ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**Розробка програмного забезпечення для підтримки бізнес-процесів кав'ярні Neroaroma мовою C#»**

Виконав: студент 4 курсу, групи
ПД–44 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Пилипов Ю.Ю
(прізвище та ініціали)
Керівник
Корецька В.О
(прізвище та ініціали)
Рецензент
_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерія програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Напрям підготовки -121 - «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

О.В. Негоденко

—___‖_____

2023 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Пилипов Юрій Юрійович

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення для підтримки бізнес-процесів кав'ярні Neroaroma мовою C#»

Керівник роботи к.п.н., доцент Корецька В.О.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від —‖_____2023 року №__.

2. Строк подання студентом роботи_____

3. Вхідні дані до роботи:

3.1 Організаційно-виробнича структура управління кав'ярні Neroaroma.
3.2 Модель потоків даних кав'ярні.
3.3 Мова C#.
3.4 Існуючі інструменти тестування програмного забезпечення.
3.5 Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1 Аналіз поточного стану та бізнес процесів кав’ярні Neroaroma.

4.2 Вибір технологій проектування та особливості архітектури програмного забезпечення.

4.3 Проектування бази даних додатку підтримки бізнес-процесів кав’ярні.

4.4 Програмна реалізація додатку.

4.5 Приклади використання та тестування додатку.

4.6 Висновки.

5. Перелік графічного матеріалу.

5.1. Титульний слайд.

5.2. Мета, об’єкт та предмет дослідження .

5.3. Актуальність роботи.

5.4. Аналоги.

5.5. Порівняння з аналогами .

5.6. Технічне завдання.

5.7. Програмні засоби реалізації.

5.8. Інструменти використані для реалізації.

5.9. Архітектура браузерного розширення.

5.10. Архітектура плеєра тестових сценаріїв .

5.11. Апробація результатів дослідження.

5.12. Висновки.

6. Дата видачі завдання: _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.23-26.02.23	Виконано
2	Аналіз бізнес-процесів кав'ярні	27.02.23-08.03.23	Виконано
3	Автоматизація системи управління замовленнями	09.03.23-22.03.23	Виконано
4	Архітектура проекту	23.03.23-01.04.23	Виконано
5	Проектування баз даних	02.04.23-10.04.23	Виконано
6	Надійність і безпека	11.04.23-16.04.23	Виконано
7	Функціональне та модульне тестування	17.04.23-01.05.23	Виконано
8	Вступ, висновки, реферат	02.05.23-19.05.23	Виконано
9	Попередній захист роботи	19.05.23-26.05.23	Виконано
10	Подання роботи в деканат	26.05.23-01.06.23	Виконано

Студент _____ Пилипов Ю.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Корецька В.О.
(підпис) (прізвище та ініціали)

Реферат

Текстова частина бакалаврської роботи 92 с., 6 табл., 42 рис., 28 джерел
Об'єкт дослідження - бізнес-процеси кав'ярні.

Предмет дослідження - програмне забезпечення для підтримки бізнес процесів кав'ярні.
Мета роботи - оптимізація бізнес процесів роботи кав'ярні за рахунок автоматизації системи управління замовленнями з використанням додатку розробленого мовою C#.
Методи дослідження - методи побудови POS систем, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування додатку.
Для досягнення мети необхідно виконати наступні завдання:

1. Провести аналіз бізнес-процесів кав'ярні для визначення існуючих проблем, вимог та можливостей для їх оптимізації.
2. Проаналізувати існуючі програмні рішення для автоматизації кав'ярні.
3. Сформувати вимоги до програмного забезпечення.
4. Спроектувати архітектуру та розробити додаток для кав'ярні.
5. Провести тестування розробленого програмного забезпечення.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI - user interface

BLL - Business Logic Layer

DAL - Data Access Layer

POS - Point of sale

ВСТУП

Сучасний розвиток технологій впливає на різні сфери життя, зокрема на бізнес. Автоматизація бізнес-процесів дозволяє підвищити ефективність підприємства і покращити обслуговування клієнтів. Особливо актуальна автоматизація стає в сфері HoReCa, в якій важлива швидкість та якість обслуговування [1]. Така тенденція ставить на порядок денний задачу розробки програмного забезпечення для підтримки бізнес-процесів в таких закладах, як кав'ярні.

Тема даної роботи актуальна у контексті потреби оптимізації та ефективної організації роботи кав'ярні. Сучасні рішення часто є загальними та не враховують специфіку окремого закладу, що суттєво знижує їх ефективність. Така ситуація свідчить про наявність проблеми та потреби у спеціалізованому рішенні для кав'ярні Neroaroma.

Робота є частиною науково-дослідницької роботи в області інформаційних технологій та систем, спрямованої на розробку високоефективних систем автоматизації бізнес-процесів. Ця тема також входить до плану наукових досліджень відділу програмного забезпечення та автоматизації бізнес-процесів нашого університету.

Розробка програмного забезпечення для кав'ярні Neroaroma спрямована на підвищення якості обслуговування клієнтів та ефективності використання ресурсів закладу. Вона є важливою не лише для Neroaroma, а й для всієї сфери HoReCa в Україні, оскільки створює базу для подальшого впровадження аналогічних рішень в інших закладах цього сектору. Таким чином, дана робота має велике практичне значення і перспективи для подальшого використання.

Метою даної роботи є оптимізація бізнес процесів роботи кав’ярні за рахунок автоматизації системи управління замовленнями з використаннями додатку розробленого мовою C#, яке дозволить оптимізувати бізнес-процеси,

підвищити якість обслуговування клієнтів та покращити загальну ефективність роботи закладу.

Для досягнення цієї мети будуть виконані наступні задачі:

– аналіз потреб і специфіки роботи кав'ярні Neroaroma для визначення основних вимог до програмного забезпечення;

– розробка концепції програмного забезпечення, що включає структуру, основні модулі та функції системи;

– реалізація програмного забезпечення на мові програмування C#, та забезпечення його тестування;

– підготовка документації по використанню та подальшій підтримці програмного забезпечення.

Об'єктом дослідження є бізнес-процеси кав'ярні Neroaroma. Вони включають в себе організацію роботи закладу, обслуговування клієнтів, управління ресурсами, контроль якості продукції та інші процеси, що відбуваються в кав'ярні на щоденній основі.

Предметом дослідження є розробка програмного забезпечення на мові програмування C#, яке спрямоване на автоматизацію та оптимізацію бізнес-процесів кав'ярні. Це включає в себе аналіз потреб та вимог до системи, вивчення сучасних технологій і практик розробки ПЗ, розробку архітектури та дизайну програмного забезпечення, його реалізацію та тестування, а також підготовку документації та підтримку системи.

Новизною цієї роботи є специфічний підхід до автоматизації та оптимізації бізнес-процесів кав'ярні Neroaroma через розробку власного програмного забезпечення, яке буде спеціально адаптоване під специфіку даного закладу.

1 АНАЛІЗ ПОТОЧНОГО СТАНУ РЕСТОРАННОГО БІЗНЕСУ

1.1 Аналіз бізнес-процесів кав‘ярні Neroaroma

Компанія «Neroaroma» є товариством з обмеженою відповідальністю, що має приватну форму власності та діє в рамках спрощеної системи оподаткування. Згідно з встановленими правилами та законодавством, які визначають функціонування такого виду підприємств, товариство з обмеженою відповідальністю має ряд особливостей:

– учасники товариства з обмеженою відповідальністю, відповідно до правового статусу, не несуть особистої відповідальності за зобов'язаннями підприємства і ризик збитків обмежується сумою їх внесків у статутний капітал. Це означає, що учасники несуть фінансові ризики тільки в межах своїх внесків, і їх особисті майнові ресурси залишаються захищеними від можливих збитків, пов'язаних з діяльністю підприємства;

– товариство з обмеженою відповідальністю не має обмежень вибору, якщо ці види діяльності не заборонені законодавством. Це означає, що «Neroaroma» може обрати будь-яку галузь діяльності, яка відповідає вимогам закону та регулюється відповідними нормативними актами.

Завдяки своєму статусу, кав’ярня також має можливість користуватися спрощеною системою оподаткування, яка надає певні пільги та спрощені процедури в плані сплати податків і звітності. Це сприяє зменшенню адміністративних та фінансових навантажень для підприємства та сприяє ефективному веденню бізнесу.

Кав'ярня Neroaroma працює в рамках приватної форми власності, що передбачає, що підприємство є власністю окремого громадянина України та має право наймати робочу силу. Приватна форма власності відображає те, що засоби виробництва та вироблені продукти належать приватним особам, які мають ексклюзивне право на володіння, користування та розпорядження об'єктами власності, будь то юридичні або фізичні особи.

Використання приватної форми власності дозволяє кав'ярні Neroaroma раціонально розподіляти ресурси та забезпечує більшу гнучкість у здійсненні певних дій, що сприяє підвищенню ефективності діяльності порівняно з іншими формами власності. Приватний власник може самостійно приймати рішення та використовувати свої ресурси з урахуванням власних інтересів і стратегії розвитку кав'ярні. Крім того, приватна форма власності сприяє створенню конкурентних умов, інноваційному підходу та адаптації до змін на ринку.

Це відмінні риси приватної форми власності, які відповідають специфіці та особливостям кав'ярні Neroaroma, дозволяючи їй ефективно управляти ресурсами та розвиватися на основі власних стратегічних рішень.

В ході аналізу предметної області кав'ярні Neroaroma було встановлено, що це підприємство громадського харчування, основними продуктами меню якого є кава і чай, супроводжувані кондитерськими виробами, закусками, коктейлями, прохолодними та алкогольними напоями. Однак, бізнес-концепція кав'ярні полягає не лише (і не стільки) у наданні послуг громадського харчування, а й у задоволенні потреб відвідувачів у проведенні дозвілля, короткого відпочинку, забезпеченні місця для спілкування та ділових зустрічей.

Отже, кав'ярня Neroaroma є місцем, що поєднує функції громадського харчування та дозвілля. Кожен відвідувач може визначити власну мету перебування в кав'ярні залежно від своїх потреб.

Концепція кав'ярні відповідає традиційному формату з елементами "фаст-фуду". Кав'ярня пропонує самообслуговування, використовує одноразові стаканчики та спеціалізується на каві "на винос".

Кав'ярня Neroaroma має "демократичну" концепцію, орієнтовану на широку аудиторію відвідувачів. Основними цільовими групами є клієнти, які

прагнуть скористатися послугами кав'ярні та провести час у приємній атмосфері.

Основні фактори успіху бізнесу кав'ярні Neroaroma, які варто враховувати, включають:

– місце розташування. Відповідне розташування кав'ярні є ключовим фактором успіху [2]. Відповідний вибір місця, де знаходиться кав'ярня, може забезпечити високий потік потенційних клієнтів, наприклад, в центральній частині міста, біля офісних комплексів або популярних туристичних визначних місць;

– склад меню і якість страв і напоїв. Ключовою складовою кав'ярні є якість їжі та напоїв, зокрема високоякісна кава [3]. Розробка різноманітного і привабливого меню, яке задовольняє смакові уподобання різних клієнтів, є важливою. Постійне покращення якості продукції та дотримання стандартів готування сприяє задоволенню клієнтів і формуванню позитивної репутації кав'ярні;

– внутрішнє і зовнішнє оформлення, зручність інтер'єру [4]. Кав'ярня повинна мати привабливий та затишний дизайн, що сприяє комфортній атмосфері для клієнтів. Зручність меблів, правильне освітлення, естетичний дизайн та використання приємних кольорів сприяють створенню приємного враження і залученню клієнтів;

– рівень сервісу. Якість обслуговування є ключовим фактором в задоволенні потреб клієнтів [5]. Швидка обробка замовлень, ввічливість персоналу, дотримання заздалегідь визначених строків та якісний післяпродажний сервіс сприяють створенню позитивного досвіду для клієнтів;

– якість управління бізнесом. Ефективне управління бізнесом включає постійний моніторинг фінансових показників, контроль за запасами, управління персоналом та маркетингові стратегії. Регулярність, постійність

та професійний підхід у управлінні допомагають підтримувати ефективну роботу кав'ярні.

Організаційно-виробнича структура управління кав'ярні Neroaroma обрана у формі лінійної структури (рис. 1.1). Це означає, що в кав'ярні існує єдиноначальник, підпорядкованість та чітке розподілення повноважень і обов'язків. Така структура забезпечує ясність у комунікації, виконання встановлених правил та дисципліну. Проте, вимагає від керівника значних компетенцій для ефективного управління різними аспектами бізнесу.

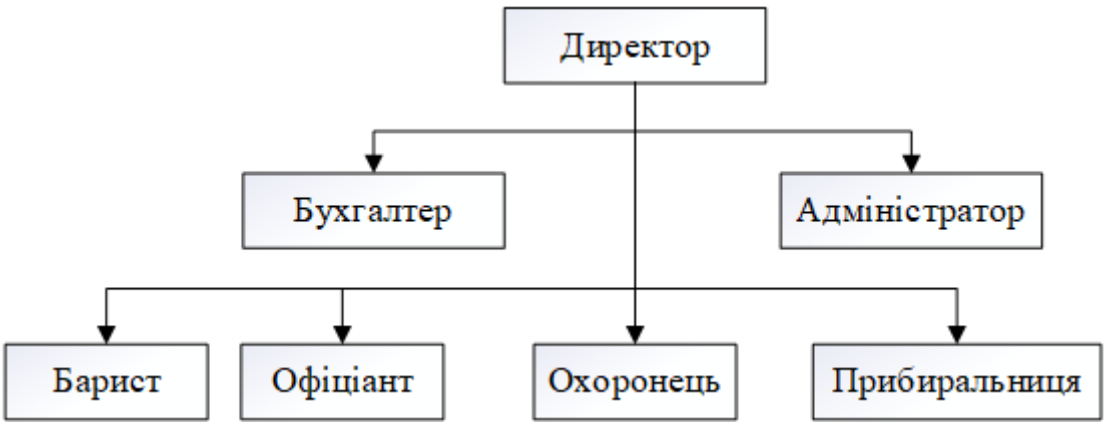

Рисунок 1.1 – Організаційна структура кав'ярні Neroaroma

Організаційна структура кав'ярні Neroaroma ґрунтується на ієрархічному принципі рівнів управління, де кожен нижчий рівень підпорядковується та контролюється вищим рівнем. Така структура забезпечує ефективну координацію та керівництво різними аспектами діяльності кав'ярні. Крім того, вона ґрунтується на принципі поділу праці на окремі функції та спеціалізації працівників, що дозволяє ефективно виконувати різноманітні завдання та забезпечує високу якість виконання роботи.

Результати дослідження діяльності кав'ярні Neroaroma дозволили визначити функціональні обов'язки його працівників наступним чином:

– директор. Він відповідає за загальну організацію роботи та контроль над всім процесом у кав'ярні. Директор виступає як

матеріально-відповідальна особа, діє від імені підприємства і представляє його інтереси у всіх інстанціях та перед постачальниками. Він укладає договори, відкриває розрахунковий рахунок у банку, видає доручення та організовує всю роботу кав'ярні. Директор несе повну відповідальність за стан підприємства, його працівників та майна.

– бухгалтер. Його відповідальність включає операції з прийому, обліку, видачі та зберігання грошових коштів в кав'ярні. Бухгалтер також виступає як матеріально-відповідальна особа і стежить за правильним веденням документації. Він організовує та виконує бухгалтерський облік, контролює достовірність інформації та дотримання законності витрат грошових і матеріальних ресурсів;

– адміністратор. Ця посада є матеріально-відповідальною особою, відповідальною за прийом сировини та оформлення необхідних документів. Адміністратор також організовує роздрібну торгівлю в кав'ярні;

– бариста. Це спеціаліст, відповідальний за приготування та подачу кавових напоїв в кав'ярні Neroaroma. Бариста має глибокі знання про різноманітні види кави, методи її приготування та майстерно володіє навичками виготовлення красивих латте-артів;

– офіціант. Офіціант виконує функцію обслуговування відвідувачів кав'ярні Neroaroma. Він приймає замовлення, подає їжу та напої, дотримується стандартів сервісу і забезпечує комфортну атмосферу для клієнтів.

Основні вимоги, що пред'являються до офіціантів і барменів у кав'ярні Neroaroma, включають комунікабельність, коректність та приємну зовнішність. Ці вимоги ставляться з метою забезпечення якісного обслуговування клієнтів та створення приємної атмосфери в кав'ярні.

У контексті функціональних обов'язків працівників кав'ярні, модель потоків даних є основним інструментом для моделювання функціональних

вимог до системи. Ця модель відображає ієрархію функціональних компонентів або процесів, які взаємодіють через потоки даних.

Джерела інформації, такі як постачальники і клієнти, генерують інформаційні потоки, наприклад, накладні та замовлення, які передають інформацію до різних підсистем або процесів. Ці підсистеми або процеси обробляють інформацію та генерують нові потоки даних, які можуть бути передані іншим процесам, підсистемам, накопичувачам даних або зовнішнім сутностям, таким як споживачі інформації.

Головна мета побудови моделі потоків даних полягає в тому, щоб зробити вимоги до системи зрозумілими на різних рівнях деталізації та розбити їх на частини з чітко визначеними відносинами між ними.

На моделі, що представлена на рис. 1.2, показана функціональна залежність між учасниками автоматизованої системи управління кав'ярні Neroaroma.

Рисунок 1.2 – Модель потоків даних кав'ярні Neroaroma

Нижній рівень управління в кав'ярні Neroaroma виступає джерелом інформації для прийняття управлінських рішень на вищих рівнях. Потік інформації від нижнього рівня до верхнього характеризується зменшенням обсягу інформації, вираженого у символах, але збільшенням смислового змісту.

Автоматизована система управління кав'ярнею повинна містити базу даних звітів, які складають співробітники підприємства:

– бариста складатиме звіти щодо продажу кавових напоїв;
– адміністратор подаватиме звіти, які міститимуть інформацію про накладні постачальників та загальні витрати підприємства;
– бухгалтер формуватиме загальні щомісячні фінансові звіти;
– система також буде обраховувати прибуток та витрати підприємства.

Розробка автоматизованої системи управління кав'ярнею, заснована на даній моделі повинна спростити рутинні операції та полегшити роботу персоналу.

1.2 Огляд існуючих рішень

На сьогоднішній день існує широкий спектр програмного забезпечення, спрямованого на автоматизацію та підтримку бізнес-процесів у галузі громадського харчування. У контексті розробки програмного забезпечення для підтримки бізнес-процесів кав'ярні Neroaroma, важливо розглянути деякі існуючі рішення, які можуть бути використані для оптимізації її функціонування.

Застосунок Toast

Toast є повнофункціональним програмним застосунком POS (точка продажу), спеціально розробленим для галузі громадського харчування, включаючи кав'ярні [6]. Основна мета застосунку Toast - надати інструменти

для ефективного керування замовленнями, оплатою та управлінням операціями кав'ярні.

Основне призначення застосунку Toast полягає в автоматизації різних бізнес-процесів кав'ярні для поліпшення ефективності та забезпечення кращого обслуговування клієнтів. Він допомагає спростити процеси замовлення та оплати, керувати запасами, контролювати продажі та надавати аналітику для прийняття управлінських рішень.

Основна архітектура застосунку Toast базується на хмарних технологіях, що дозволяє йому працювати стабільно та масштабуватись в залежності від потреб кав'ярні. Він має інтуїтивний інтерфейс, який легко використовувати персоналом кав'ярні (рис. 1.3).

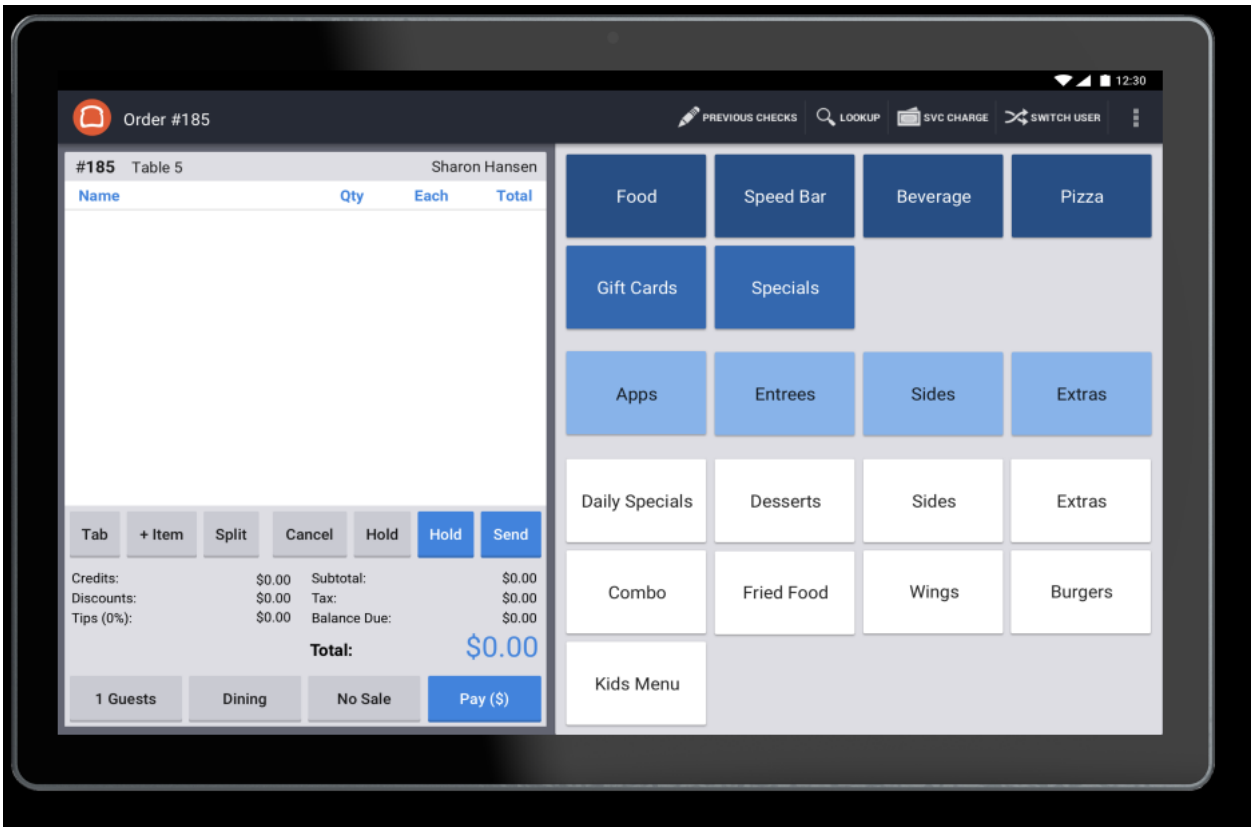

Рисунок 1.3 – Інтерфейс користувача застосунку «Toast»

Деякі основні функції застосунку Toast включають [7]:

– прийом замовлень. Дозволяє персоналу кав'ярні легко приймати замовлення вручну або за допомогою мобільних пристроїв, спрощуючи процес обслуговування клієнтів;

– управління запасами. Дозволяє вести облік та контроль за складом товарів, сприяючи ефективному плануванню замовлень і запобіганню дефіциту товарів;

– звіти про продажі. Надає детальну інформацію про продажі, фінансову продуктивність та інші ключові показники, що дозволяє аналізувати результативність кав'ярні та приймати відповідні рішення;

– інтеграція з онлайн-замовленнями. Забезпечує можливість приймати та обробляти замовлення онлайн, спрощуючи процес доставки та забезпечуючи зручність для клієнтів.

