheme ? DarkRichTextBoxTextColor : LightRichTextBoxTextColor;

parent.BackColor = backgroundColor;

foreach (Control control in parent.Controls)

{

if (control is Button)

{

control.BackColor = buttonColor;

control.ForeColor = buttonTextColor;

control.Font = BoldFont;

}

else if (control is TextBox)

{

control.BackColor = textBoxColor;

control.ForeColor = textBoxTextColor;

control.Font = BoldFont;

}

else if (control is RichTextBox)

{

control.BackColor = richTextBoxColor;

control.ForeColor = richTextBoxTextColor;

control.Font = BoldFont;

}

else if (control is ComboBox)

{

control.BackColor = comboboxColor;

control.ForeColor = comboboxTextColor;

control.Font = BoldFont;

}

else if (control is GroupBox)

{

control.ForeColor = groupBoxTextColor;

control.Font = BoldFont;

}

else if (control is DataGridView dataGridView)

{

dataGridView.DefaultCellStyle.BackColor = dataGridViewBackColor;

dataGridView.ForeColor = dataGridViewTextColor;

dataGridView.Font = BoldFont;

dataGridView.BackgroundColor = dataGridViewBackGroundColor;

dataGridView.RowHeadersDefaultCellStyle.BackColor = dataGridViewRowHeadersColor;

dataGridView.ColumnHeadersDefaultCellStyle.BackColor = dataGridViewColumnHeadersColor;

dataGridView.DefaultCellStyle.ForeColor = dataGridViewTextColor;

dataGridView.ColumnHeadersDefaultCellStyle.ForeColor = dataGridViewTextColor;

}

else if (control is Label)

{

control.ForeColor = labelTextColor;

control.Font = BoldFont;

}

else if (control is MenuStrip menuStrip)

{

ApplyMenuStripStyle(menuStrip, menuBackground, menuTextColor);

}

else

{

control.ForeColor = textColor;

control.BackColor = backgroundColor;

}

if (parent is Form form)

{

form.StartPosition = FormStartPosition.Manual;

form.Location = new Point(

(Screen.PrimaryScreen.WorkingArea.Width - form.Width) / 2,

(Screen.PrimaryScreen.WorkingArea.Height - form.Height) / 2

);

}

if (control.HasChildren)

ApplyTheme(control);

}

}

private static void ApplyMenuStripStyle(MenuStrip menuStrip, Color backgroundColor, Color textColor)

{

menuStrip.BackColor = backgroundColor;

menuStrip.ForeColor = textColor;

menuStrip.Font = BoldFont;

foreach (ToolStripMenuItem menuItem in menuStrip.Items)

{

ApplyMenuItemStyle(menuItem, backgroundColor, textColor);

}

}

private static void ApplyMenuItemStyle(ToolStripMenuItem menuItem, Color backgroundColor, Color textColor)

{

menuItem.BackColor = backgroundColor;

menuItem.ForeColor = textColor;

menuItem.Font = BoldFont;

foreach (ToolStripItem subItem in menuItem.DropDownItems)

{

if (subItem is ToolStripMenuItem subMenuItem)

{

subMenuItem.BackColor = backgroundColor;

subMenuItem.ForeColor = textColor;

subMenuItem.Font = BoldFont;

}

}

}

public static void ToggleTheme(Form form)

{

isDarkTheme = !isDarkTheme;

ApplyTheme(form);

}

public static void Change\_theme\_img(PictureBox change\_theme)

{

change\_theme.Image = ThemeStyles.isDarkTheme

? Properties.Resources.to\_white\_theme

: Properties.Resources.to\_dark\_theme;

}

}

}

5. Файл FormOrders.cs:

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace program\_bicycle\_repair\_shop\_polynko\_kn\_23\_1

{

public partial class FormOrders : Form

{

BindingSource bindingSource = new BindingSource();

public FormOrders()

{

InitializeComponent();

ThemeStyles.ApplyTheme(this);

ThemeStyles.Change\_theme\_img(change\_theme);

dataGridView1.Left = (dataGridView1.Width) / 8;

lblResult.Text = "";

cbPayment.Enabled = false;

btnAdd.Enabled = true;

}

private void change\_theme\_Click(object sender, EventArgs e)

{

change\_theme.Image = ThemeStyles.isDarkTheme

? Properties.Resources.to\_dark\_theme

: Properties.Resources.to\_white\_theme;

ThemeStyles.ToggleTheme(this);

}

private void FormOrders\_Load(object sender, EventArgs e)

{

LoadClients();

LoadWorkers();

ordersDAO oDAO = new ordersDAO();

bindingSource.DataSource = oDAO.getALLOrders();

dataGridView1.DataSource = bindingSource;

if (dataGridView1.Rows.Count > 0)

{

dataGridView1.FirstDisplayedScrollingRowIndex = dataGridView1.Rows.Count - 1;

}

}

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

cbPayment.Enabled = true;

btnAdd.Enabled = false;

DataGridViewRow row = dataGridView1.Rows[e.RowIndex];

txtID.Text = row.Cells["order\_id"].Value.ToString();

lstClientID.SelectedValue = Convert.ToInt32(row.Cells["client\_id"].Value);

lstWorkerID.SelectedValue = Convert.ToInt32(row.Cells["worker\_id"].Value);

dtpOrderDate.Text = row.Cells["order\_date"].Value.ToString();

var paymentValue = row.Cells["payment\_date"].Value;

cbPayment.Checked = paymentValue != null && !string.IsNullOrWhiteSpace(paymentValue.ToString());

var paymentAmountValue = row.Cells["payment\_amount"].Value;

bool hasPaymentAmount = paymentAmountValue != null &&

decimal.TryParse(paymentAmountValue.ToString(), out decimal amount) &&

amount > 0;

cbPayment.Enabled = hasPaymentAmount;

}

}

private void очиститиToolStripMenuItem\_Click(object sender, EventArgs e)

{

txtID.Clear();

cbPayment.Checked = false;

cbPayment.Enabled = false;

btnAdd.Enabled = true;

dtpOrderDate.Value = DateTime.Now;

}

private void повернутисядоменюToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Close();

}

private void FormOrders\_FormClosed(object sender, FormClosedEventArgs e)

{

var menu = new Menu();

menu.Show();

}

private void btnShow\_Click(object sender, EventArgs e)

{

ordersDAO oDAO = new ordersDAO();

bindingSource.DataSource = oDAO.getALLOrders();

dataGridView1.DataSource = bindingSource;

}

private void btnAdd\_Click(object sender, EventArgs e)

{

Orders order = new Orders

{

client\_id = Convert.ToInt32(lstClientID.SelectedValue),

worker\_id = Convert.ToInt32(lstWorkerID.SelectedValue),

order\_date = dtpOrderDate.Value,

};

ordersDAO oDAO = new ordersDAO();

int result = oDAO.addOrder(order);

if (result != 1)

{

lblResult.Text = "Додавання: помилка!";

}

else

{

lblResult.Text = "Додавання: успіх!";

int newOrderId = oDAO.getLastOrderId();

var order\_detail = new FormOrderDetail(newOrderId);

order\_detail.ShowDialog();

}

bindingSource.DataSource = oDAO.getALLOrders();

dataGridView1.DataSource = bindingSource;

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

Orders order = new Orders

{

order\_id = int.Parse(txtID.Text),

client\_id = Convert.ToInt32(lstClientID.SelectedValue),

worker\_id = Convert.ToInt32(lstWorkerID.SelectedValue),

payment\_date = cbPayment.Checked ? DateTime.Now : (DateTime?)null

};

int currentOrderId = int.Parse(txtID.Text);

ordersDAO oDAO = new ordersDAO();

int result = oDAO.updateOrder(order);

var order\_detail = new FormOrderDetail(currentOrderId);

order\_detail.ShowDialog();

bindingSource.DataSource = oDAO.getALLOrders();

dataGridView1.DataSource = bindingSource;

if (result != 1)

{

lblResult.Text = "Оновлення: помилка!";

}

else

{

lblResult.Text = "Оновлення: успіх!";

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

Orders order = new Orders

{

order\_id = int.Parse(txtID.Text),

};

ordersDAO oDAO = new ordersDAO();

int result = oDAO.deleteOrder(order);

bindingSource.DataSource = oDAO.getALLOrders();

dataGridView1.DataSource = bindingSource;

if (result != 1)

{

lblResult.Text = "Видалення: помилка!";