Переваги застосунку Toast для кав'ярні Neroaroma:

– функціональність. Toast є повнофункціональним POS-застосунком, який надає широкі можливості для керування замовленнями, оплатою, запасами та аналітикою;

– інтуїтивний інтерфейс. Має зручний інтерфейс, який легко вивчити і використовувати персоналу кав'ярні. Це дозволяє зменшити час, необхідний для навчання персоналу і забезпечує швидку роботу з програмою;

– гнучкість і масштабованість. Застосунок побудований на хмарних технологіях, що дозволяє легко масштабувати його функціональність та відповідати зростаючим потребам кав'ярні [8];

– інтеграція з іншими системами. Підтримує інтеграцію з різними зовнішніми системами, такими як платіжні системи, програми лояльності та онлайн-замовлення.

Недоліки застосунку Toast для кав'ярні Neroaroma:

– вартість. Є комерційним продуктом, що може вимагати певних інвестицій у придбання ліцензій та обслуговування. Вартість використання

застосунку може бути вищою порівняно з деякими іншими рішеннями на ринку;

– залежність від Інтернету. Базується на хмарних технологіях, що означає, що для його коректної роботи потрібне стабільне підключення до Інтернету. Відсутність зв'язку може призвести до недоступності деяких функцій або затримок у роботі.

Отже, застосунок Toast є повнофункціональним POS-застосунком, який надає широкі можливості для керування бізнес-процесами в кав'ярні Neroaroma. Він має інтуїтивний інтерфейс, забезпечує гнучкість та масштабованість, а також можливість інтеграції з іншими системами. Проте, використання Toast вимагає вартості та залежить від Інтернет-підключення. Необхідне навчання персоналу та операційна залежність також є важливими факторами для врахування. Загалом, перед прийняттям рішення щодо використання Toast, слід уважно розглянути його переваги та недоліки в контексті потреб та вимог кав'ярні.

Square for Restaurants

Застосунок Square for Restaurants є спеціалізованим POS-застосунком, розробленим для використання в ресторанному бізнесі [9]. Основне призначення цього застосунку - надання зручних і ефективних інструментів для керування замовленнями, оплатою та управлінням ресторанними операціями.

На рис. 1.4 зображено інтерфейс користувача для роботи із Square for Restaurants.

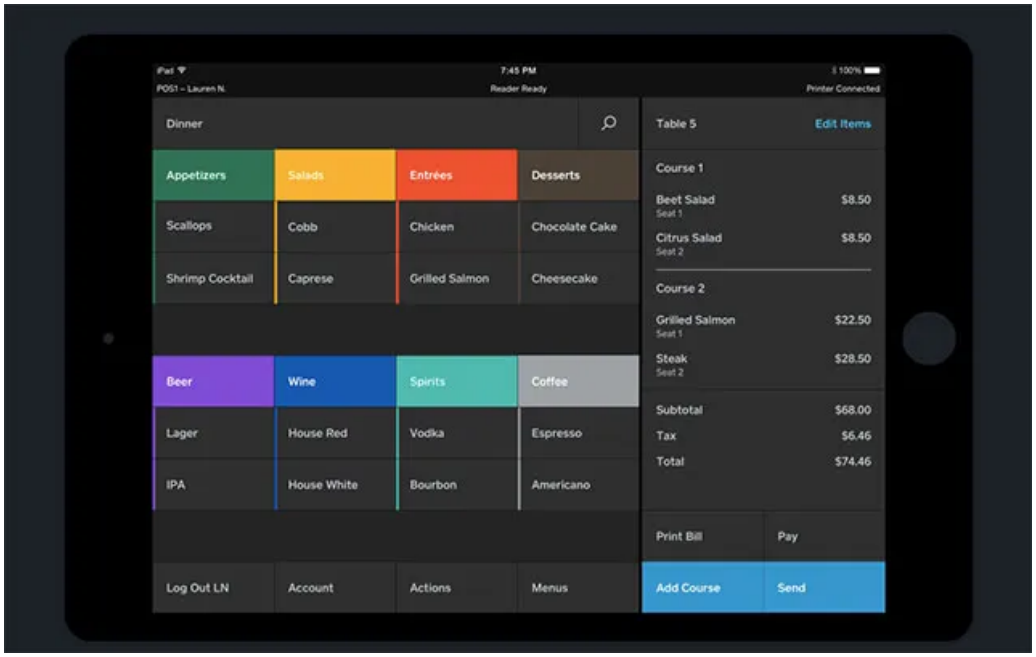

Рисунок 1.4 – Інтерфейс користувача застосунку «Square for Restaurants»

Основні архітектурні компоненти Square for Restaurants включають [10]:

– POS-термінал. Застосунок надає зручний інтерфейс POS-терміналу для обробки замовлень, включаючи створення, зміну та оплату замовлень, додавання дисконтів і чайових, а також роздруківку чеків для клієнтів;

– керування столиками. Можна легко керувати столиками в ресторані, резервувати їх, призначати замовлення до певних столиків і відстежувати їх статус;

– управління замовленнями. Дозволяє ефективно керувати замовленнями, включаючи розділення рахунків, додавання товарів і послуг до замовлення, зміну складу замовлення та відслідковування статусу замовлення;

– звітність і аналітика. Надає можливість отримувати детальну звітність і аналітику щодо продажів, запасів, прибутку та інших ресторанних показників. Це допомагає аналізувати продуктивність бізнесу та приймати обґрунтовані управлінські рішення.

Переваги Square for Restaurants:

– інтуїтивний і простий у використанні. Має зручний та легкий у використанні інтерфейс, що спрощує навчання персоналу та забезпечує ефективну роботу;

– широкий функціонал. Застосунок пропонує різноманітні функції, включаючи керування замовленнями, оплатою, столиками, запасами, звітністю та аналітикою, що дозволяє ресторанам повністю контролювати свої бізнес-процеси [11];

– інтеграція з іншими системами. Може бути легко інтегрований з іншими системами, такими як бухгалтерський облік або система управління запасами;

– мобільність. Працює на різних пристроях, включаючи смартфони та планшети, що дає змогу персоналу зручно працювати на майданчику та забезпечує більшу мобільність в обслуговуванні клієнтів.

Недоліки Square for Restaurants:

– вартість. Використання вимагає певних витрат на оплату послуг та обслуговування. Вартість може варіюватися залежно від розміру ресторану та обсягу функцій, які використовуються;

– залежність від Інтернету. Вимагає стабільного підключення до Інтернету для оптимальної роботи. Проблеми зі зв'язком можуть призвести до перебоїв у роботі та недоступності деяких функцій;

– обмежене налаштування. Індивідуальні налаштування є обмеженими, що не враховує специфічні потреби окремих ресторанів. Це може створити обмеження для деяких бізнес-процесів або потребувати додаткового зусилля для пристосування.

Отже, Square for Restaurants - потужна POS-система, спеціально розроблена для ресторанного бізнесу, з метою оптимізації управління кав'ярнею Neroaroma. Вона пропонує широкий функціонал, включаючи керування замовленнями, оплатою, столиками та звітністю, і має інтуїтивний

інтерфейс для легкого використання. Square for Restaurants допомагає підвищити ефективність бізнес-процесів, забезпечує аналітику та спрощує керування ресторанним бізнесом.

Lightspeed Restaurant

Lightspeed Restaurant - це POS-система, спеціально розроблена для ресторанного бізнесу, з метою полегшення управління кав'ярнею Neroaroma та оптимізації бізнес-процесів [12].

Головне призначення Lightspeed Restaurant полягає в керуванні замовленнями, оплатою, столиками, запасами, звітністю та інвентарем. Він надає рестораторам широкий функціонал для контролю і оптимізації всіх аспектів їхнього бізнесу.

Основна архітектура Lightspeed Restaurant базується на хмарних технологіях, що дозволяє забезпечити стабільну та безперебійну роботу системи. Вона включає в себе мобільний додаток для замовлення та оплати, POS-термінали для обслуговування клієнтів на місці та веб-панель управління для контролю над всіма аспектами ресторанного бізнесу.

Всі ці функції дозволяють кав'ярні Neroaroma легко керувати замовленнями, ефективно виконувати операції з оплати, управляти запасами та забезпечувати звітність для аналізу та прийняття стратегічних рішень.

Інтерфейс користувача системи «Lightspeed Restaurant» представлено на рис. 1.5.

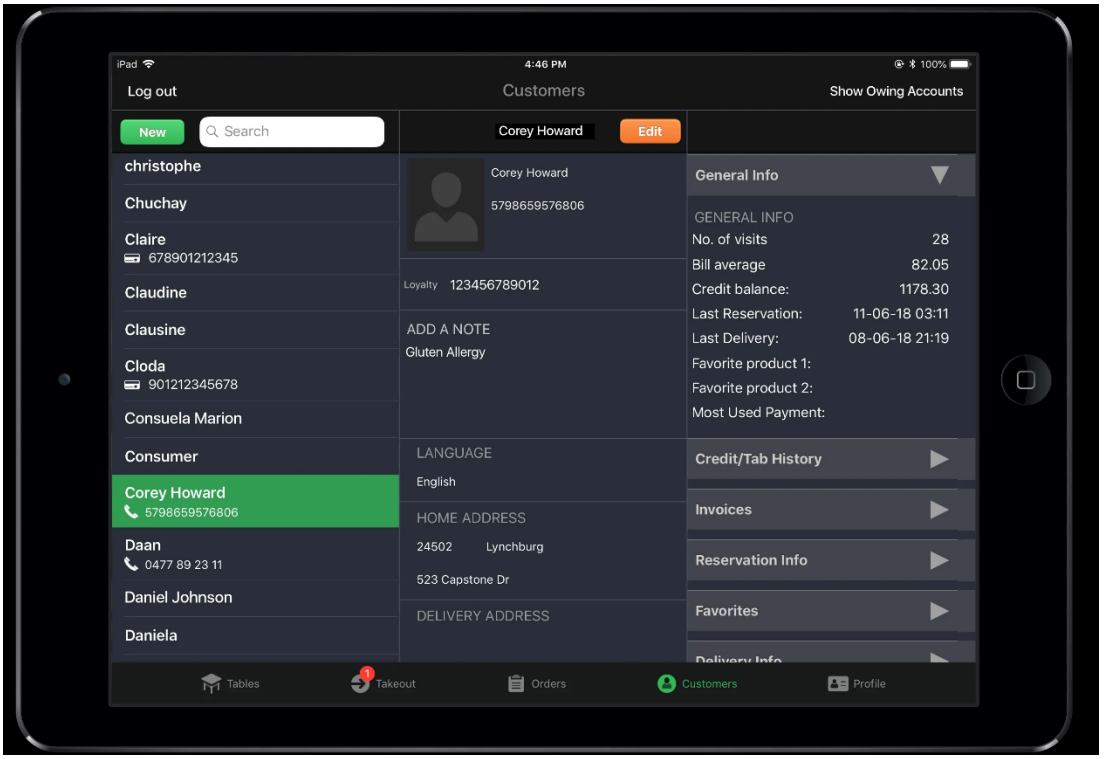

Рисунок 1.4 – Інтерфейс користувача системи «Lightspeed Restaurant»

Переваги Lightspeed Restaurant [13]:

– широкий функціонал. Пропонує різноманітні функції для ефективного управління кав'ярнею, включаючи керування замовленнями, оплатою, столиками, запасами та звітністю;

– гнучкість та мобільність. Система працює на мобільних пристроях, що дозволяє персоналу приймати замовлення та здійснювати оплату безпосередньо біля столиків або в будь-якому місці в кав'ярні;

– хмарна архітектура. Використання хмарних технологій забезпечує стабільну та безперебійну роботу системи, а також зручний доступ до даних та звітів з будь-якого місця.

Недоліки Lightspeed Restaurant:

– вартість. В порівнянні з іншими POS-системами, вартість використання Lightspeed Restaurant може бути високою, особливо для невеликих кав'ярень з обмеженим бюджетом;

– навчання та імплементація. Впровадження системи та навчання персоналу можуть вимагати певного часу та зусиль, особливо якщо вони не мають досвіду в роботі з POS-системами [14];

– залежність від Інтернет-з'єднання. Оскільки Lightspeed Restaurant базується на хмарних технологіях, для його ефективної роботи потрібне стабільне Інтернет-з'єднання. Відсутність Інтернету може спричинити перебої у функціональності системи.

Отже, Lightspeed Restaurant - це потужна POS-система, розроблена спеціально для ресторанного бізнесу, яка пропонує широкий функціонал для ефективного управління кав'ярнею Neroaroma. Вона включає в себе керування замовленнями, оплатою, столиками, запасами та звітністю, а також має гнучку та мобільну архітектуру, що сприяє зручному використанню на мобільних пристроях. Незважаючи на високу вартість та потребу в навчанні, Lightspeed Restaurant є потужним інструментом для оптимізації бізнес-процесів та підвищення ефективності кав'ярні.

Для отримання більш глибокого розуміння функціональності розглянутих додатків, доцільно провести порівняння їх загальних функціональних характеристик. У цілях порівняння була підготовлена таблиця (табл. 1.1), яка включає основні характеристики та можливості додатків Toast, Square for Restaurants та Lightspeed Restaurant. Це дозволить зробити аналіз їх сильних сторін, особливостей та визначити, який з них найбільше відповідає потребам та вимогам кав'ярні Neroaroma.

Таблиця 1.1– Порівняльний аналіз розглянутих систем

Характеристика	Toast	Square	Lightspeed	Neroaroma
Формування звітності	+	+	+	+
Багатоплатформність	+	+	+	-
Керування столиками	-	+	-	+
Облік фінансових операцій	+	-	+	+
Редагування інформації користувачів	-	+	-	+

Розробка нового програмного забезпечення для кав'ярні Neroaroma може бути обґрунтованою необхідністю з кількох причин. По-перше, існуючі застосунки, такі як Toast, Square for Restaurants і Lightspeed Restaurant, хоча й пропонують широкий функціонал та забезпечують ефективне управління ресторанним бізнесом, не повністю відповідають специфіці кав'ярні Neroaroma і її потребам. Розробка власного ПЗ дозволить налаштувати систему під конкретні потреби та процеси кав'ярні.

По-друге, нове програмне забезпечення може бути розроблене з урахуванням індивідуальних вимог та пропонувати унікальні функціональні можливості, що сприятимуть підвищенню ефективності та конкурентоспроможності кав'ярні.

Крім того, розробка власного ПЗ дозволить забезпечити більшу гнучкість, контроль та налагодження процесів управління, а також

забезпечить можливість майбутнього розширення та адаптації до змінних потреб кав'ярні.

1.3 Постановка задачі

Автоматизована система, розроблена для кав'ярні Neroaroma, має на меті вирішити основні завдання, пов'язані з обліком меню та столиків, обліком замовлень та управлінням фінансами. Система повинна зберігати дані про всі замовлення, які надходять до ресторану, реєструє замовлення для столиків та меню. Крім того, вона повинна забезпечувати облік працівників та їхню заробітну плату, а також дозволяти формувати різноманітну звітність.

Розробка системи передбачає створення схеми бази даних та реалізацію трьох основних модулів: модуля вводу/редагування інформації в базі даних, модуля захисту інформації та модуля формування звітності про діяльність ресторану. Реалізація додатка здійснюється з використанням технологій .NET та мови програмування C#, що забезпечує ефективний розвиток та функціональність системи.

Основні функції системи мають включати:

– вводити та редагувати інформацію про користувачів системи, категорії меню, саме меню, клієнтів, працівників, посади та столики;

– фіксувати замовлення, зроблені в ресторані, включаючи деталі та склад замовлення;

– фіксувати додаткові витрати, пов'язані з діяльністю ресторану, такі як закупівля сировини або обладнання.

Також, система повинна формувати наступну звітність:

– за вибраний період часу, в якому детально відображається статистика по замовленнях, зроблених протягом цього періоду;

– детальну статистика про замовлення клієнтів, включаючи інформацію про їхні вибори та витрати;

– звітність, згрупована за офіціантами, що дозволяє відобразити детальну статистику щодо обслуговування столиків кожним окремим офіціантом;

– звітність про загальні витрати ресторану за вибраний період, що дозволить детально проаналізувати витрати, пов'язані з діяльністю ресторану.

1.4 Висновок

У цьому розділі було проведено детальний аналіз бізнес-процесів кав'ярні Neroaroma. Це включало вивчення організаційної структури закладу та моделі потоків даних. Через розгляд різних рівнів управління та взаємодії між ними, було виявлено декілька областей, де автоматизація і оптимізація можуть зробити процеси більш ефективними та продуктивними.

Проведений огляд існуючих програмних рішень, включаючи Toast, Square for Restaurants та Lightspeed Restaurant, виявив, що жодне з них не відповідає специфічним потребам Neroaroma. Ці системи мають ряд недоліків та обмежень, що не дозволяють їм ефективно вирішувати певні задачі, які виникають в процесі роботи кав'ярні. Також, вони мають загальну орієнтацію, що не враховує унікальні особливості і потреби Neroaroma.

Таким чином, було обґрунтовано необхідність розробки нового програмного рішення, яке було б спеціалізовано під потреби кав'ярні Neroaroma. Розробка власного програмного забезпечення на мові програмування C# дозволить максимально адаптувати систему під потреби кав'ярні, що, в свою чергу, підвищить ефективність управління та забезпечить кращі умови для обслуговування клієнтів.

На основі проведеного аналізу була сформульована задача дослідження, яка передбачає розробку програмного забезпечення для автоматизації та оптимізації бізнес-процесів кав'ярні Neroaroma з урахуванням всіх виявлених особливостей і потреб закладу.

2 ВИБІР ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог. Use-case діаграми. Основні прецеденти

Для створеної системи " Neroaroma" використання діаграм прецедентів є важливим інструментом для детального опису її функціональності та забезпечення зрозуміння використання користувачами. Прецеденти визначають сервіси, які надає система та описують, як користувачі можуть взаємодіяти з системою для досягнення певних результатів [15]. Діаграми прецедентів дозволяють уточнити вимоги до системи, описати її поведінку з точки зору користувачів та забезпечити ефективну взаємодію між ними та системою. Використання діаграм прецедентів є важливим етапом проектування, що допомагає уточнити функціональні вимоги та забезпечити конкретний та точний опис функціональності системи [16].

Таблиця 2.1 – Функціональні вимоги до додатку

Вимоги	Опис
REQ-1	Реєстрація та авторизація користувачів
REQ-2	Ведення журналу подій
REQ-3	Керування обліковими записами користувачів
REQ-4	Керування інформацією про клієнтів
REQ-5	Керування інформацією про посади
REQ-6	Керування інформацією про працівників
REQ-7	Керування інформацією про меню ресторану
REQ-8	Керування інформацією про столики ресторану
REQ-9	Фіксація замовлення клієнта
REQ-10	Формування звітності за різноманітними критеріями

Таблиця 2.2 – Нефункціональні вимоги до додатку

Вимоги	Опис
REQ-11	Додаток повинен бути стабільним і надійним, з високим рівнем доступності. Він повинен забезпечувати безперебійну роботу і відсутність втрати даних, навіть у випадку відмови апаратного чи програмного забезпечення.
REQ-12	Додаток повинен забезпечувати захист конфіденційної інформації, включаючи дані клієнтів, працівників та фінансову інформацію. Він має мати механізми аутентифікації, авторизації та контролю доступу, а також механізми резервного копіювання та відновлення даних.
REQ-13	Додаток повинен працювати ефективно та швидко, з мінімальними затримками при обробці запитів користувачів і генерації звітів. Він повинен бути масштабованим, здатним обробляти великий обсяг даних і забезпечувати швидку відповідь навіть при збільшенні навантаження.

Таблиця 2.3 – Актори та цілі додатку

Актори	Цілі
Адміністратор	Мета адміністратора в системі полягає у керуванні обліковими записами. Він відповідає за створення, редагування та видалення облікових записів користувачів.
Користувач	Головна мета користувача полягає у зручному та ефективному використанні системи для оптимального управління ресторанним бізнесом.
База даних	Мета бази даних полягає в забезпеченні доступу до актуальної та надійної інформації для подальшого використання в роботі системи та генерації звітів.

Таблиця 2.4 – Опис варіантів використання додатку

Варіант використання	Ім’я	Опис
UC1	Автентифікація в системі	Процедура автентифікації користувачів
UC2	Вивід каталогу користувачів	Перегляд каталогу користувачів для адміністратора
UC3	Додати користувача	Додавання нових користувачів у систему
UC4	Редагувати користувача	Редагування записів облікових даних користувачів
UC5	Вивід каталогу клієнтів	Перегляд каталогу клієнтів
UC6	Додати клієнта	Додавання інформації про нового клієнта
UC7	Редагувати клієнта	Редагування інформації вибраного із списку клієнта
UC8	Вивід каталогу посад	Виведення каталогу всіх посад
UC9	Додати посаду	Додавання інформації про нову посаду
UC10	Редагувати посаду	Редагування інформації вибраної із списку посади
UC11	Вивід каталогу працівників	Виведення каталогу всіх працівників
UC12	Додати працівника	Додавання інформації про нового працівника
UC13	Редагувати працівника	Редагування інформації вибраного із списку працівника
UC14	Вивід каталогу столиків	Виведення каталогу всіх столиків ресторану

UC15	Додати столик	Додавання інформації про новий столик
UC16	Редагувати столик	Редагування інформації вибраного із списку столика
UC17	Прийняти замовлення	Прийняття замовлення меню клієнта
UC18	Відкриті замовлення	Перегляд інформації про відкриті замовлення
UC19	Закриті замовлення	Перегляд інформації про закриті замовлення
UC20	Звіт за період часу	Формування звітності по обслуговуванню клієнтів за вибраний період часу
UC21	Звіт по клієнту	Формування звітності по обслуговуванню вибраного клієнта
UC22	Звіт по офіціанту	Формування звітності по обслуговуванню вибраного офіціанта
UC23	Звіт по всіх витратах ресторану	Формування звітності по загальних витратах ресторану за вибраний період часу
UC24	Вивід системних подій	Виведення всіх подій, які відбулися в системі

На основі зібраних даних було розроблено use-case діаграми прецедентів для ролей системного адміністратора та користувача. Діаграма прецедентів для системного адміністратора зображена на рис. 2.1, а для користувача - на рис. 2.2. Ці діаграми відображають взаємодію користувачів з системою та описують функціональні можливості, які вони можуть виконувати. Вони допомагають зрозуміти вимоги до системи та забезпечують чітку специфікацію її функціональності.

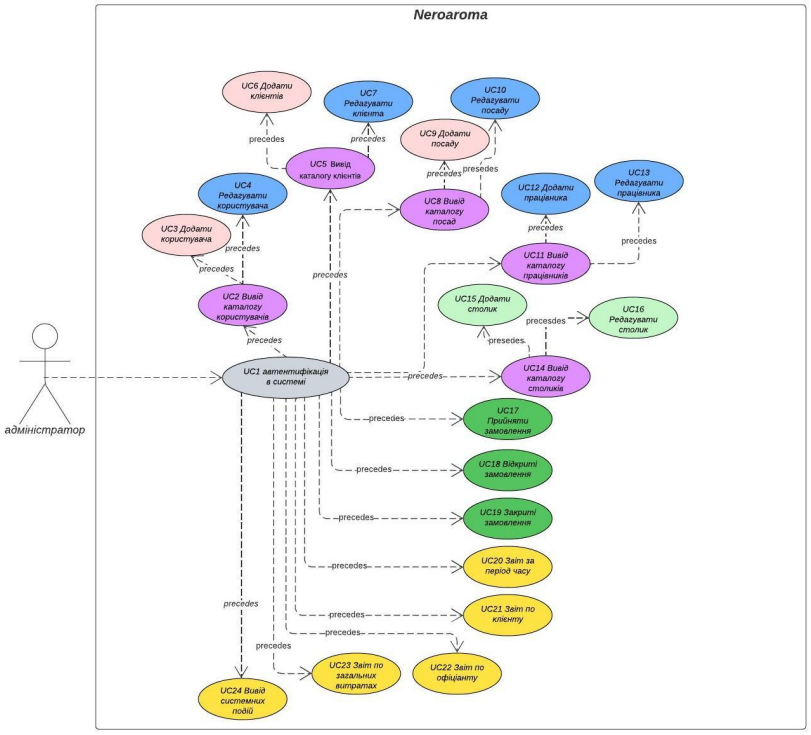

Рисунок 2.1 – Діаграма use-case для ролі «системний адміністратор»

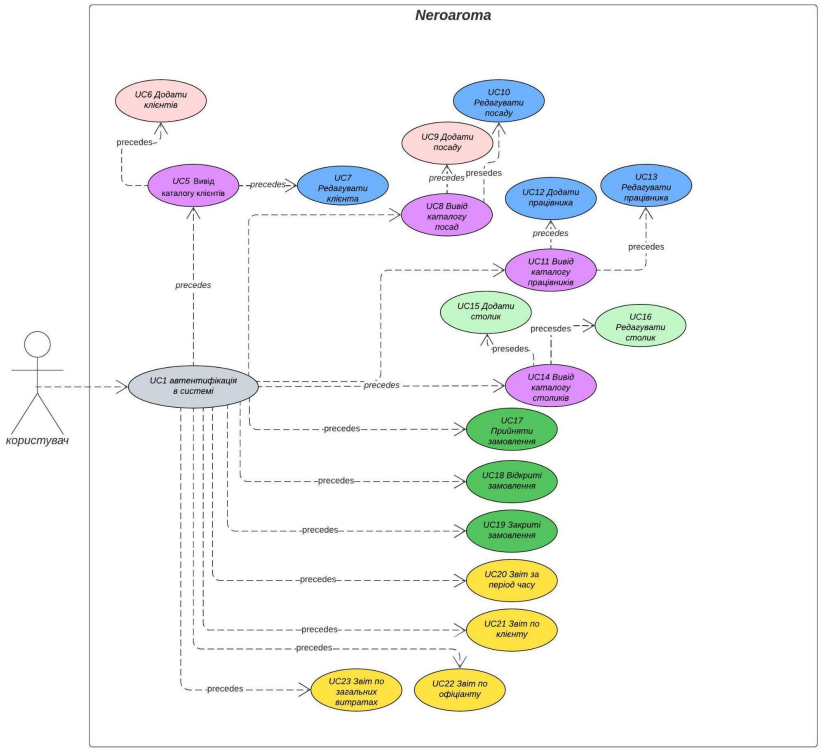

Рисунок 2.2 – Діаграма use-case для ролі «користувач»

2.2 Вибір технологій

Вибір оптимальних технологій є важливим аспектом при розробці автоматизованої системи обліку витрат. Він впливає на ефективне втілення функціональності системи, забезпечення її надійності та масштабованості. Підбір правильних технологій дозволяє використовувати сучасні інструменти та рішення, що відповідають потребам проекту, забезпечують оптимальну продуктивність та забезпечують зручність у розробці та підтримці системи.

2.2.1 Мова програмування

З огляду на вимоги до продуктивності, масштабованості та сумісності з різними платформами, варто розглянути мови програмування C#, C++ та Java.

Мова програмування C# є об'єктно-орієнтованою мовою, розробленою компанією Microsoft [17]. Вона володіє високою продуктивністю та ефективним керуванням пам'яттю, що робить її популярним вибором для розробки широкого спектру застосунків. C# має широку підтримку бібліотек та фреймворків, що спрощує розробку різноманітних програмних рішень.

Мова програмування C++ відзначається високою продуктивністю та можливістю безпосереднього керування ресурсами комп'ютера [18]. Вона часто використовується для розробки системного програмного забезпечення, компіляторів та інших критичних за ресурсами додатків. C++ також підтримує об'єктно-орієнтований та процедурний підходи до програмування.

Мова програмування Java, розроблена компанією Sun Microsystems (пізніше придбана Oracle), є платформонезалежною та має високу переносимість між різними операційними системами [19]. Вона широко використовується для розробки мобільних додатків, веб-додатків та великих підприємницьких систем. Java має вбудовану підтримку багатопотоковості та забезпечує високий рівень безпеки.

У табл. 2.5 проведено порівняльний аналіз можливостей кожної з розглянутих мов програмування: C#, C++ та Java.

Таблиця 2.5 – Порівняльний аналіз мов програмування

Характеристика	C#	C++	Java
Продуктивність	Висока	Висока	Середня
Масштабованість	Висока	Висока	Висока
Сумісність	Платформо-незалежна	Можливість компіляції під різні платформи	Платформо-незалежна
Підтримка ООП	Повна підтримка	Повна підтримка	Повна підтримка
Керування пам'яттю	Автоматичне керування пам'яттю	Ручне керування пам'яттю	Автоматичне керування пам'яттю
Екосистема	Microsoft .NET	Різноманітні фреймворки	Java Virtual Machine (JVM)
Використання	Розробка десктопних та веб-додатків	Розробка системного ПЗ, графічних додатків	Розробка веб-додатків, серверних програм
Спільнота	Велика	Велика	Велика

Вибір мови програмування C# для підтримки бізнес-процесів кав'ярні Neroaroma обгрунтовується декількома факторами:

– забезпечує повну підтримку об'єктно-орієнтованого програмування, що дозволяє ефективно моделювати бізнес-процеси та створювати структуровані, гнучкі та легко розширювані додатки;

– має велику спільноту розробників та розширену екосистему Microsoft .NET, що забезпечує доступ до різноманітних інструментів,

бібліотек і фреймворків, що полегшують розробку, тестування та супровід додатків;

– підтримує автоматичне керування пам'яттю, що сприяє зменшенню ризиків пов'язаних з помилками у керуванні пам'яттю і полегшує процес розробки, особливо для команд, що не мають великого досвіду у низькорівневому програмуванні.

З урахуванням цих факторів, мова програмування C# є відмінним варіантом для реалізації системи підтримки бізнес-процесів кав'ярні Neroaroma. Вона забезпечує широкий функціонал, швидкість розробки, надійність та масштабованість, що важливо для успішної автоматизації та оптимізації роботи кав'ярні.

2.2.2 Середовище розробки

Середовища розробки відіграють важливу роль у процесі створення програмного забезпечення. Для автоматизованої системи обліку витрат в ресторанній діяльності можна розглянути наступні середовища розробки:

– Visual Studio (IDE) розроблене компанією Microsoft. Воно надає розробникам можливість створювати, відлагоджувати та тестувати програми на різних мовах програмування, включаючи C#, C++, Java, Python та інші [20]. Visual Studio має широкий набір інструментів та функціональних можливостей, що полегшують розробку і підтримку додатків.

– Інтегроване середовище розробки Eclipse розроблене фондом Eclipse. Воно спеціалізується на розробці програм на мові програмування Java та підтримує інші мови. Eclipse надає розробникам потужні інструменти для розробки, відлагодження та тестування програм, а також розширення та налаштування середовища з використанням плагінів [21].

– Середовище розробки PyCharm розроблене компанією JetBrains, спеціалізоване на розробці програм на мові програмування Python. PyCharm надає розробникам розширені можливості для розробки, відлагодження та

тестування програм на мові Python, включаючи автоматичне завершення коду, аналіз помилок, підтримку віртуальних середовищ та інше [22].

Таблиця 2.6 – Порівняльний аналіз середовищ розробки

Особливість	Visual Studio	Eclipse	PyCharm
Підтримувані мови	C#, C++, Visual Basic, F#, і багато інших	Java, C++, PHP, Python, і багато інших	Python, JavaScript, TypeScript, і багато інших
Інтегрована система керування версіями	Так	Так	Так
Вбудований дебагер	Так	Так	Так
Підтримка тестування	Так	Так	Так
Вбудована документація	Так	Так	Так
Функція/Особливість	Visual Studio	Eclipse	PyCharm
Підтримувані мови	C#, C++, Visual Basic, F#, і багато інших	Java, C++, PHP, Python, і багато інших	Python, JavaScript, TypeScript, і багато інших

Отже, Visual Studio є популярним і потужним інтегрованим середовищем розробки, особливо для розробки програм на мовах, таких як C#, C++ і Visual Basic. Воно має широкий набір функцій і можливостей, включаючи розширення, плагіни, вбудований дебагер, систему керування версіями та підтримку тестування. Visual Studio також має велику активну спільноту користувачів і офіційну підтримку, що робить його надійним вибором для багатьох розробників.

2.2.3 СУБД

У сучасному світі, де обсяги даних неперервно зростають, системи управління базами даних (СУБД) набувають все більшої вагомості. Вони є вирішальним елементом для збереження, організації та доступу до даних для різноманітних додатків та систем. Вибір відповідної СУБД може суттєво вплинути на швидкодію роботи додатків, рівень безпеки даних та їх доступність для аналізу та використання.

У цьому підрозділі розглянуто кілька популярних СУБД, а також їх переваги та недоліки. Одним з важливих факторів при виборі СУБД є розмір даних, оскільки різні системи можуть мати різні обмеження на обсяги даних, з якими вони ефективно працюють. Крім того, складність запитів може впливати на ефективність та продуктивність СУБД. Деякі СУБД можуть мати покращені механізми оптимізації запитів, що дозволяє їм працювати швидше при складних запитах.

MS SQL Server є однією з провідних реляційних баз даних, розроблених компанією Microsoft. Вона використовується для збереження та управління даними на різних платформах, зокрема Windows та Linux [23]. MS SQL Server володіє широким спектром функціональних можливостей, які дозволяють ефективно опрацьовувати та керувати різними типами даних. Однією з ключових переваг MS SQL Server є його розширена підтримка мови SQL (Structured Query Language), що дозволяє легко виконувати запити до бази даних та маніпулювати даними. Вона також надає можливості для розробки складних процедур, функцій та тригерів, що дозволяють забезпечити більшу гнучкість та функціональність в управлінні даними.

Oracle - це одна з найпоширеніших реляційних баз даних у світі, розроблена компанією Oracle Corporation [24]. Її використання поширене у різних галузях, включаючи бізнес, організації та установи. Oracle володіє великим набором вбудованих функцій та засобів, які дозволяють підвищити ефективність роботи з даними та оптимізувати взаємодію з базою даних. Однією з визначних переваг Oracle є його висока масштабованість та

надійність. Він може обробляти великі обсяги даних та підтримувати велику кількість одночасних користувачів. База даних Oracle також володіє вбудованою механікою відновлення, що забезпечує збереження даних у випадку аварійної ситуації та забезпечує високий рівень надійності.

MS Access - це реляційна база даних, розроблена компанією Microsoft, призначена для роботи з невеликими базами даних. Хоча MS Access має обмежену масштабованість, вона володіє простим інтерфейсом та надає зручні можливості для швидкого створення та редагування баз даних [25].

Однією з особливостей MS Access є його легкість використання та навчання. Завдяки інтуїтивно зрозумілому інтерфейсу, користувачі можуть швидко створювати таблиці, запити, форми та звіти без необхідності в глибоких знаннях SQL або програмування. Це робить MS Access популярним інструментом для невеликих проектів та персонального використання.

MS Access підтримує стандартні функції реляційних баз даних, такі як зв'язки між таблицями, запити для вибору та модифікації даних, індексацію та сортування. Він також має вбудовані інструменти для створення звітів та форм, що дозволяє відображати та організовувати дані в зручному для користувача форматі.

Порівняння основних характеристик MS SQL Server, Oracle та MS Access можна побачити в табл. 2.7.

Таблиця 2.7 – Порівняльний аналіз сховищ БД

Характеристика	*MS SQL Server*	*Oracle*	*MS Access*
Розробник	Microsoft	Oracle Corporation	Microsoft
Тип бази даних	Реляційна	Реляційна	Реляційна
Версія	2019	19c	2019
Масштабованість	Велика	Велика	Мала
Підтримка мов програмування	T-SQL, C#, Java та ін.	PL/SQL, Java та ін.	VBA, SQL

Інтерфейси	ODBC, OLE DB, JDBC, ADO.NET	ODBC, JDBC, ADO.NET	ODBC, OLE DB
Підтримка операційних систем	Windows, Linux	Windows, Linux, UNIX	Windows

У СУБД для зберігання даних додатку було обрано MS Access. Однією з основних переваг MS Access є його простий інтерфейс та зручність в створенні невеликих баз даних, які відповідають потребам користувача. Ця СУБД надає можливість з легкістю створювати таблиці, запити, форми та звіти без потреби у глибоких знаннях SQL або програмування. MS Access спрощує процес розробки та управління базою даних, особливо для невеликих проектів або користувачів без спеціалізованих навичок у цій області.

Додатковою перевагою використання MS Access є те, що вона входить до складу популярного пакету програм Microsoft Office. Це робить MS Access доступною для користувачів, які вже використовують інші програми Office, такі як Microsoft Word чи Excel. Це створює зручність та сумісність між різними програмами, що сприяє легкому обміну даними та інтеграції між різними компонентами Office Suite.

2.3 Архітектура проекту

Для розробки даного проекту була використана трьохрівнева архітектура, яка є типом архітектури програмного забезпечення, що базується на розподілі системи на три рівні: клієнтський, логічний та даних. Кожен рівень виконує певні функції та має свої відповідальності та обмеження [26].

На першому рівні, клієнтському рівні, знаходяться компоненти, які відповідають за інтерфейс користувача та взаємодію з користувачем. Це можуть бути веб-сторінки, додатки для мобільних пристроїв або десктопні

додатки. На цьому рівні відбувається представлення даних користувачам та збір вхідних даних від них. Клієнтський рівень відповідає за забезпечення зручного та ефективного способу взаємодії користувача з системою.

На другому рівні, логічному рівні, розташовані компоненти, які відповідають за бізнес-логіку системи. Цей рівень включає обробку даних, валідацію, розрахунки та управління бізнес-процесами [27]. Сервіси, що надаються на цьому рівні, забезпечують взаємодію між клієнтським та даних рівнями. Логічний рівень відповідає за забезпечення коректності та консистентності даних, а також за виконання бізнес-правил та логіки додатку.

На третьому рівні, рівні даних, розташовані компоненти, які відповідають за зберігання та обробку даних. Це може бути база даних, файлові системи або будь-які інші механізми зберігання даних. На цьому рівні знаходяться механізми для зчитування, запису, модифікації та видалення даних зі зберігаючих систем. Рівень даних відповідає за забезпечення безпеки, доступності та ефективного зберігання даних.

Трьохрівнева архітектура має кілька основних переваг [28]:

– розділення відповідальності. Кожен рівень (клієнтський, логічний, даних) має свої відповідальності та функції. Це дозволяє забезпечити чітку структуру та розподіл завдань між компонентами системи. Розділення рівнів спрощує розробку, тестування та підтримку програмного забезпечення;

– модульність. Кожен рівень може бути розглянутий як окремий модуль з власними функціями та інтерфейсами. Це дозволяє легко змінювати або розширювати один рівень, не впливаючи на інші. Модульність сприяє перевикористанню коду та забезпечує гнучкість системи;

– покращена розподіленість. Трьохрівнева архітектура дозволяє розподіляти компоненти системи на різних серверах або в різних середовищах. Це сприяє горизонтальній масштабованості, дозволяючи розділити навантаження та поліпшити продуктивність системи.

– більша безпека. Завдяки трьохрівневій архітектурі, можливо забезпечити більшу безпеку. Клієнтський рівень взаємодіє лише з логічним рівнем, а логічний рівень взаємодіє з даними рівнем. Це дозволяє контролювати доступ до даних та застосовувати заходи безпеки на рівні, який є найбільш доцільним;

– підвищена швидкодія та ефективність. Розділення функцій на рівні архітектури дозволяє оптимізувати роботу кожного окремого рівня. Клієнтський рівень може бути оптимізований для відображення та взаємодії з користувачем, логічний рівень - для обробки даних та бізнес-логіки, а рівень даних - для ефективного зберігання та доступу до даних. Це сприяє покращенню продуктивності та швидкодії системи.

Отже, трьохрівнева архітектура є популярним підходом в розробці програмного забезпечення, оскільки вона надає багато переваг, зокрема чітку структуру, модульність, розподіленість, безпеку та покращену ефективність.

2.3.1 Особливості розробки рівня даних (DAL)

Задача Data Access Layer (DAL) полягає у забезпеченні доступу до даних, знаходячись на рівні бази даних, з інших рівнів системи, зокрема з рівня сервісів (Business Logic Layer - BLL). У контексті системи обліку витрат у ресторанній діяльності, DAL було реалізовано з використанням технології ADO.NET, зокрема за допомогою об'єкту System.Data.OleDb, який надає можливість з'єднання з базою даних за допомогою різних постачальників даних (зображено на рис. 2.3).

DAL виконує важливу роль у системі, оскільки він відповідає за обробку запитів до бази даних та взаємодію з нею. Він надає абстракцію над конкретним постачальником даних та забезпечує інтерфейс для виконання операцій, таких як отримання, вставлення, оновлення та видалення даних. Такий підхід дозволяє зробити систему більш гнучкою, оскільки зміна постачальника даних не впливає на решту системи.

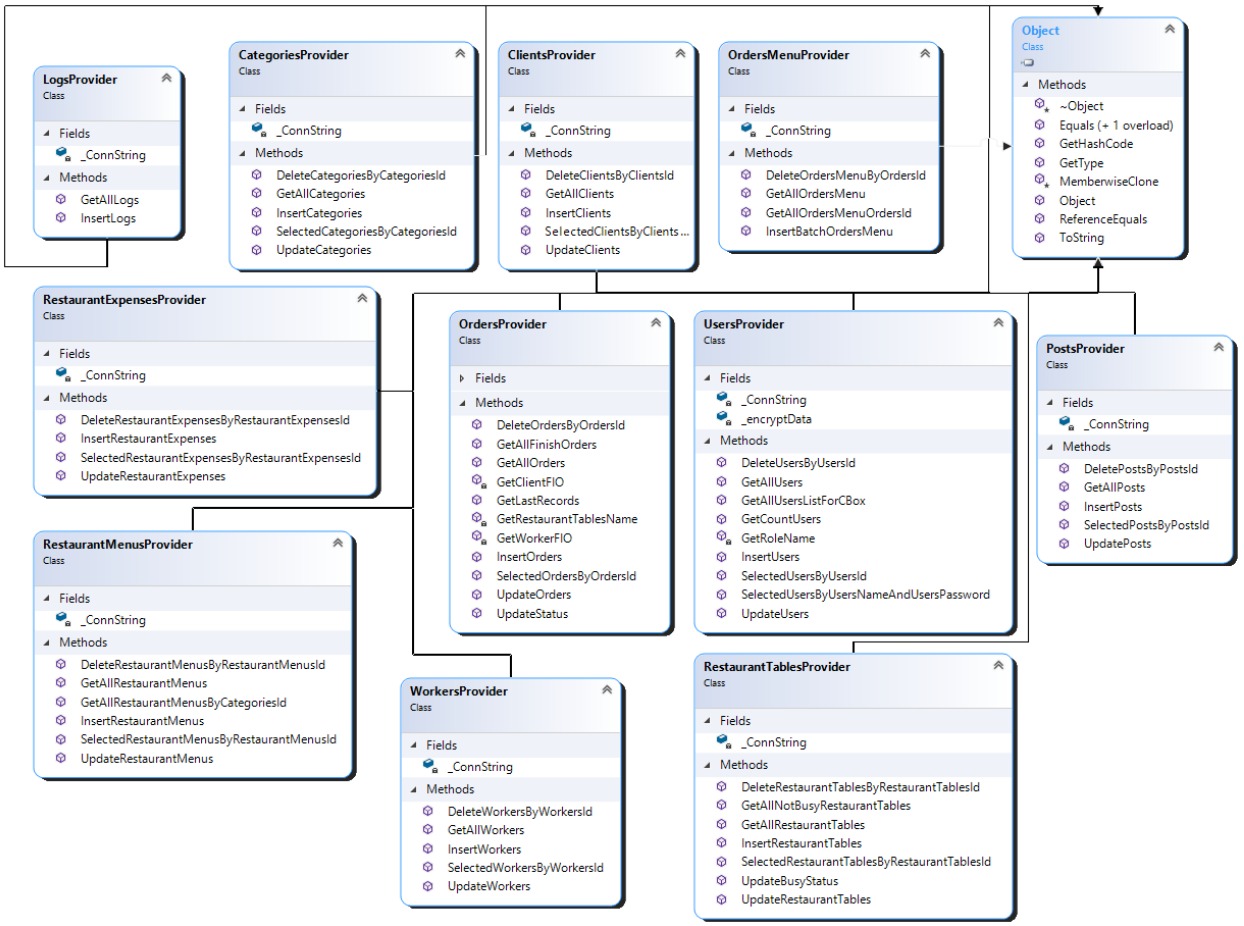

Рисунок 2.3 – Діаграма класів рівня даних

Як можна побачити із зображення на рис. 2.3 діаграмою класів рівня даних, цей рівень складається з одинадцяти основних класів, кожен з яких виконує певні функції:

– клас LogsProvider відповідає за запис подій та дій користувачів в системі обліку витрат. Він містить методи для додавання нового запису в журнал подій, відображення журналу та очищення журналу. Цей клас забезпечує ведення журналу подій, що дозволяє відстежувати дії користувачів і забезпечує історичну інформацію про події в системі;

– клас OrdersMenuProvider відповідає за роботу зі замовленнями та стравами. Він містить методи для обробки замовлень, включаючи додавання нового замовлення, видалення замовлення, редагування деталей замовлення та взаємодію зі списком страв. Цей клас забезпечує ефективне управління замовленнями та стравами, необхідними для ресторанного обліку витрат;

– клас PostsProvider відповідає за операції, пов'язані з посадами працівників ресторану. Він містить методи для отримання списку посад з бази даних, додавання нової посади, видалення та редагування існуючої посади;

– клас CategoriesProvider відповідає за операції, пов'язані з категоріями страв. Він надає методи для отримання списку категорій з бази даних, додавання нової категорії, видалення та редагування існуючої категорії. Цей клас дозволяє керувати категоріями страв і забезпечує необхідні операції з ними;

– клас ClientsProvider відповідає за роботу з клієнтами. Він містить методи для отримання списку клієнтів, додавання нового клієнта, видалення та редагування інформації про існуючого клієнта. Цей клас дозволяє ефективно керувати клієнтами системи та забезпечує зручні операції з ними;

– клас RestaurantExpensesProvider відповідає за роботу з витратами ресторану. Він містить методи для управління витратами, включаючи додавання нових витрат, видалення та редагування витрат. Цей клас дозволяє системі вести облік та контролювати витрати ресторану, що є важливим елементом управління фінансами;

– клас RestaurantMenusProvider відповідає за роботу зі списком страв та меню ресторану. Він надає методи для керування списком страв, включаючи додавання нових страв, видалення та редагування страв. Крім того, цей клас дозволяє формувати та оновлювати меню ресторану, забезпечуючи гнучкість та зручність управління стравами;

– клас RestaurantTablesProvider відповідає за роботу зі списком столиків ресторану. Він містить методи для отримання списку столиків з бази даних, додавання нового столика, видалення та редагування існуючого столика. Цей клас дозволяє ефективно керувати столиками ресторану та надає можливість оперативного управління розташуванням та наявністю столиків;

– клас WorkersProvider забезпечує роботу з працівниками ресторану в системі. Він містить методи для отримання списку працівників з бази

даних, додавання нового працівника, видалення та редагування існуючого працівника. Крім того, цей клас надає можливість відображення статистики роботи працівників та формування звітів, що дозволяє відстежувати продуктивність та ефективність роботи працівників ресторану;

– клас UsersProvider відповідає за роботу з користувачами системи. Він надає методи для отримання списку користувачів з бази даних, додавання нового користувача, видалення та редагування існуючого користувача. Цей клас забезпечує керування доступом користувачів до системи та забезпечує аутентифікацію та авторизацію користувачів.

2.3.2 Особливості розробки рівня бізнес-логіки BLL

Створення бізнес-логіки в програмному додатку відіграє важливу роль у його архітектурі та розробці. Цей процес полягає у розділенні функціональності на логічні блоки, що сприяє покращенню модульності та гнучкості системи у майбутньому.

Один з основних вигод розділення бізнес-логіки полягає у можливості змінювати окремі частини функціоналу без впливу на решту системи. Це означає, що розробники можуть вносити зміни або додавати новий функціонал до окремих модулів, не впливаючи на решту коду. Такий підхід полегшує підтримку та розвиток проекту, оскільки він зменшує ризик виникнення неочікуваних проблем та конфліктів між різними частинами додатку.

Крім того, розділення бізнес-логіки від бази даних та користувацького інтерфейсу дозволяє використовувати різні бази даних та різні інтерфейси без необхідності змінювати саму бізнес-логіку. Це дозволяє системі працювати з різними джерелами даних та інтерфейсами, що може бути корисним у ситуаціях, коли змінюються вимоги до системи або коли необхідно інтегрувати її з існуючими системами.

На рисунку 2.4 показана діаграма цього шару, яка відображає структуру та взаємодію логічних блоків бізнес-логіки у програмному додатку. Ця діаграма надає візуальне представлення взаємозв'язків та взаємодії між компонентами бізнес-логіки, що сприяє зрозумінню та аналізу їхньої роботи.

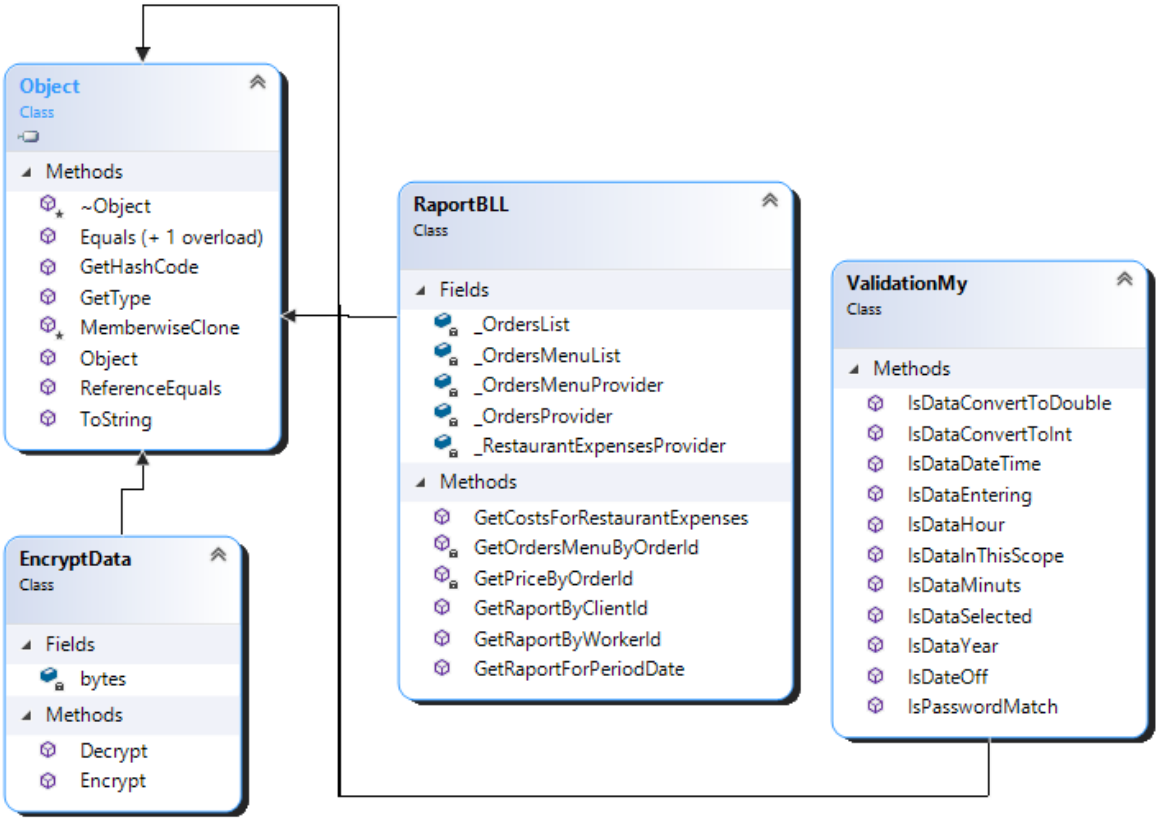

Рисунок 2.4 – Діаграма бізнес-логіки

Даний шар архітектури включає наступні класи, кожен з яких відіграє важливу роль у функціонуванні системи:

– клас EncryptData є відповідальним за шифрування даних в системі. Він надає методи для зашифрування та дешифрування даних з використанням певних алгоритмів шифрування. Крім того, він забезпечує зберігання та управління ключами шифрування, що забезпечує безпеку та конфіденційність даних, які обробляються в системі;

– клас ValidationMy відповідає за перевірку правильності введених даних в систему. Він надає методи для перевірки формату введених даних, перевірки наявності помилок, валідації обов'язкових полів та інші перевірки,

що допомагають забезпечити коректність та достовірність даних, які обробляються системою;

– клас RaportBLL відповідає за створення звітів та аналітики в системі. Він надає методи для генерації різних типів звітів, що включають статистичні дані, результати аналізу та іншу інформацію. Крім того, цей клас забезпечує обробку та аналіз даних, необхідних для створення звітів, що допомагає користувачам системи отримати цінні інсайти та приймати обґрунтовані рішення.

2.3.3 Особливості розробки рівня користувацького інтерфейсу (UI)

Ефективна автоматизація процесів обліку витрат у ресторанній діяльності вимагає ретельного розроблення рівня користувацького інтерфейсу (UI), який забезпечує зручну та інтуїтивно зрозумілу взаємодію між користувачами та системою. З урахуванням великої кількості різноманітних продуктів у ресторані, важливо, щоб інтерфейс користувача був максимально зрозумілим та забезпечував зручність використання для персоналу.

На рівні користувацького інтерфейсу були розроблені форми, що складаються з набору елементів керування, таких як кнопки, текстові поля та таблиці, які взаємодіють з іншими рівнями системи через обробники подій. Ці форми дозволяють користувачам взаємодіяти з системою, вводити необхідну інформацію, формувати звіти та отримувати результати у зручному для них форматі.

Діаграма класів системи рівня користувацького інтерфейсу, яка представлена на рисунку 2.6, відображає взаємозв'язки та структуру класів, що відповідають за роботу та взаємодію на цьому рівні. Ця діаграма надає візуальне уявлення про компоненти та їхні взаємозв'язки, що сприяє зрозумінню та аналізу функцій та можливостей, доступних для користувачів системи.

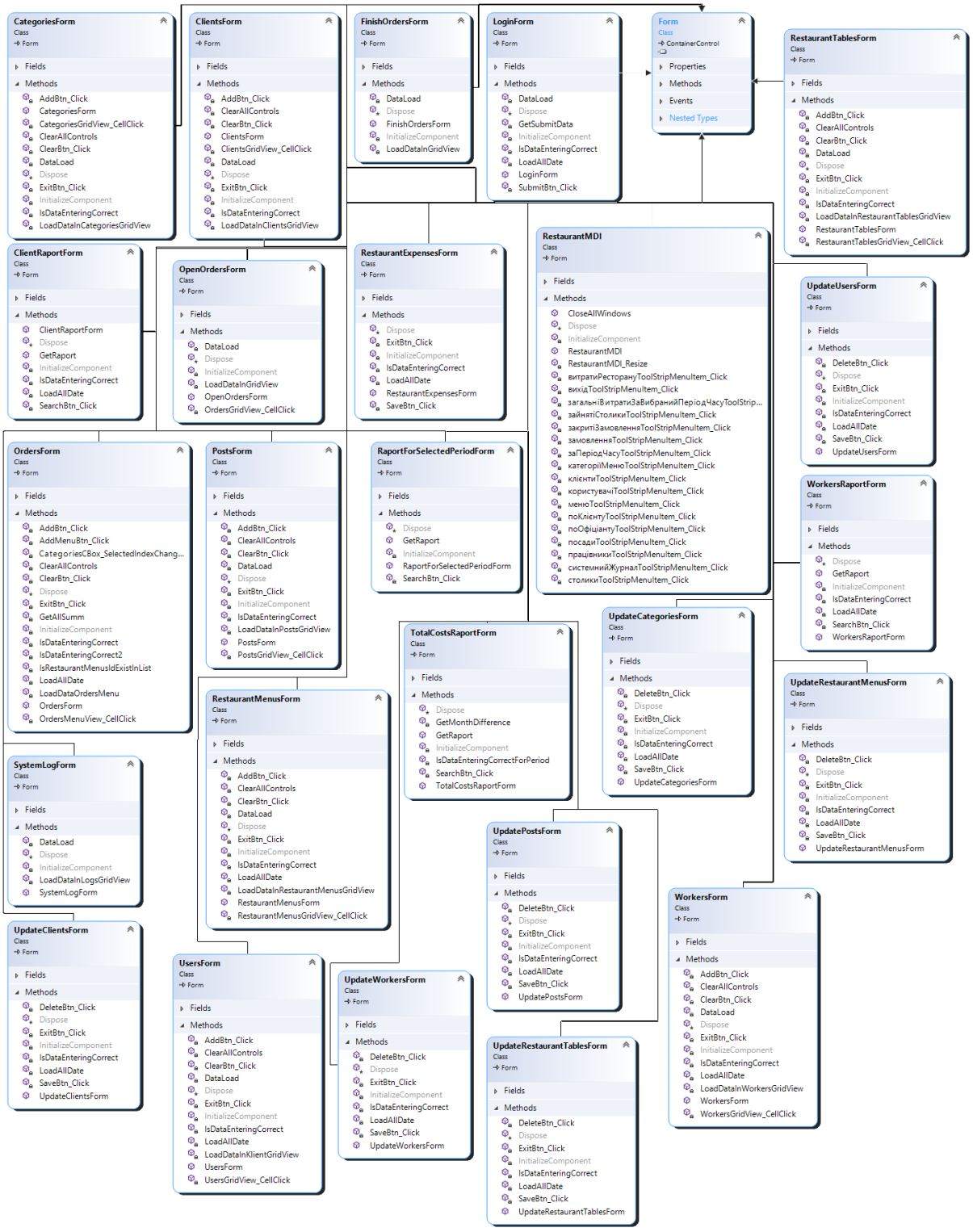

Рисунок 2.6 – Діаграма класів інтерфейсу системи

Діаграма рівня користувацького інтерфейсу (UI) складається з 25 класів, які є похідними від класу Form і мають графічний інтерфейс.

Один з головних класів у цій діаграмі - «RestaurantMDI», є центральним вікном програми. Він містить головне меню, яке дозволяє користувачеві

відкривати інші форми та виконувати необхідні дії. Для реалізації функціоналу головного меню використовується клас «MenuStrip», який дозволяє створювати меню з підменю та елементами меню. Він додається до головного вікна за допомогою методу «Controls.Add()».

Клас «ClientsForm» забезпечує функціональність роботи з клієнтами у системі. Він містить методи для додавання та перегляду інформації про клієнтів, що дозволяє управляти даними про клієнтів у системі.

Клас «PostsForm» забезпечує роботу з посадами персоналу ресторану. Він містить методи для додавання та перегляду інформації про посади, що дозволяє управляти даними про посади персоналу у системі.

Клас «RestaurantsExpensesForm» забезпечує функціональність обліку витрат ресторану. Він містить методи для введення та зберігання інформації про витрати ресторану. Цей клас дозволяє користувачам вводити дані про витрати, такі як закупівля інгредієнтів, платежі за послуги, зарплати працівників та інші витрати, і зберігати цю інформацію для подальшого обліку та аналізу.

Клас «RestaurantsMenusForm» забезпечує функціональність з меню ресторану. Він містить методи для додавання та перегляду інформації про страви та їх вартість у меню. Цей клас надає можливість користувачам додавати нові страви до меню, встановлювати їх ціну та переглядати наявність та характеристики різних страв у ресторані.

Клас «RestaurantsTablesForm» забезпечує функціональність для роботи з обліком столиків ресторану. Він містить методи для додавання та перегляду інформації про столики, такі як їх номер, кількість місць, стан (вільний або зайнятий).

Клас «UpdatesCategoriesForm» забезпечує функціональність опрацювання даних категорій страв у меню ресторану. Він містить методи для редагування та видалення категорій страв у меню, а також для перегляду

детальної інформації про категорії страв. Цей клас надає можливість користувачам змінювати назву категорії, додавати нові категорії, видаляти існуючі та переглядати детальну інформацію про наявні категорії страв у меню ресторану.

Клас «UpdatesClientsForm» забезпечує функціональність для опрацювання інформації про клієнтів у системі. Він містить методи для зміни та оновлення даних про клієнтів, такі як персональна інформація, контактні дані, інформація про замовлення та інші відомості, що стосуються клієнтів ресторану.

Клас «UpdatesRestaurantMenusForm» відповідає за редагування інформації про меню ресторану у системі автоматизованого обліку витрат. Він містить методи для зміни даних про страви, їхньої вартості, опису та доступності у меню ресторану.

Клас «ClientsRaportForm» відповідає за створення звіту про витрати клієнтів. Він містить методи для отримання інформації про витрати кожного клієнта та генерації звіту, який включає деталі про замовлення, вартість та перелік страв, які були спожиті клієнтом. Цей клас надає можливість аналізувати та відстежувати витрати окремих клієнтів.

Клас «RaportsForSelectedPeriodForm» відповідає за створення звіту про витрати ресторану. Він містить методи для отримання інформації про витрати за вибраний період, наприклад, за день, тиждень або місяць, і генерує звіт, який відображає загальну суму витрат за цей період та деталізовану інформацію про витрати по різних категоріях або стравах.

Клас «TotalCostRaportForm» відповідає за створення звіту про загальні витрати ресторану у системі. Він містить методи для отримання інформації про загальні витрати, включаючи витрати на інгредієнти, оренду, оплату праці, постачальників та інші витрати. Даний клас дозволяє генерувати звіт,

який показує загальну суму витрат ресторану та допомагає у керуванні фінансами ресторанної діяльності.

2.4 Висновок

У процесі аналізу вимог були використані use-case діаграми, які дозволили ідентифікувати основні прецеденти системи та визначити їх функціональність та взаємодію.

При виборі технологій були розглянуті різні аспекти, зокрема мова програмування. З урахуванням вимог та можливостей, мова C# була обрана як основна мова програмування для реалізації проекту. Для розробки було вибрано середовище розробки Visual Studio, що надає зручні інструменти та підтримку для мови C#. Також, для зберігання даних використовується СКБД MS Access, яка відповідає потребам проекту.

Архітектура проекту була побудована на трьохрівневій архітектурі, яка дозволяє розділити функціональність системи на рівні: рівень даних (DAL), рівень бізнес-логіки (BLL) та рівень користувацького інтерфейсу (UI). Кожен рівень має свої особливості та відповідальності, що забезпечує модульність, гнучкість та розширюваність проекту.

На рівні даних (DAL) використовується підхід ADO.NET для забезпечення доступу до бази даних MS Access. Рівень бізнес-логіки (BLL) містить класи, які відповідають за логіку обробки даних та бізнес-процесів. Рівень користувацького інтерфейсу (UI) включає форми та елементи керування, що забезпечують взаємодію користувача з системою.

Загалом, вибір технологій та архітектури програмного забезпечення був здійснений з метою створення ефективної та функціональної системи для підтримки бізнес-процесів кав'ярні Neroaroma. Обрані технології та архітектура дозволять реалізувати потрібний функціонал, забезпечити зручний інтерфейс користувача та ефективну роботу з даними.

3 РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування бази даних

Розробка проекту розпочинається з проектування бази даних, яке в даному випадку було здійснено за допомогою методу "сутність-зв'язок" (ER метод). Цей підхід до проектування баз даних передбачає ідентифікацію сутностей, їх атрибутів та взаємозв'язків між ними. Застосування цього методу дозволяє створити логічну модель бази даних, яка відображає предметну область та відношення між її складовими елементами.

У процесі проектування бази даних за методом "сутність-зв'язок" ідентифікуються сутності, які є ключовими елементами предметної області, наприклад, таблиці "Клієнти", "Страви", "Замовлення" тощо. Кожна сутність має свої атрибути, які описують властивості цієї сутності, наприклад, у таблиці "Клієнти" можуть бути атрибути "Ім'я", "Прізвище", "Адреса" та інші.

Зв'язки між сутностями визначаються на основі залежностей між ними. Наприклад, між сутностями "Клієнти" і "Замовлення" може бути зв'язок "Один-до-багатьох", оскільки один клієнт може мати кілька замовлень. Ці зв'язки відображаються у логічній моделі бази даних, що дозволяє відобразити структуру та взаємозв'язки між сутностями.

У процесі проектування бази даних за методом "сутність-зв'язок" для предметної області було виділено наступні сутності та атрибути:

– користувачі системи (зберігає інформацію про облікові записи користувачів): прізвище, ім’я, ім’я облікового запису, пароль, загальна інформація, ідентифікатор ролі;

– категорії продукції (зберігає інформацію про категорії продукції): назва та опис;

– клієнти (зберігає інформацію про клієнтів): прізвище, ім’я, № телефону, адреса проживання та електронна адреса;

– замовлення (зберігає інформацію про історію замовлень): дата/час замовлення, ідентифікатор клієнта, ідентифікатор працівника, ідентифікатор столика, опис та статус оплати;

– замовлення меню (зберігає інформацію про історію здійснені замовлення): ідентифікатор замовлення, ідентифікатор страви, кількість порцій та ціна;

– посади (зберігає інформацію про посади): назва та опис;

– витрати ресторану (зберігає інформацію про витрати ресторану): електроенергія, вода, газ, інтернет/телефон, та послуги з вивезення сміття;

– меню ресторану (зберігає інформацію про страви та напої): назва блюда/напитку, ціна, інгредієнти, кількість калорій, опис, вага та ідентифікатор та категорії;

– столики (зберігає інформацію про столики ресторану): назва столика, кількість сидінь, розташування, опис та статус зайнятості;

– працівники (зберігає інформацію про працівників): прізвище, ім’я, № телефону, адреса проживання та електронна адреса, заробітна плата, ідентифікатор посади;

– події (зберігає інформацію про події в системі) ідентифікатор користувача, опис події, дата/час події.

Під час проектування бази даних було визначено зв'язки між сутностями та встановлені зовнішні та внутрішні ключі для їхнього взаємозв'язку. Застосування зовнішнього та внутрішнього ключа дозволило встановити зв'язки між таблицями та визначити типи зв’язків, такі як "один до одного", "один до багатьох" та "багато до багатьох". Визначення типів зв’язків між сутностями допомогло правильно побудувати логічну модель бази даних, що забезпечить її коректну роботу та ефективне використання.