}

else

{

lblResult.Text = "Видалення: успіх!";

}

}

private void btnSearch\_Click(object sender, EventArgs e)

{

Orders order = new Orders

{

client\_id = Convert.ToInt32(lstClientID.SelectedValue)

};

ordersDAO oDAO = new ordersDAO();

bindingSource.DataSource = oDAO.getSearchOrders(order);

dataGridView1.DataSource = bindingSource;

}

private void детальнішеПроЗамовленняToolStripMenuItem\_Click(object sender, EventArgs e)

{

var expandedOrderDetailView = new FormExpandedOrderDetailView();

expandedOrderDetailView.ShowDialog();

}

private void LoadClients()

{

using (var conn = new MySqlConnection(DBConfig.LoadConfig()))

{

conn.Open();

var cmd = new MySqlCommand("SELECT client\_id, pib FROM clients ORDER BY client\_id DESC", conn);

var adapter = new MySqlDataAdapter(cmd);

var table = new DataTable();

adapter.Fill(table);

lstClientID.DataSource = table;

lstClientID.DisplayMember = "pib";

lstClientID.ValueMember = "client\_id";

}

}

private void LoadWorkers()

{

using (var conn = new MySqlConnection(DBConfig.LoadConfig()))

{

conn.Open();

var cmd = new MySqlCommand("SELECT worker\_id, pib FROM workers ORDER BY worker\_id DESC", conn);

var adapter = new MySqlDataAdapter(cmd);

var table = new DataTable();

adapter.Fill(table);

lstWorkerID.DataSource = table;

lstWorkerID.DisplayMember = "pib";

lstWorkerID.ValueMember = "worker\_id";

}

}

}

}

**Додаток В  
Скрипти створення тригерів**

DELIMITER //

CREATE TRIGGER after\_order\_detail\_insert

AFTER INSERT ON order\_detail

FOR EACH ROW

BEGIN

DECLARE total\_amount DECIMAL(15,2);

SELECT SUM(d.price \* od.quantity\_details) INTO total\_amount

FROM order\_detail od

JOIN details d ON od.detail\_id = d.detail\_id

WHERE od.order\_id = NEW.order\_id;

UPDATE `order`

SET payment\_amount = total\_amount

WHERE order\_id = NEW.order\_id;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER after\_order\_detail\_update

AFTER UPDATE ON order\_detail

FOR EACH ROW

BEGIN

DECLARE total\_amount DECIMAL(15,2);

SELECT SUM(d.price \* od.quantity\_details) INTO total\_amount

FROM order\_detail od

JOIN details d ON od.detail\_id = d.detail\_id

WHERE od.order\_id = NEW.order\_id;

UPDATE `order`

SET payment\_amount = total\_amount

WHERE order\_id = NEW.order\_id;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER before\_order\_delete

BEFORE DELETE ON `order`

FOR EACH ROW

BEGIN

DELETE FROM order\_detail WHERE order\_id = OLD.order\_id;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER before\_order\_insert

BEFORE INSERT ON `order`

FOR EACH ROW

BEGIN

IF (SELECT position FROM workers WHERE worker\_id = NEW.worker\_id) NOT LIKE 'Manager' THEN

SET NEW.worker\_id = NULL;

END IF;

END //

DELIMITER //

CREATE TRIGGER before\_order\_update

BEFORE UPDATE ON `order`

FOR EACH ROW

BEGIN

IF (SELECT position FROM workers WHERE worker\_id = NEW.worker\_id) NOT LIKE 'Manager' THEN

SET NEW.worker\_id = NULL;

END IF;

END //

DELIMITER //

CREATE TRIGGER before\_sign\_up\_repairs\_insert

BEFORE INSERT ON sign\_up\_repairs

FOR EACH ROW

BEGIN

IF (SELECT position FROM workers WHERE worker\_id = NEW.worker\_id) NOT LIKE 'Master' THEN

SET NEW.worker\_id = NULL;

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER before\_sign\_up\_repairs\_update

BEFORE UPDATE ON sign\_up\_repairs

FOR EACH ROW

BEGIN

IF (SELECT position FROM workers WHERE worker\_id = NEW.worker\_id) NOT LIKE 'Master' THEN

SET NEW.worker\_id = NULL;

END IF;

END //

DELIMITER ;