Організовано такі зв'язки між таблицями:

– "Posts" (поле "PostsId") – "Workers" (поле "PostsId"), вид зв’язку 1:N;

– "Categories" (поле "CategoriesId") – "RestaurantMenus" (поле "CategoriesId"), вид зв’язку 1:N;

– "Users" (поле "UsersId") – "Logs" (поле "UsersId"), вид зв’язку 1:N;

– "RestaurantMenus" (поле "RestaurantMenusId") – "OrdersMenu" (поле "RestaurantMenusId"), вид зв’язку 1:N;

– "Orders" (поле "OrdersId") – "OrdersMenu" (поле "OrdersId"), вид зв’язку 1:N;

– "RestaurantTables" (поле "RestaurantTablesId") – "Orders" (поле "RestaurantTablesId"), вид зв’язку 1:N;

– "Workers" (поле "WorkersId") – "Orders" (поле "WorkersId"), вид зв’язку 1:N;

– "Posts" (поле "PostsId") – "Workers" (поле "PostsId"), вид зв’язку 1:N.

ER-діаграма є ключовим етапом у процесі проектування бази даних, оскільки вона дозволяє графічно відобразити зв'язки між сутностями та їх атрибутами. Ця діаграма використовує концептуальну модель для визначення сутностей, зв'язків та атрибутів, що утворюють основу бази даних.

У результаті проведеного проектування бази даних для даного проекту було створено ER-діаграму, яка чітко відображає взаємозв'язки між таблицями та їх полями. Ця діаграма відображає структуру бази даних та показує, як таблиці пов'язані між собою за допомогою ключів та зв'язків. Кожна таблиця на діаграмі представлена як прямокутник, в якому перераховані її атрибути, а зв'язки між таблицями показані за допомогою стрілок та символів, що вказують на тип зв'язку.

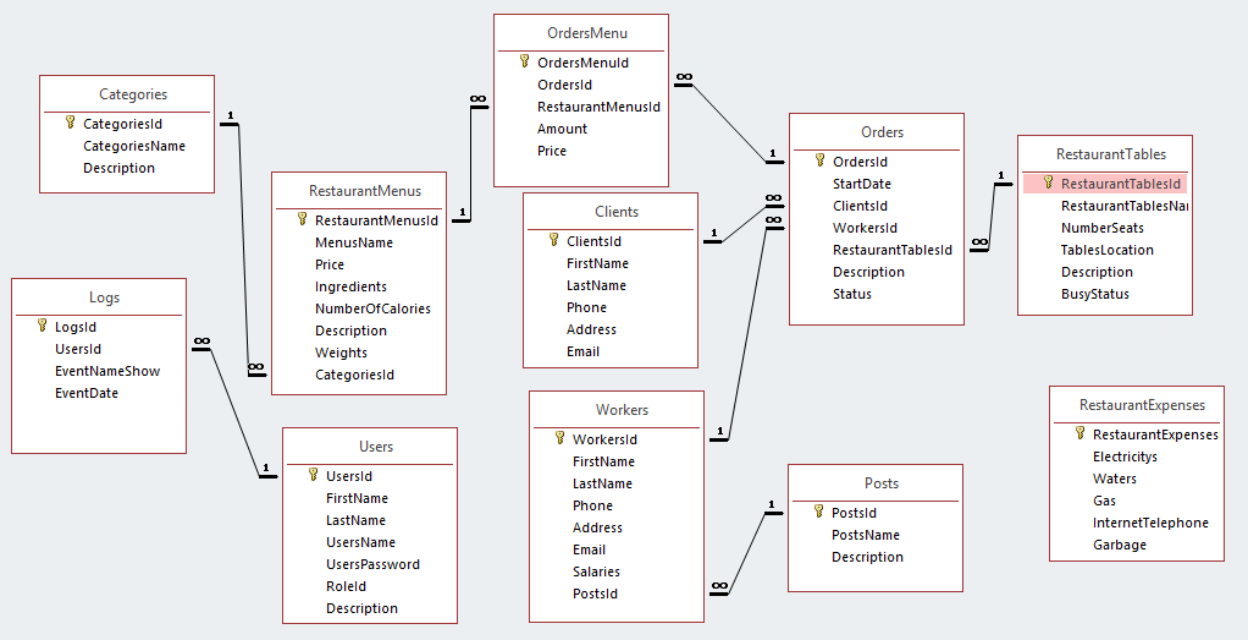

Рисунок 3.1 – Діаграма бази даних

3.2 Розробка основних алгоритмів

Завдання розробки додатку "Neroaroma" полягає у створенні зручної та легко у використанні платформи, яка забезпечує користувачів необхідною інформацією та можливістю ефективної взаємодії з додатком. Для досягнення цієї мети, розробка ключових алгоритмів є невід'ємною частиною процесу розробки.

У цьому розділі будуть розглянуті основні алгоритми, необхідні для належної роботи додатку "Neroaroma". Особлива увага буде приділена процесу додавання інформації про нове замовлення у ресторані. Цей процес є одним із ключових для функціональності додатку та вимагає ретельної розробки та реалізації відповідних алгоритмів.

На рисунку 3.2 наведена блок-схема, яка ілюструє послідовність операцій при додаванні інформації про нове замовлення у ресторані. Ця блок-схема слугує візуальним засобом для розуміння та аналізу процесу та послідовності виконання операцій. Кожен блок блок-схеми відображає певну

дію або операцію, а стрілки з'єднують блоки, показуючи логічну послідовність виконання.

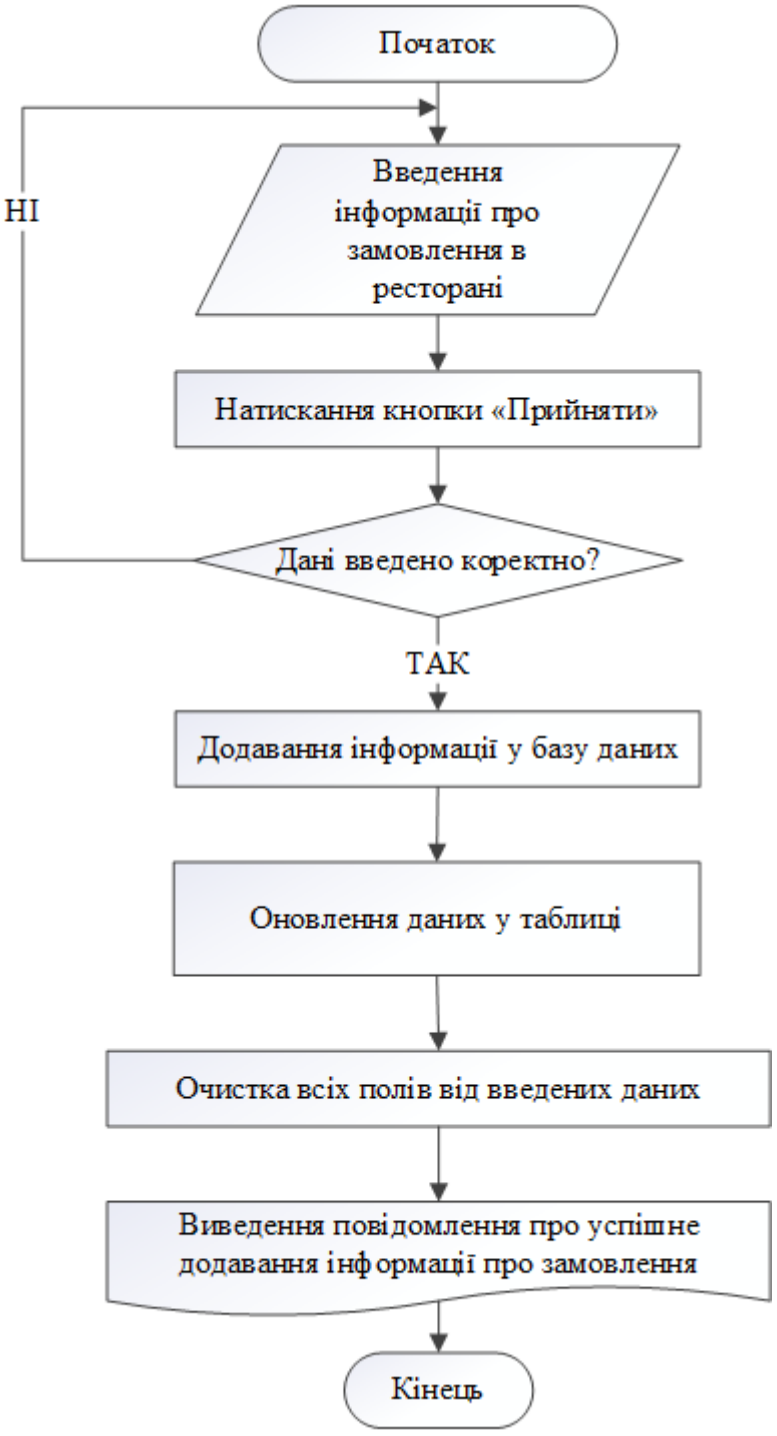

Рисунок 3.2 – Додавання інформації про нове замовлення

На рис. 3.3 показана блок-схема редагування інформації працівника ресторану із таблиці та виведення інформації про всіх працівників, інформація про яких зберігається у базі даних.

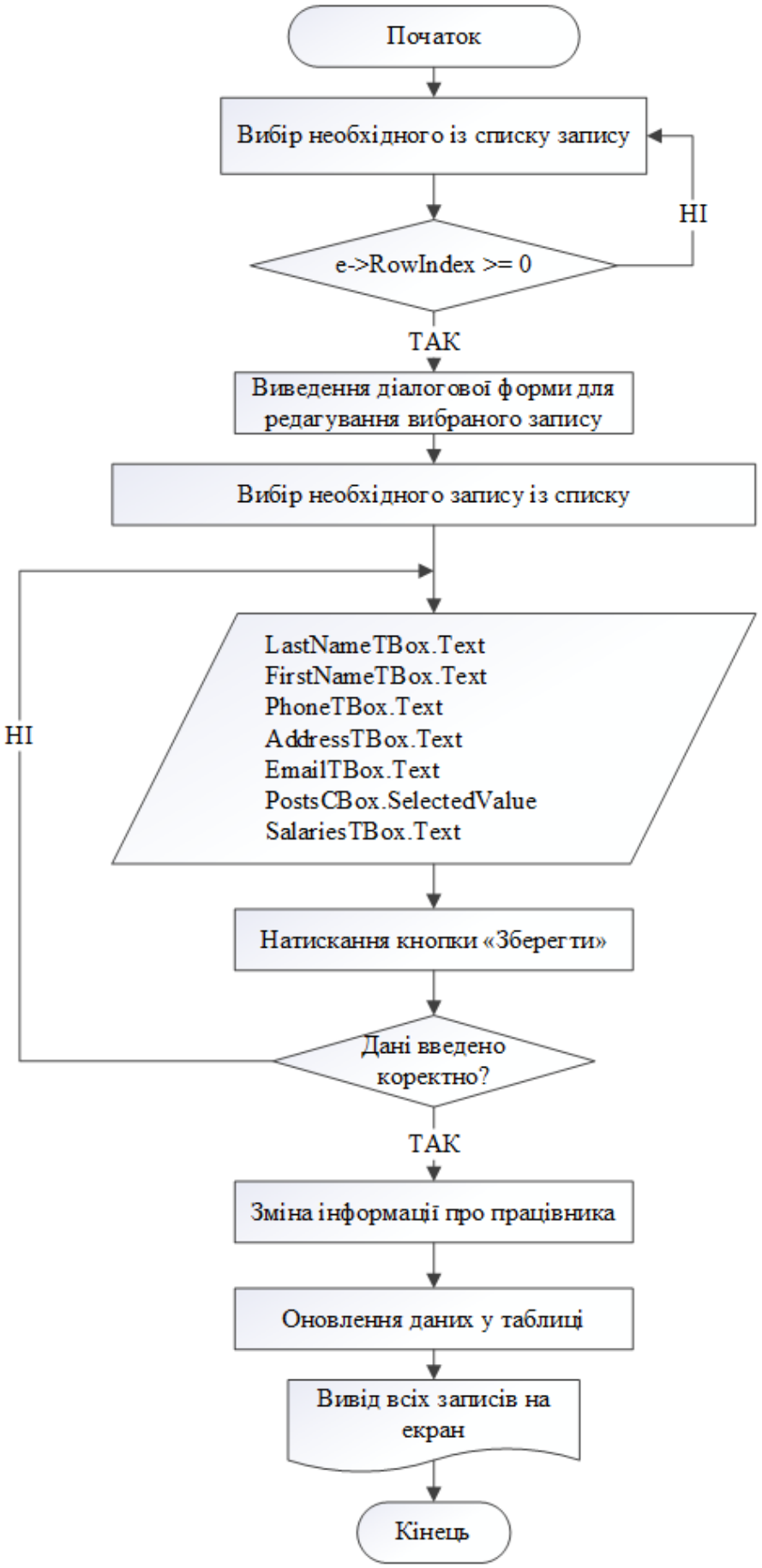

Рисунок 3.3 – Редагування інформації вибраного працівника

Рис. 3.4 відображає блок-схему видалення вибраного запису із бази даних та виведення інформації про всіх інших працівників ресторану.

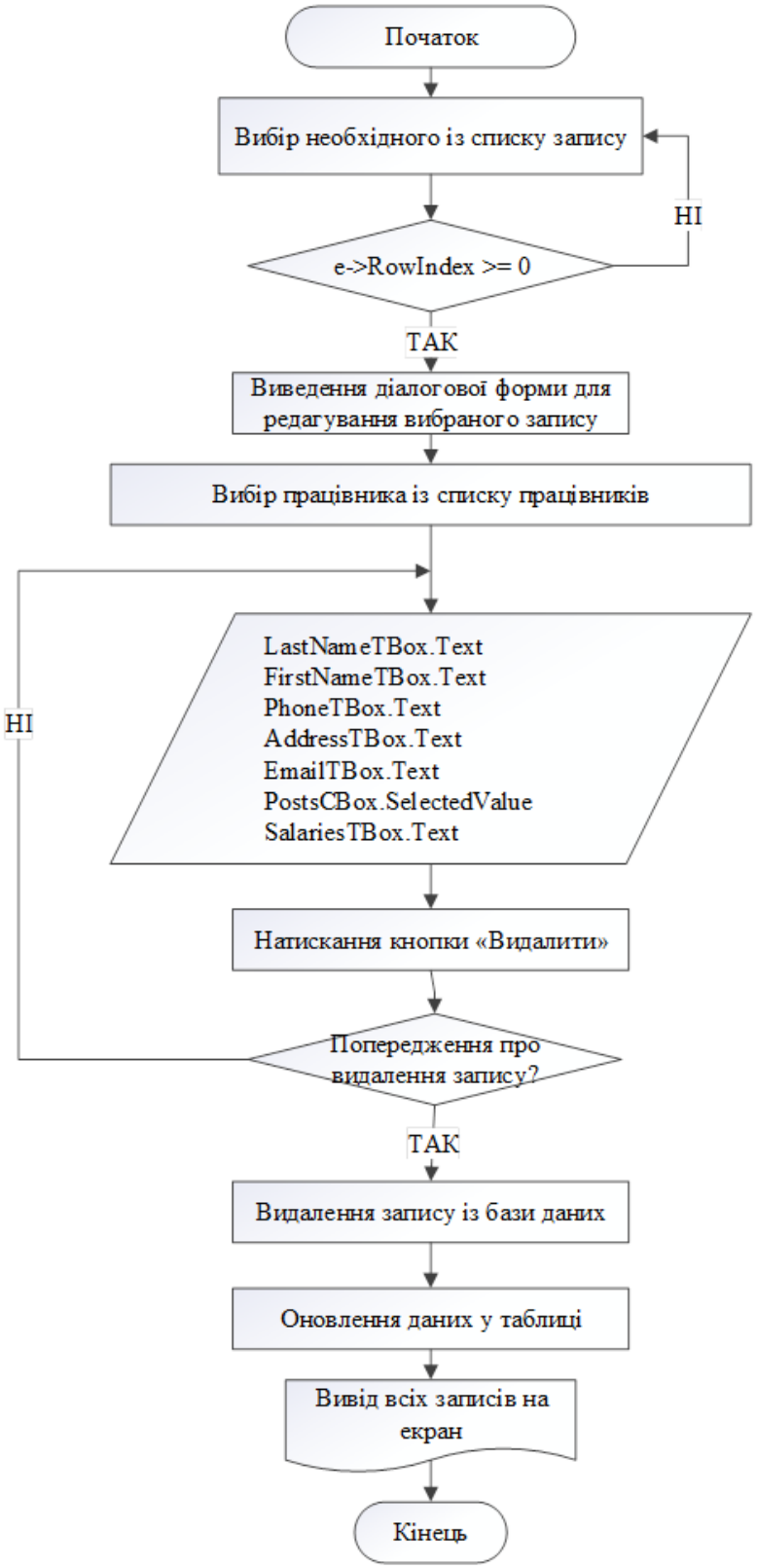

Рисунок 3.4 – Видалення інформації вибраного запису

3.3 Розробка програмних модулів

Після успішного створення бази даних, наступним кроком є підключення її до системи за допомогою розробкового середовища Visual Studio 2019. Цей процес дозволяє забезпечити доступ до таблиць бази даних з форм додатку, що надає користувачам можливість додавати, редагувати та видаляти дані. Для здійснення підключення, необхідно налаштувати змінну "CONNECT" у файлі проекту "App.config" та встановити параметри, які можна ознайомитися з ними на рисунку 3.5.

У файлі проекту "App.config" знаходиться конфігураційний файл, який містить параметри та налаштування, необхідні для правильної роботи додатку. Для підключення до бази даних, необхідно створити змінну з назвою "CONNECT" у цьому файлі. Значення цієї змінної визначає параметри підключення до бази даних, такі як адреса сервера бази даних, назва бази даних, ім'я користувача та пароль.

На рис. 3.5 показані параметри, які потрібно вказати для змінної "CONNECT". Це включає адресу сервера бази даних, її назву, ім'я користувача та пароль. Ці значення повинні бути налаштовані відповідно до конфігурації бази даних, щоб забезпечити успішне підключення до неї з додатку.

```
<appSettings>
<!-- Підключення до бази даних -->
<add key="CONNECT" value="Provider=Microsoft.Jet.OLEDB.4.0;
     Data Source=|DataDirectory|\DataBase.mdb;"/>
</appSettings>
```

Рисунок 3.5 – Змінна із параметрами налаштування бази даних

Для ефективної роботи з базою даних у проекті "Neroaroma" використовується простір імен System.Data.OleDb. Цей простір містить набір класів, які забезпечують зв'язок з базою даних за допомогою технології

OleDbConnection. Клас OleDbConnection дозволяє встановлювати з'єднання з базою даних, виконувати SQL-запити та зберігати зміни.

Створення меню в проекті "Neroaroma" було зроблено з допомогою елементу menuStrip. Цей елемент надає можливість користувачам вибирати різні опції та викликати функції програми через простий інтерфейс. Використання menuStrip спрощує навігацію користувачів та забезпечує зручний спосіб взаємодії з функціоналом додатку.

На рисунку 3.6 показано результат використання menuStrip в проекті "Neroaroma". Цей інтерфейсний елемент додається до головного вікна програми та відображається в верхній частині вікна. Він містить різні пункти меню, які можуть бути обраними користувачем для виклику певних функцій або опцій додатку.

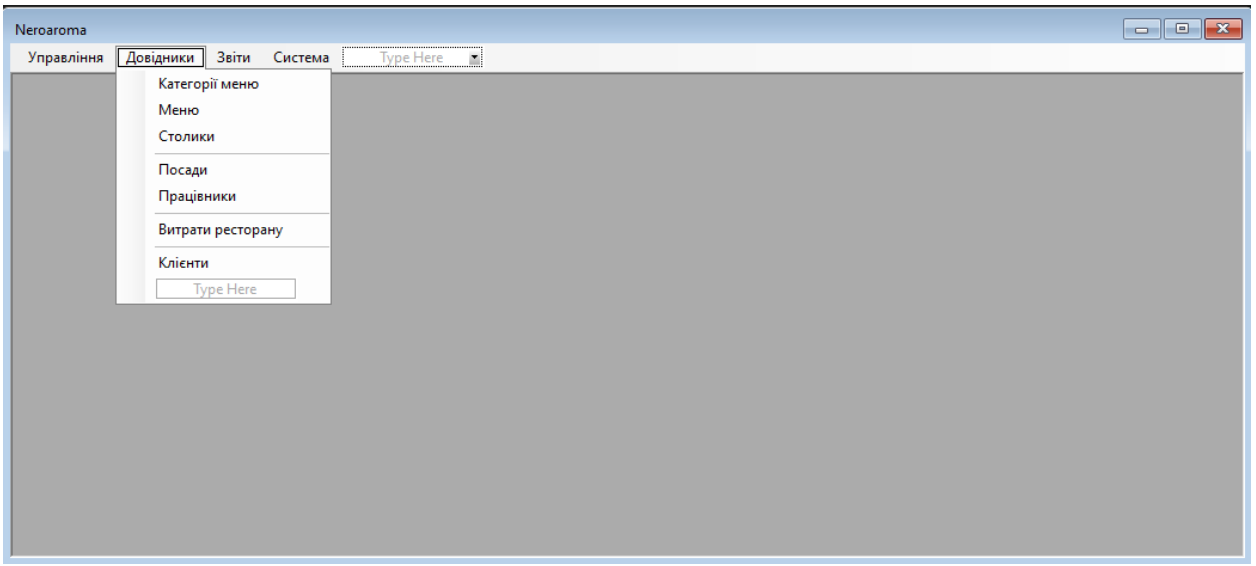

Рисунок 3.6 – Додавання головного меню

Для кожного пункту меню в додатку "Neroaroma" був доданий відповідний код, який створює новий екземпляр форми при його виборі. Цей код ініціалізує форму та відкриває її, зробивши її активною для користувача. Перед відкриттям нової форми також був доданий код, який забезпечує закриття попередньо відкритого вікна, щоб уникнути конфліктів та забезпечити коректне відображення вікон.

Аналогічний код застосовується для всіх пунктів меню в додатку "Neroaroma", щоб забезпечити їх правильну роботу та взаємозв'язок з відповідними формами. Код дозволяє відкривати необхідну форму, коли користувач обирає певний пункт меню, та замикати попереднє вікно, щоб уникнути накладання форм або конфліктів відображення.

На рис. 3.7 наведено приклад такого коду для одного з пунктів меню. Цей код виконується при виборі пункту меню і створює новий екземпляр форми "SomeForm", відкриває його та закриває попереднє вікно, якщо таке існує. Це забезпечує правильну роботу та відображення форм в додатку "Neroaroma".

```
private void замовленняToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  OrdersForm ordersForm = new OrdersForm();
  ordersForm.MdiParent = this;
  ordersForm.WindowState = FormWindowState.Maximized;
  ordersForm.Show();
}
```

Рисунок 3.7 – Код пунктів меню

Розроблено класи, які відповідають за взаємодію з базою даних в додатку "Neroaroma". Нижче наведено часткові реалізації методів з детальним описом для декількох класів. Наприклад, для додавання інформації про замовлення у базу даних був створений метод з назвою "InsertOrders".

```
1 reference
public void InsertOrders(DateTime StartDate, int ClientsId, int WorkersId, int
  RestaurantTablesId, string Description) {
  string SqlString = "INSERT INTO Orders (StartDate, ClientsId, WorkersId, " +
    "RestaurantTablesId, Description" +
    ") Values(?, ?, ?, ?, ?)";
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("StartDate", StartDate.ToString());
      cmd.Parameters.AddWithValue("ClientsId", ClientsId);
      cmd.Parameters.AddWithValue("WorkersId", WorkersId);
      cmd.Parameters.AddWithValue("RestaurantTablesId", RestaurantTablesId);
      cmd.Parameters.AddWithValue("Description", Description);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рисунок 3.8 – Код методу «InsertOrders»

Метод InsertOrders, виконує збереження інформації про нове замовлення в таблиці Orders бази даних. Ця таблиця містить рядки, які представляють окремі замовлення в ресторані. Кожен рядок таблиці містить такі дані, як дата початку замовлення, ідентифікатор (ID) клієнта, ID працівника, ID столика та опис замовлення.

Збереження цих даних дозволяє системі ефективно відстежувати та управляти замовленнями в ресторані. Наприклад, за допомогою цих даних можна швидко знайти замовлення, які належать певному клієнту або столику. Також ці дані дозволяють формувати звіти про діяльність ресторану, такі як витрати та доходи від замовлень. Звіти цієї природи надають цінну інформацію про ефективність ресторанного бізнесу та допомагають у прийнятті управлінських рішень.

Цей метод виконує вставку нового рядка в таблицю Orders, заповнюючи відповідні поля значеннями, переданими в параметрах методу. Таким чином, дані про нове замовлення зберігаються у базі даних, що дозволяє подальшу роботу з ними та забезпечує функціональність додатку "Neroaroma" у відстеженні та управлінні замовленнями ресторану.

Для видалення даних про замовлення, також був розроблений метод, код якого представлено на рис. 3.9.

```
public void DeleteOrdersByOrdersId(int OrdersId) {
  string SqlString = "DELETE FROM Orders WHERE OrdersId=" + OrdersId.ToString();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рисунок 3.9 – Код методу «DeleteOrdersByOrdersId»

В процесі роботи цей метод перевіряє наявність замовлення з вказаним ідентифікатором у базі даних. Якщо замовлення з таким ідентифікатором існує, то воно видаляється з таблиці Orders, а всі пов'язані з ним дані також видаляються. Це забезпечує коректність та цілісність даних у базі даних.

Видалення даних про замовлення є важливою функціональністю для системи Neroaroma, оскільки дозволяє користувачам видаляти замовлення, які більше не потрібні або є помилковими. Цей метод допомагає забезпечити ефективне управління замовленнями та збереження актуальних даних в базі даних.

Після завершення розробки класів рівня даних, було проведено розробку форм, що забезпечують взаємодію користувача з програмою. Однією з таких розроблених форм є форма для опрацювання даних працівників ресторану. Ця форма відображається на екрані користувача і надає зручний інтерфейс для внесення, редагування та видалення інформації про працівників (рис. 3.10).

Форма для опрацювання даних працівників ресторану містить різні елементи керування, такі як текстові поля для введення особистих даних працівника, кнопки для збереження змін, видалення запису, а також таблицю для відображення списку працівників та їхньої інформації. Користувач може виконувати різні операції, такі як додавання нового працівника, редагування існуючого запису чи видалення працівника зі списку.

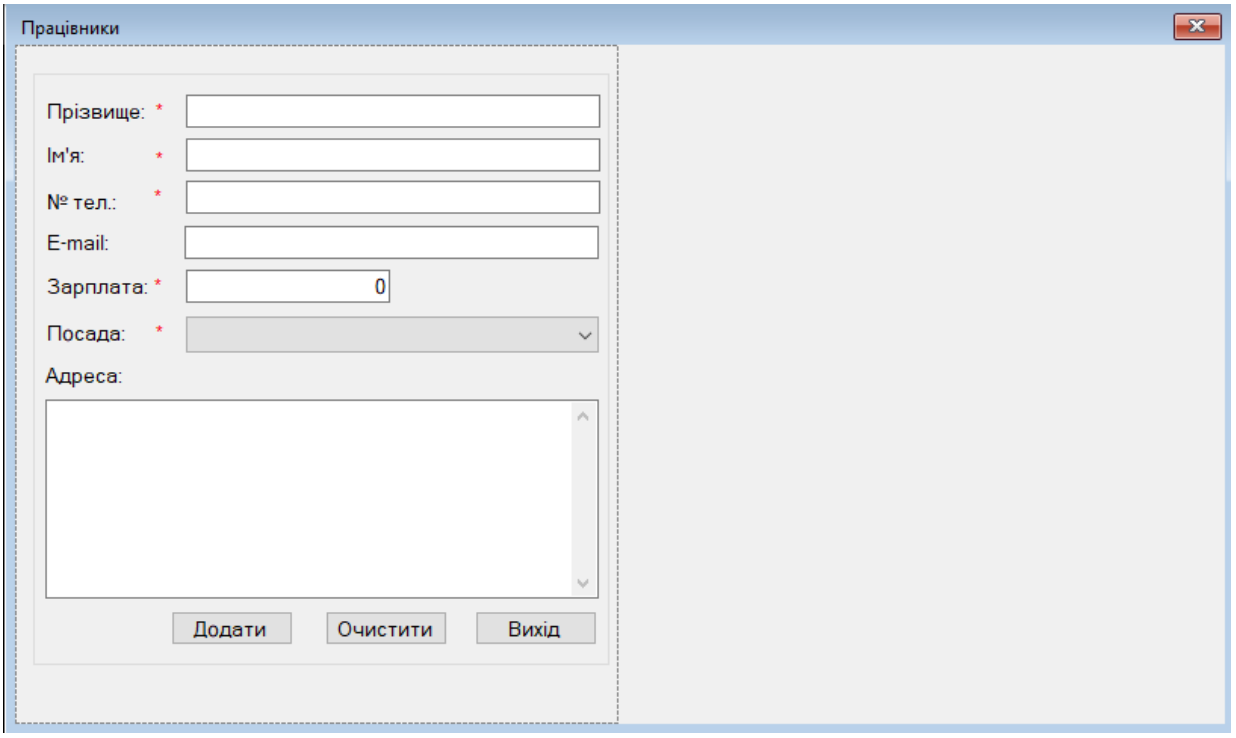

Рисунок 3.10 – Розробка форми опрацювання даних працівників ресторану

При завантаженні форми, в її конструкторі викликається метод "LoadAllData". Цей метод відповідає за завантаження всіх необхідних даних для коректної роботи форми. Одним з етапів цього завантаження є заповнення випадаючого списку (ComboBox) зі списком посад ресторанного бізнесу (рис. 3.11).

```
1 reference
private void LoadAllDate() {
    _PostsList = _PostsProvider.GetAllPosts();
    PostsCBox.DataSource = _PostsList;
    PostsCBox.ValueMember = "PostsId";
    PostsCBox.DisplayMember = "PostsName";
}
```

Рисунок 3.11 – Код методу «LoadAllDate»

Метод "LoadAllData" виконує запит до бази даних, отримує список посад ресторану та передає цей список до випадаючого списку на формі. Кожен елемент списку представляє собою окрему посаду, яку можна вибрати користувачем.

Завантаження посад ресторанного бізнесу до випадаючого списку дозволяє користувачу легко вибрати необхідну посаду для працівника без

необхідності введення тексту вручну. Це полегшує взаємодію з програмою та допомагає уникнути можливих помилок при вводі даних.

У розроблених формах, крім завантаження даних, також виконується перевірка коректності введених даних. Кожна форма містить власний метод з назвою "IsDataEnteringCorrect", який відповідає за перевірку даних у всіх обов'язкових полях.

Метод "IsDataEnteringCorrect" для форми "WorkersForm" має вигляд, що показаний на рис. 3.12. В цьому методі здійснюється перевірка коректності введених даних, зокрема для обов'язкових полів. У перевірці використовуються різні правила та умови, що залежать від вимог проекту та типу даних.

```
1 reference
private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataEntering(LastNameTBox.Text)) {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(FirstNameTBox.Text)) {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(PhoneTBox.Text)) {
    PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(SalariesTBox.Text)) {
    SalariesValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    SalariesValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (Convert.ToInt32(PostsCBox.SelectedValue) > 0) {
    PostsValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PostsValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  return isCorrect;
}
```

Рисунок 3.12 – Код методу для перевірки коректності введених даних

Якщо дані введені коректно, метод повертає значення true, що свідчить про те, що дані введені правильно. У разі виявлення помилок або неправильних даних, метод виводить повідомлення про помилку або неправильність та повертає значення false. Це дозволяє користувачу виправити помилки та введені дані до того, як вони будуть збережені у базі даних.

Перевірка коректності даних у формах є важливим етапом, оскільки дозволяє забезпечити введення лише валідних та правильних даних у систему, що покращує її надійність та використовуваність.

У формі "UpdateWorkersForm" реалізована подія "DeleteBtn_Click", яка відповідає за видалення даних про працівника з бази даних. Код даної події представлений на рисунку 3.13.

```
1 reference
private void DeleteBtn_Click(object sender, EventArgs e) {
  if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити", MessageBoxButtons.YesNo)
    == DialogResult.Yes) {
    _WorkersProvider.DeleteWorkersByWorkersId(_WorkersId);
    this.Close();
  }
}
```

Рисунок 3.13 – Код події «DeleteBtn_Click»

При виклику події "DeleteBtn_Click" спочатку перевіряється, чи був вибраний працівник зі списку працівників. Якщо вибір був здійснений, то отримується ідентифікатор працівника та виконується запит до бази даних для видалення відповідного запису про працівника. Після виконання запиту інформація оновлюється у відображенні списку працівників на формі.

Цей код забезпечує можливість користувачу видаляти працівників з бази даних у формі "WorkersForm". При видаленні працівника, його дані будуть видалені з бази даних, що дозволяє підтримувати актуальність інформації та оновлювати список працівників у системі.

В даному підрозділі було надано частковий опис системи, яка була реалізована. Розглянуті були різні аспекти розробки, такі як проектування бази даних, вибір технологій, розробка ключових алгоритмів, створення форм для взаємодії з користувачем та реалізація функціональності для додавання, редагування та видалення даних.

3.4 Функціональне та модульне тестування

Тестування є невід'ємною складовою розробки програмного забезпечення і відіграє важливу роль у забезпеченні якості та надійності системи. У контексті розробки додатку "Система обліку витрат в ресторанній діяльності" було проведено функціональне тестування з метою перевірки основних функцій додатку та визначення його відповідності вимогам користувачів.

Під час тестування були створені тест-кейси, які включали набір тестових сценаріїв для перевірки різних аспектів функціональності додатку. Ці тест-кейси враховували функціональні вимоги, що ставилися до системи, і дозволяли перевірити, чи виконує додаток очікувані операції та повертає правильні результати. Під час тестування проводилась перевірка таких аспектів, як взаємодія з базою даних, обробка даних користувачів, збереження і відновлення даних, а також взаємодія з інтерфейсом користувача.

Метою функціонального тестування було забезпечити стабільну та надійну роботу додатку. Під час тестування виявлялися можливі недоліки, помилки та некоректна робота системи. Ці виявлені проблеми дозволяли розробникам вжити заходів для виправлення та вдосконалення функціональності додатку. Тестування також сприяло забезпеченню високої якості програмного забезпечення, зменшенню ризиків помилок та покращенню задоволеності користувачів.

Тест-кейс №1. Вхід до системи. Перевірка на наявність правильної авторизації користувача:

– відкрити форму авторизації;
– ввести існуючий логін та пароль користувача;
– перевірити, що користувач має доступ до головної форми застосунку.

В результаті, користувач має мати доступ до головної форми застосунку.

Тест-кейс №2. Додавання нового запису про працівника ресторану. Перевірка на правильність заповнення полів:

– відкрити форму додавання нового запису про працівника;
– заповнити всі обов'язкові поля (ім'я, прізвище, посада тощо);
– натиснути кнопку "Додати";
– перевірити, що новий запис про працівника збережений у базі даних.

В результаті, запис про працівника має бути доданий в базу даних та відображається у списку всіх працівників.

Тест-кейс №3. Редагування запису працівника. Перевірка на правильність редагування полів:

– вибрати необхідний запис про працівника;
– змінити значення одного з полів (наприклад, прізвище);
– натиснути кнопку "Зберегти";
– перевірити, що відредагований запис про працівника збережений у БД.

В результаті, запис про працівника має бути відредагований в базі даних та відображається у списку всіх записів про працівників.

Тест-кейс №4. Звітність за вибраний період часу. Перевірка на правильність відображення даних:

– відкрити форму звітності із назвою «За період часу»;

– вибрати період часу (наприклад, останній місяць);

– натиснути кнопку "Формувати";

– перевірити, що у вікні виводяться всі дані, які були успішно додані за обраний період часу.

В результаті формується звіт, що містить правильну та повну інформацію про проведені замовлення в ресторані за вказану дату.

Тест-кейс №5. Звітність по загальним витратам за вибраний період часу. Перевірка на правильність відображення даних:

– відкрити форму із назвою «Загальні витрати за вибраний період часу»;

– вибрати період часу (наприклад, останній 2 останні місяці);

– натиснути кнопку "Формувати";

– перевірити, що у вікні виводяться всі дані та правильно рахується сума.

В результаті формується звіт, що містить повну інформацію про витрати ресторану за два місяці часу.

Тест-кейс №6. Видалення запису про працівника. Перевірка на правильність видалення:

– відкрити форму видалення запису про працівника;

– вибрати працівника, якого потрібно видалити;

– натиснути кнопку "Видалити";

– перевірити, що обраний запис про працівника був успішно видалений з бази даних ресторану.

В результаті, інформація про працівника має бути видалена коректно та більше не відображається у списку з іншими працівниками.

Після завершення функціонального тестування системи, наступним етапом є модульне тестування, яке є важливою складовою розробки програмного забезпечення. Модульне тестування спрямоване на перевірку окремих модулів або компонентів програми для забезпечення їх відповідності вимогам та специфікації.

Для проведення модульного тестування був створений новий проект типу "Unit Test Project" в середовищі Visual Studio. Цей проект містить набір тестів, які перевіряють роботу окремих функцій або методів програми. Тести були розроблені згідно з вимогами та очікуваннями щодо функціональності кожного модуля.

Під час модульного тестування використовуються різні тестові дані та сценарії для перевірки різних можливих ситуацій та роботи програми. Тестові дані обираються таким чином, щоб вони покривали різні варіанти вхідних даних та різні шляхи виконання програми.

Після написання тестів, їх було запущено, і результати були проаналізовані. Якщо всі тести пройшли успішно, це свідчить про те, що модулі програми працюють вірно та відповідають очікуванням. У разі виявлення проблем або невідповідності вимогам, розробники здійснюють необхідні виправлення та вдосконалення.

Модульне тестування дозволяє виявляти можливі помилки та проблеми в роботі окремих модулів програми та вчасно їх виправляти. Воно сприяє поліпшенню якості програмного забезпечення та забезпечує його стабільну та надійну роботу. Результати модульного тестування є важливою інформацією для подальшого вдосконалення системи та забезпечення задоволеності користувачів.

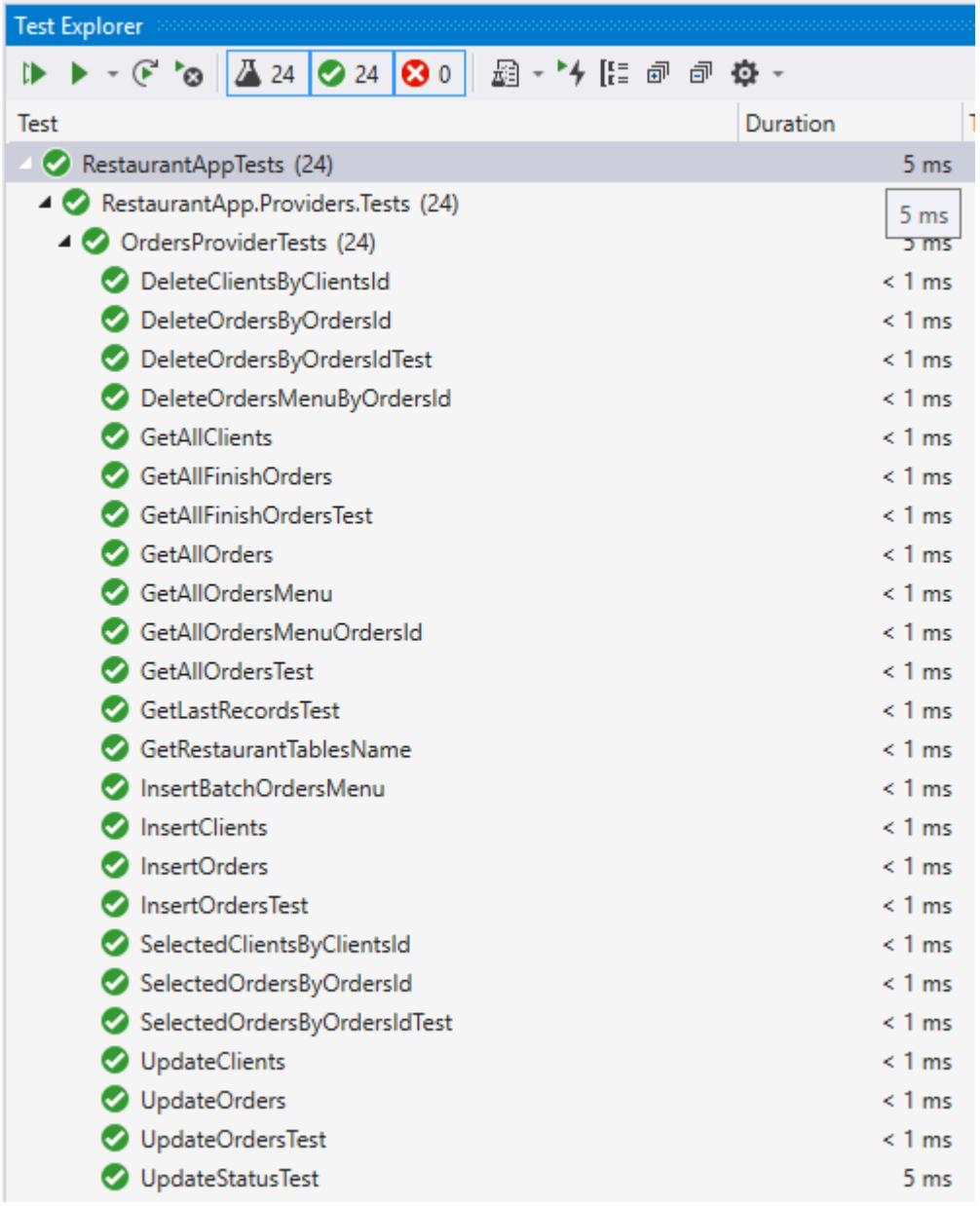

Рисунок 3.14 – Результати проведення модульного тестування

3.5 Інструкція користувача

Після запуску додатку "Neroaroma", користувачу буде відображено вікно вхідної аутентифікації. В цьому вікні користувачу потрібно ввести своє ім'я та пароль для доступу до функціоналу програми (рис. 3.15). Інтерфейс вхідної форми надає зручний спосіб авторизації, що дозволяє контролювати доступ користувачів до системи та забезпечує безпеку даних. Після успішної аутентифікації користувач матиме можливість використовувати всі функції та опції, які надає додаток "Neroaroma".

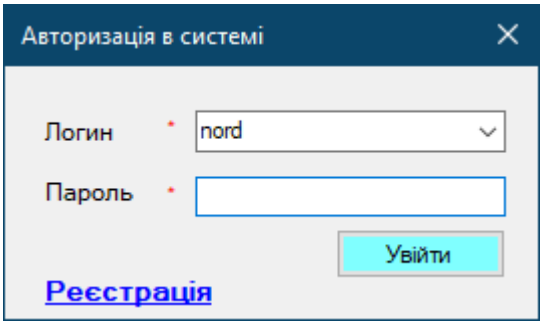

Рисунок 3.15 – Ідентифікація користувача в системі

У разі, якщо користувач введе некоректну інформацію під час процесу аутентифікації, програма "Neroaroma" надасть відповідну зворотну інформацію. На екрані буде виведено повідомлення, яке повідомляє користувача про некоректні дані або помилку у введені інформації (рис. 3.16). Це повідомлення дозволяє користувачеві бути інформованим про проблему та змусить його виправити некоректну інформацію перед продовженням використання програми.

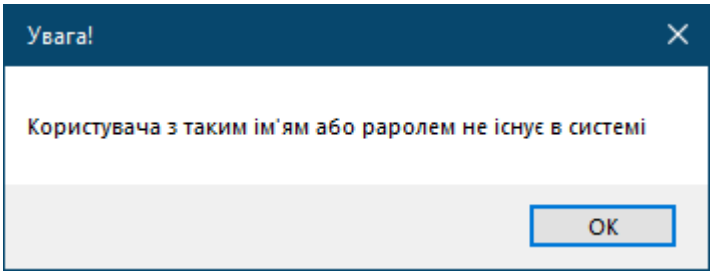

Рисунок 3.16 – Попередження про неправильне введення даних

З метою полегшення та прискорення процесу пошуку, програма "Neroaroma" надає можливість вибору імені з випадаючого списку або автоматично виділяє введене ім'я під час введення даних. Це забезпечує більш зручний та швидкий доступ до потрібної інформації. У разі, якщо користувач вводить ім'я з клавіатури, програма автоматично підкреслює відповідний варіант у випадаючому списку, розташовуючи його на першому місці для швидшого доступу (рис. 3.17). Це дозволяє забезпечити зручний та ефективний пошук імені в програмі "Neroaroma".

Після успішної ідентифікації користувача системи, програма "Neroaroma" відкриє головне вікно, яке містить основне меню і надає користувачеві доступ до різних функцій та опцій програми (рис. 3.17).

Головне меню містить різні пункти, які представляють собою можливості роботи з додатком, наприклад, перегляд і редагування даних, створення звітів, управління користувачами та інше. Користувач може обрати потрібний пункт меню, виконати відповідну дію та взаємодіяти з програмою у зручний для себе спосіб. Головне вікно є центральним елементом інтерфейсу, з якого користувач розпочинає роботу з програмою та отримує доступ до всіх її можливостей.

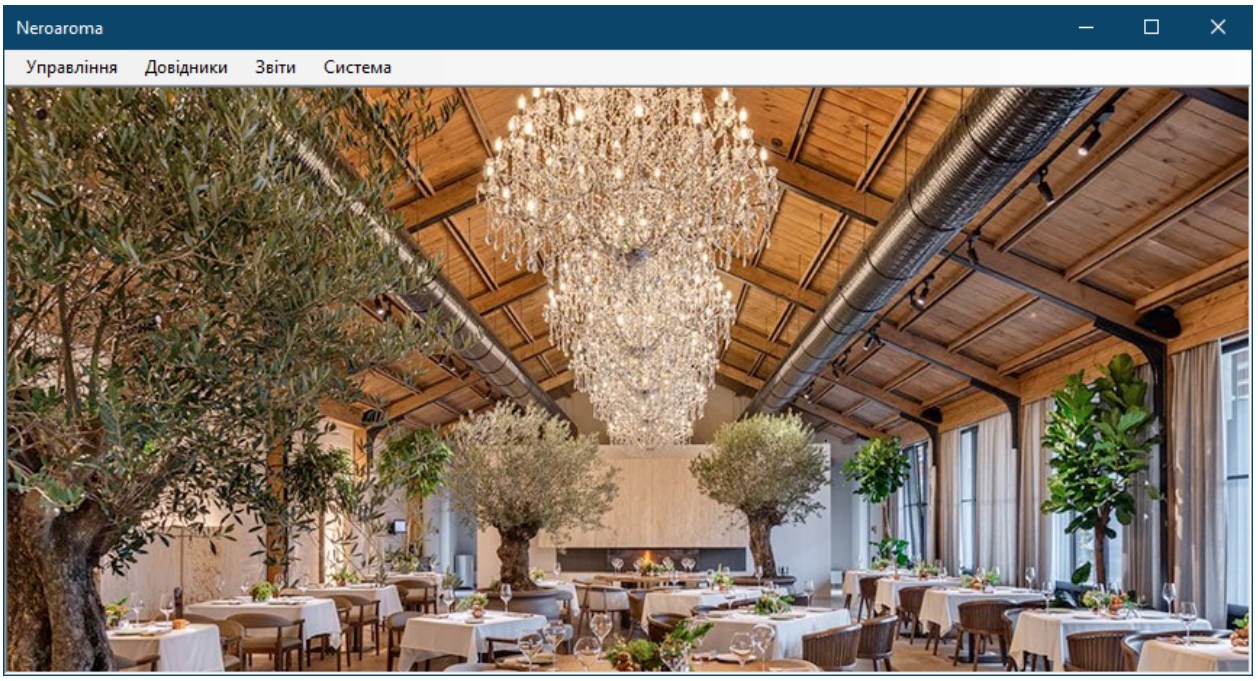

Рис. 3.17 – Головне меню програми

Для того, щоб приймати замовлення від клієнтів, необхідно належним чином заповнити інформаційну систему даними про клієнтів, співробітників, столики та меню. Щоб розпочати цей процес, необхідно перейти до відповідного меню програми, яке називається "Довідники", і обрати опцію "Клієнти".

У вікні, яке з'явиться (рис. 3.18), вам буде запропоновано ввести всю необхідну інформацію про клієнта, таку як прізвище, ім'я, контактні дані та інші відомості. Після введення даних ви можете натиснути кнопку "Додати", щоб додати нового клієнта до системи. Якщо операція пройде успішно, новий клієнт з'явиться у правій частині екрану у списку всіх клієнтів, підтверджуючи успішне додавання інформації про нього до системи.

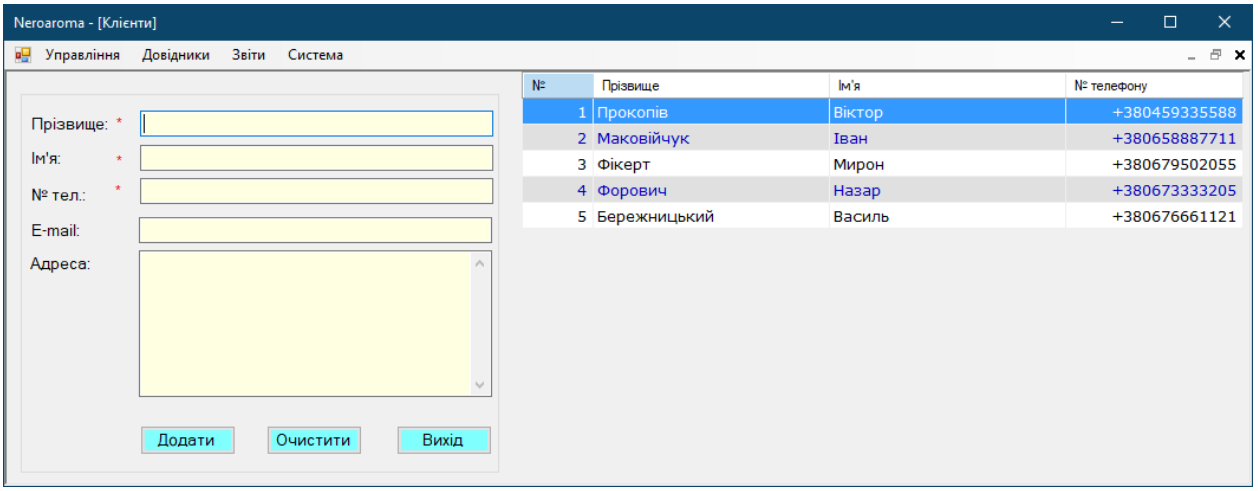

Рисунок 3.18 – Вікно для опрацювання даних про клієнтів

Реалізована можливість редагування та видалення даних про клієнтів для забезпечення більшої гнучкості та можливості оновлення інформації. У вікні редагування (зображено на рисунку 3.19) знаходяться всі поля, пов'язані з інформацією про клієнта, які можна змінювати. Користувач має можливість ввести нові дані або виправити існуючі. Після внесення необхідних змін користувач може зберегти зміни, натиснувши кнопку "Зберегти".

У вікні редагування також присутня кнопка "Видалити", яку користувач може використати для повного видалення запису про клієнта з системи. Перед виконанням видалення система запитає у користувача підтвердження цієї операції, щоб переконатися, що користувач бажає продовжити з видаленням.

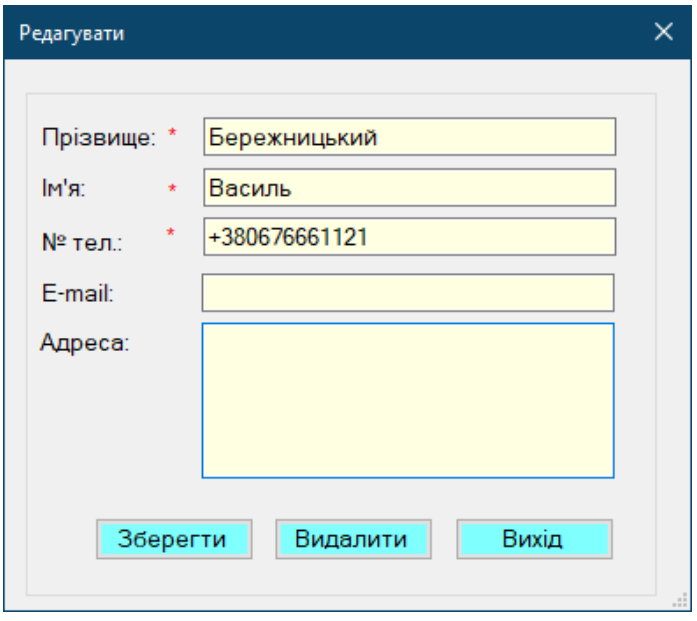

Рисунок 3.19 – Вікно для редагування даних

Для додавання даних про співробітників в базу необхідно скористатися вікном для опрацювання даних про співробітників, яке зображено на рис. 3.20. В цьому вікні користувач може ввести всю необхідну інформацію про співробітника, таку як прізвище, ім'я, посада, контактні дані та інші відомості. Після введення всіх необхідних даних користувач має можливість натиснути кнопку "Додати", щоб зберегти нового співробітника до бази даних.

У вікні опрацювання даних про співробітників також реалізована можливість редагування та видалення існуючих записів. Кнопка "Зберегти" дозволяє користувачу внести зміни до даних співробітника, а кнопка "Видалити" - видалити вибраний запис про співробітника з бази даних. Після внесення змін або видалення система може повідомити користувача про результат операції.

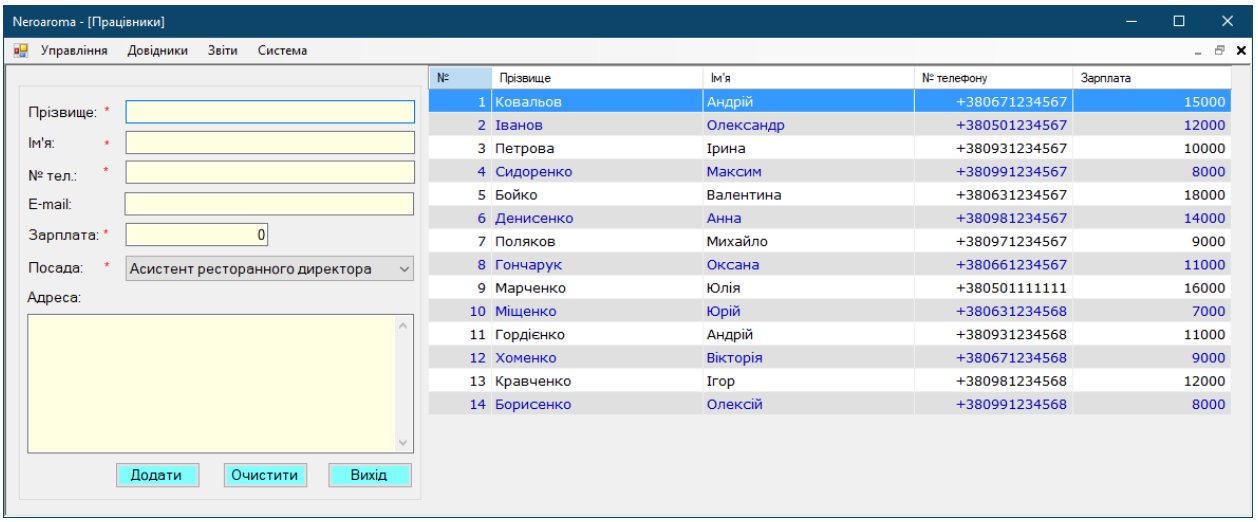

Рисунок. 3.20 – Вікно для опрацювання даних про співробітників

Для редагування даних про співробітників, користувач може вибрати необхідного співробітника у правій частині екрану шляхом натискання лівої кнопки мишки на відповідному записі. Після цього відкриється вікно для редагування даних вибраного співробітника, яке зображено на рис. 3.21.

У вікні редагування даних користувач може внести необхідні зміни до інформації про співробітника, такі як зміна прізвища, імені, посади,

контактних даних та інших відомостей. Після внесення змін користувач має можливість натиснути кнопку "Зберегти", щоб зберегти внесені зміни до бази даних.

Також у вікні редагування даних реалізована функціональність видалення запису про співробітника. Для цього передбачена окрема кнопка "Видалити", яка дозволяє користувачу видалити вибраний запис про співробітника з бази даних.

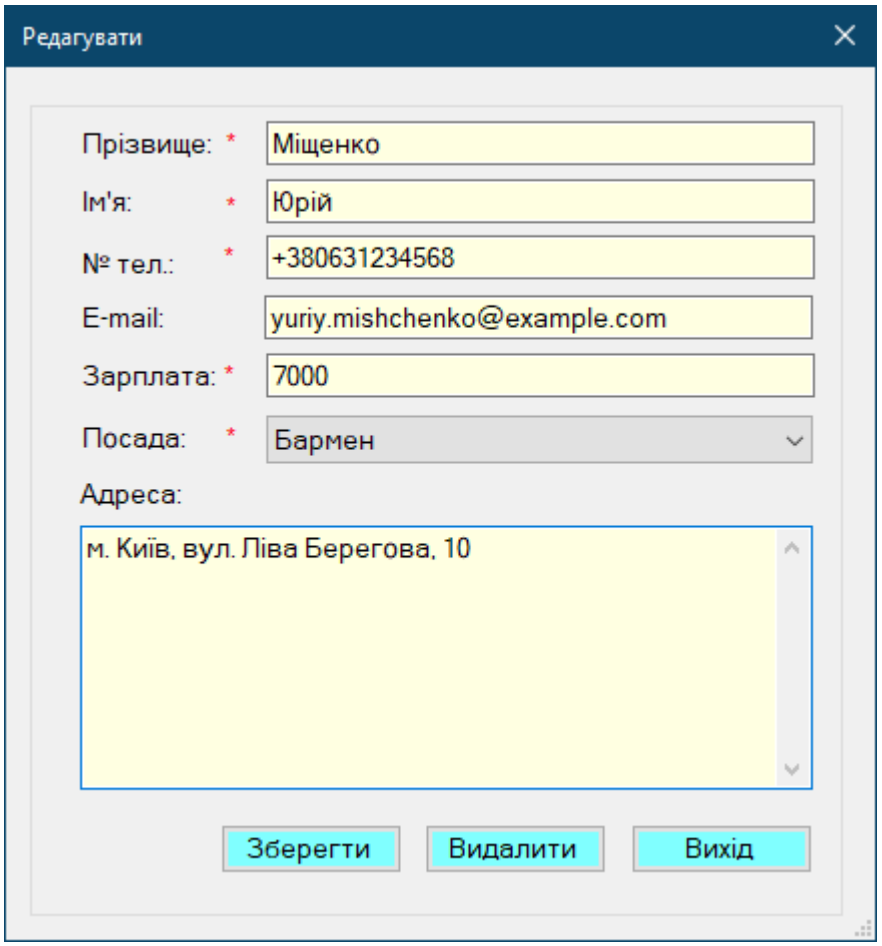

Рисунок 3.21 – Вікно для редагування даних вибраного співробітника

З метою додавання нових столиків у програмі, користувачу необхідно виконати наступні кроки. Спочатку він повинен перейти до пункту меню під назвою "Довідники", а після цього обрати пункт "Столики". В результаті цих дій з'явиться відповідне вікно, яке зображено на рисунку 3.22.

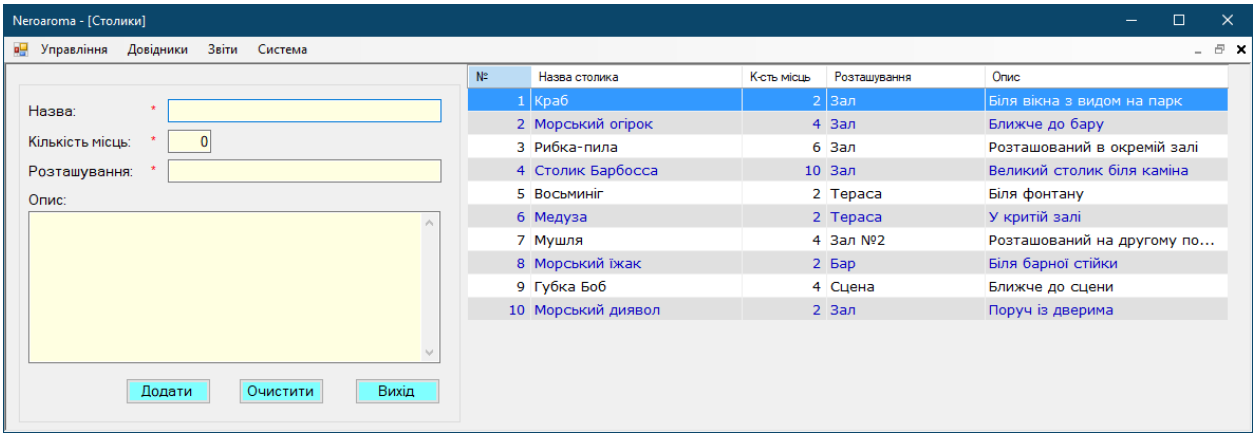

Рисунок 3.22 – Вікно для опрацювання даних про столики

У вікні додавання столиків користувач може ввести необхідну інформацію про новий столик, зокрема номер столика, кількість місць, опис та інші характеристики. Після введення відповідних даних, користувач може натиснути кнопку "Додати", щоб зберегти інформацію про новий столик у базі даних.

Для редагування даних про столики у програмі, користувачу необхідно виконати наступні дії. Спочатку, в правій частині вікна, він повинен вибрати потрібний столик шляхом натискання лівої кнопки миші на відповідний запис. Після цього з'явиться вікно для редагування даних вибраного столика, яке зображено на рисунку 3.23.

У вікні редагування столика користувач може змінити існуючі дані про столик, наприклад, змінити номер столика, кількість місць або опис. Після внесення необхідних змін, користувач може натиснути кнопку "Зберегти" для оновлення інформації про столик у базі даних. Таким чином, користувач має можливість актуалізувати дані про столики згідно зі змінами, які відбулися в ресторані.

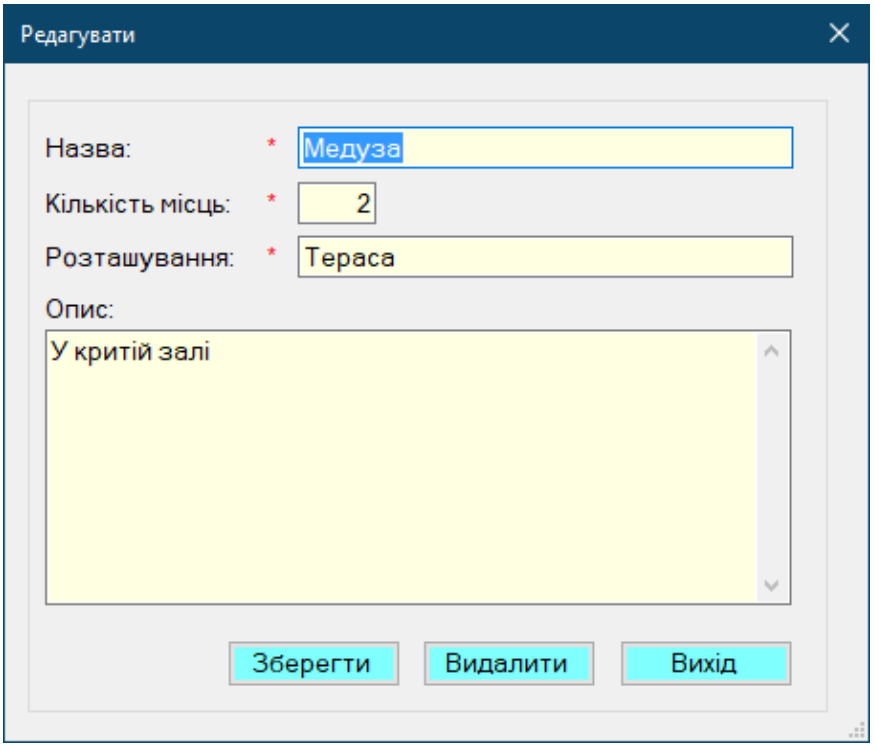

Рисунок 3.23 – Вікно для редагування даних столиків

Для роботи з категоріями меню, меню та посадами працівників у програмі, застосовується аналогічний підхід. Користувач може перейти до відповідного пункту меню, наприклад, "Довідники" -> "Категорії меню", "Довідники" -> "Меню" або "Довідники" -> "Посади працівників". Після вибору відповідного пункту меню, з'явиться вікно, де користувач може ввести, відредагувати або видалити відповідні дані.

Наприклад, для редагування категорій меню, користувач може вибрати потрібну категорію у правій частині вікна, натиснувши ліву кнопку миші на відповідний запис. Після цього з'явиться вікно редагування даних про категорію меню, де користувач може змінити назву, опис або інші характеристики категорії. Аналогічно, редагування меню та посад працівників відбувається за аналогічним принципом.

Такий підхід до роботи з категоріями меню, меню та посадами працівників дозволяє користувачам зручно та ефективно управляти цими даними в системі.

Для прийняття замовлення від клієнта в програмі "Neroaroma", користувачу необхідно виконати наступні кроки. Спочатку потрібно перейти до пункту меню "Управління" та обрати опцію "Прийняти замовлення". Після виконання цих дій відкриється вікно, яке відображає деталі замовлення та доступні опції (рис. 3.24).

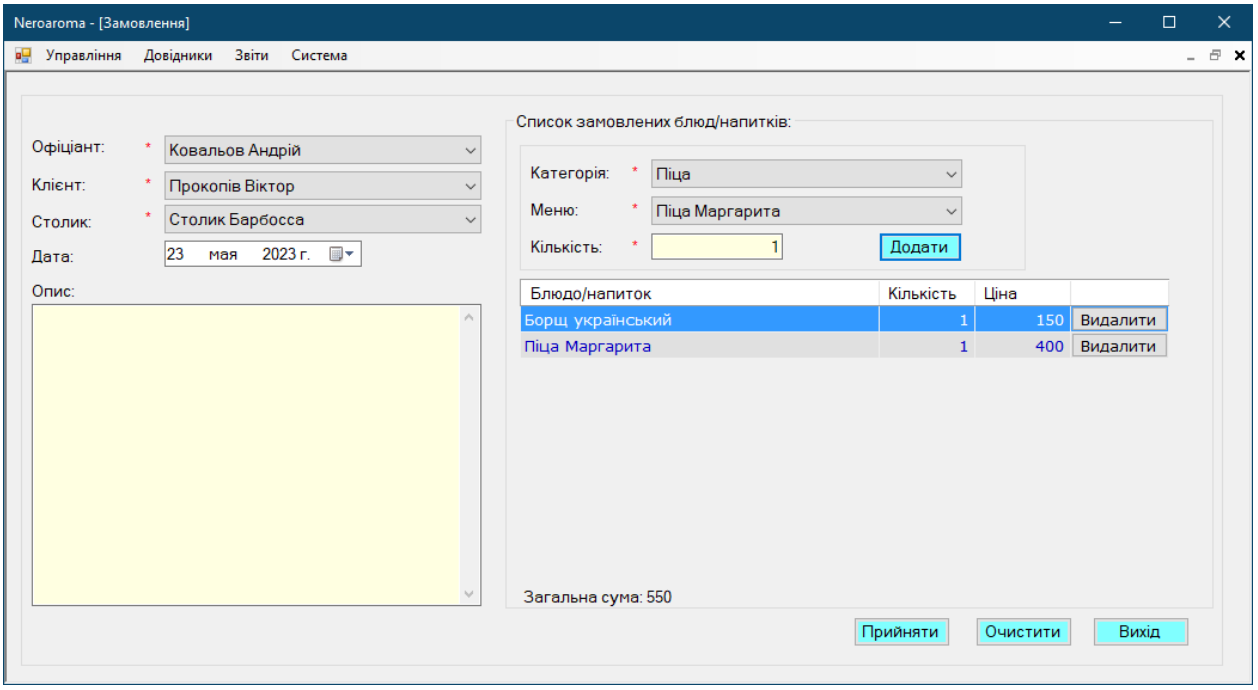

Рисунок 3.24 – Вікно для приймання замовлень

У вікні прийняття замовлення користувач може вибрати клієнта зі списку, вибрати столик, додати страви з меню та вказати їх кількість. Також можуть бути надані додаткові можливості, наприклад, вибір спеціальних приміток або побажань щодо замовлення. Після введення всіх необхідних даних користувач може натиснути кнопку "Підтвердити замовлення" для завершення процесу прийняття замовлення.

Ця функціональність дозволяє користувачам зручно та ефективно приймати замовлення в ресторанній системі, забезпечуючи точність та швидкість обробки замовлень.

Для закриття замовлення та звільнення столика в програмі "Neroaroma", користувачу необхідно виконати наступні кроки. Спочатку потрібно перейти до пункту меню "Управління" та обрати опцію "Відкриті замовлення". Після

виконання цих дій відкриється вікно, яке відображає список активних замовлень (рис. 3.25).

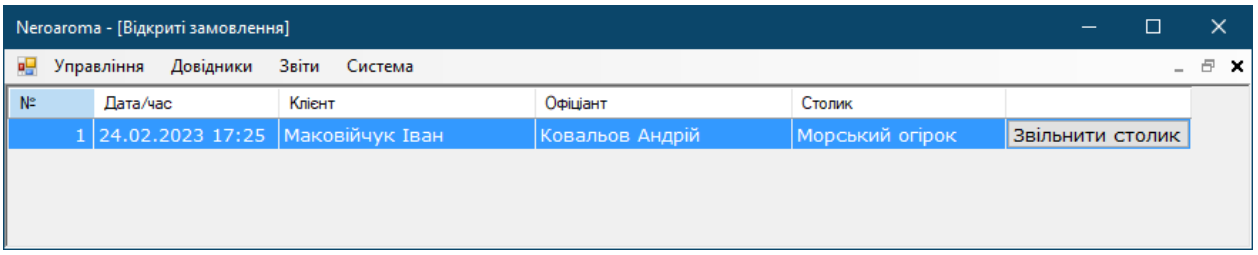

Рисунок 3.25 – Відкриті замовлення

У вікні "Відкриті замовлення" користувач може переглянути список замовлень, що ще не були закриті, та їх деталі. Для звільнення столика, пов'язаного з конкретним замовленням, користувач має натиснути кнопку "Звільнити столик", розташовану поряд з відповідним замовленням.

Ця функціональність дозволяє користувачам ефективно керувати активними замовленнями та швидко звільняти столики після їх закриття, забезпечуючи ефективне використання ресторанного простору.

Для перевірки завершених замовлень в програмі "Neroaroma", користувачу слід виконати наступні дії. По-перше, перейдіть до пункту меню "Управління" і оберіть опцію "Закриті замовлення". Після виконання цих дій відкриється вікно, що відображає список замовлень, які вже були завершені (рис. 3.26).

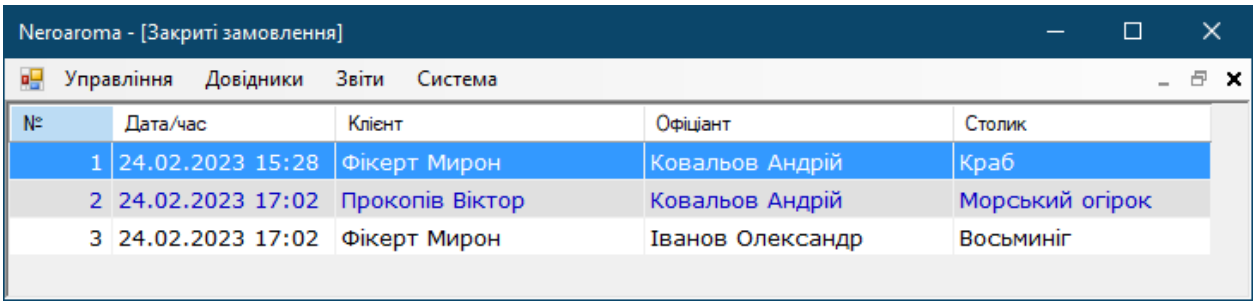

Рисунок 3.27 – Закриті замовлення

У вікні "Закриті замовлення" користувач може переглянути список завершених замовлень разом з детальною інформацією про них. Ця функція дозволяє зручно відстежувати і контролювати завершені замовлення, а також забезпечує можливість генерування звітів та аналізу діяльності ресторану на підставі історичних даних.

Перевірка завершених замовлень дозволяє ресторанному персоналу відслідковувати продажі, визначати популярність певних страв та ефективність роботи ресторану в цілому.

Після переходу до пункту меню "Звіти" та обрання опції "За період часу" в програмі "Neroaroma", користувачу відкривається можливість формувати звітність за вибраний період часу. Ця функція дозволяє отримати детальну інформацію про витрати, доходи та інші показники ресторанної діяльності за певний проміжок часу (рис. 3.28).

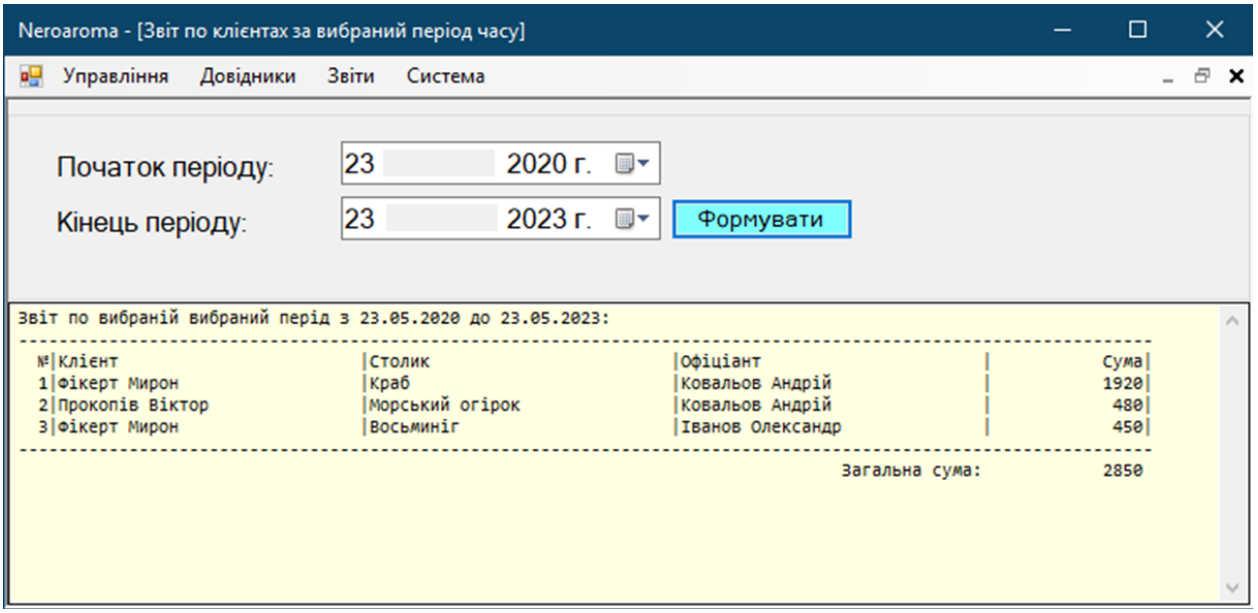

Рисунок 3.28 – Звіт по клієнтах за вибраний період

У вікні "За період часу" користувач може вказати початкову та кінцеву дати періоду, за який потрібно сформувати звіт. Після вибору дат та натискання відповідної кнопки, програма здійснює обробку даних та генерує звіт з необхідною інформацією.

За допомогою цієї функціональності ресторанний персонал може аналізувати фінансові показники, витрати на інгредієнти, заробітну плату працівників та інші аспекти діяльності ресторану. Це сприяє прийняттю обґрунтованих рішень щодо управління та розвитку бізнесу.

Після переходу до пункту меню "Звіти" та обрання опції "По офіціанту" в програмі "Neroaroma", користувачу надається можливість формувати

звітність за вибраного офіціанта, що приймав замовлення. Ця функція дозволяє отримати детальну інформацію про продуктивність та роботу конкретного офіціанта.

У вікні "По офіціанту" користувач може обрати конкретного офіціанта зі списку доступних ініціалів або прізвищ, після чого програма здійснює обробку даних та генерує звіт, який містить інформацію про замовлення, що були прийняті цим офіціантом (рис. 3.29). Звіт може включати такі дані, як кількість замовлень, сумарний обсяг продажів, середня чек, час обслуговування та інші параметри, які дозволяють оцінити продуктивність та ефективність роботи офіціанта.

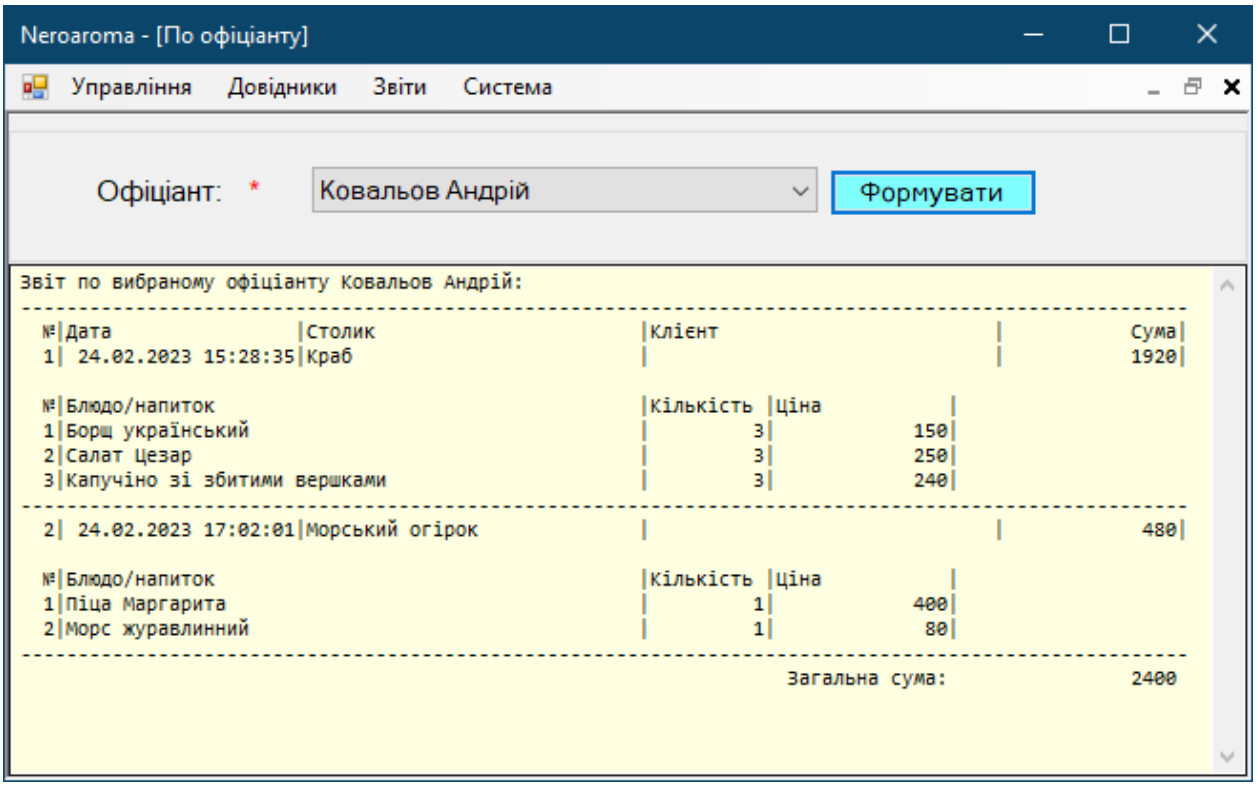

Рисунок 3.29 – Звіт по вибраному офіціанту

За допомогою цієї функціональності ресторанний персонал та керівництво можуть аналізувати роботу окремих офіціантів, визначати найбільш успішних працівників, виявляти можливі недоліки та здійснювати відповідні корективи у роботі персоналу. Це сприяє покращенню якості обслуговування та задоволенню клієнтів.

Пункт меню "Загальні витрати за вибраний період часу" в програмі "Neroaroma" надає можливість формувати звітність про загальні витрати ресторану за обраний період часу. Ця функція дозволяє керівництву та фінансовому відділу аналізувати витрати, контролювати бюджет та приймати відповідні рішення.

Після вибору пункту меню "Загальні витрати за вибраний період часу", користувачу відкривається вікно, де він може встановити початкову та кінцеву дати для періоду, за який необхідно сформувати звіт. Після введення необхідних дат і натискання кнопки "Сформувати звіт", програма здійснює обробку даних та показує інформацію про загальні витрати ресторану за вказаний період часу (рис. 3.30).

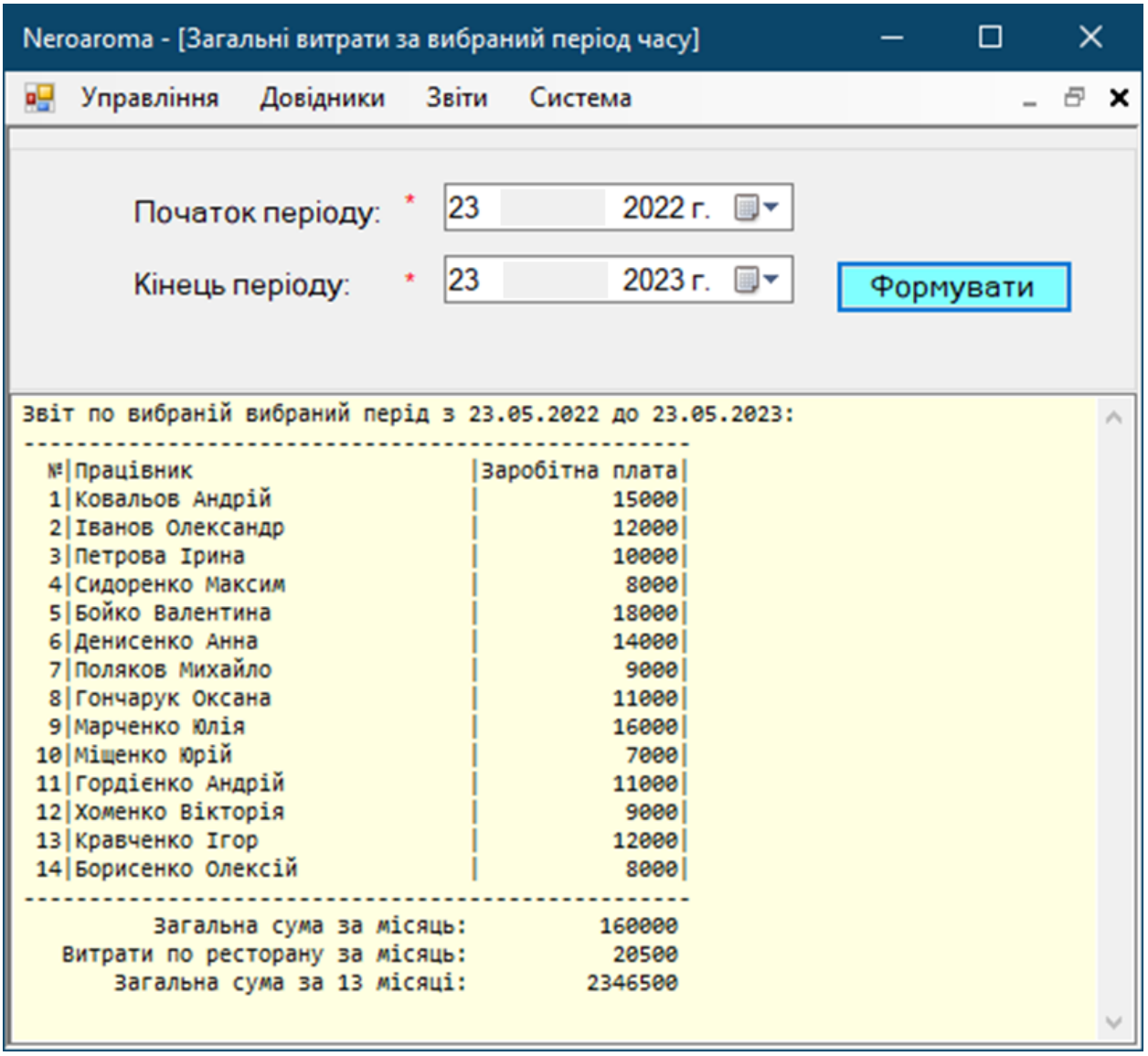

Рисунок 3.30 – Звіт по загальним витратам ресторану

Користувач з роллю "Системний адміністратор" в програмі "Neroaroma" має особливі привілеї та доступ до функціоналу, пов'язаного з управлінням обліковими записами системи. Це дозволяє адміністратору здійснювати керування користувачами, їх правами доступу та іншими параметрами, пов'язаними з безпекою та адмініструванням.

Після входу в систему з обліковим записом адміністратора та переходу до меню "Управління" - "Облікові записи", користувачу надається можливість переглядати список облікових записів системи, включаючи їх інформацію та параметри. Зображення вікна управління обліковими записами системи наведено на рис. 3.31.

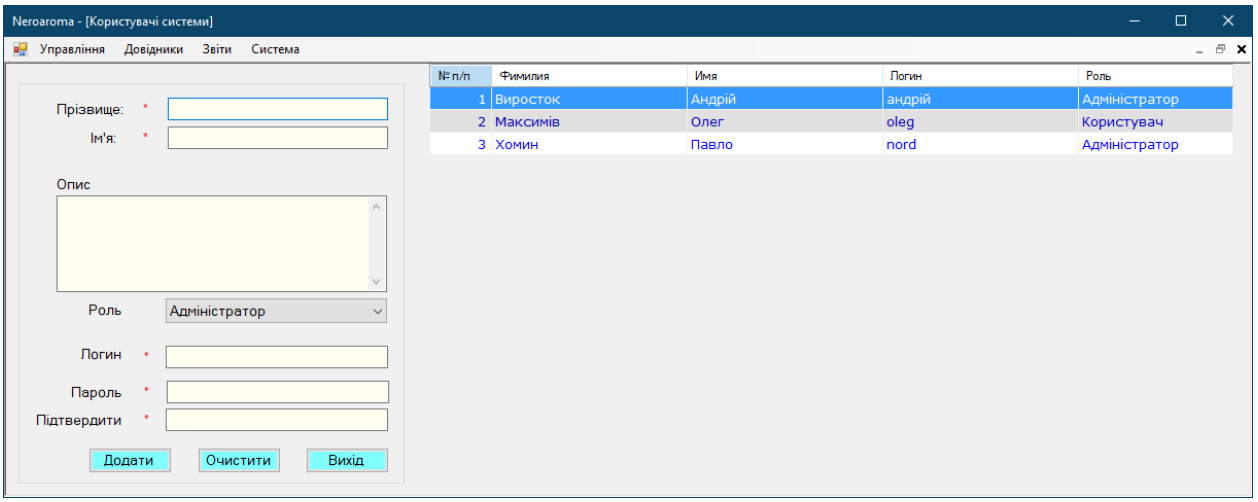

Рисунок 3.31 – Вікно для управління користувачами системи

У програмі "Neroaroma" існує можливість змінити дані та пароль будь-якого користувача в разі потреби. Для цього користувачу, який має відповідні права доступу, необхідно перейти до меню "Управління" - "Облікові записи" та вибрати потрібний запис у правій частині вікна, як показано на рис. 3.32.

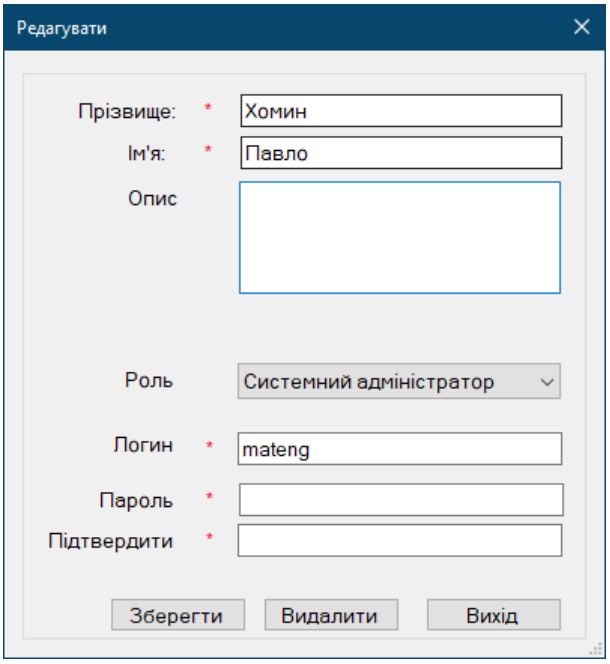

Рисунок 3.32 – Редагування облікових даних

Завершення роботи в програмі "Neroaroma" передбачає виконання дії виходу з програми. Для цього користувачу необхідно перейти до меню "Управління" та обрати пункт "Вихід".

3.6 Висновок

Розробка основних алгоритмів була проведена для забезпечення роботи додатку Neroaroma та взаємодії з користувачами. Були розроблені методи для додавання і редагування інформації про замовлення, клієнтів, співробітників, столики та інші функції, необхідні для ефективного функціонування кав'ярні.

Розробка програмних модулів передбачала створення відповідних форм та вікон для взаємодії користувачів з програмою. Були реалізовані функції додавання, редагування та видалення даних, а також можливість формування звітності за різними критеріями, такими як період часу, офіціант або загальні витрати ресторану.

Функціональне та модульне тестування були проведені для перевірки відповідності програмного забезпечення вимогам та очікуванням користувачів. Тестування включало перевірку основних функцій програми, її

продуктивності та надійності. Застосування тест-кейсів та модульного тестування дозволило виявити та виправити помилки та недоліки в роботі програми, забезпечивши якість та стабільність програмного забезпечення.

Інструкція користувача була розроблена з метою надання користувачам необхідної інформації щодо використання програми Neroaroma. Вона включала пояснення кожного пункту меню та функціоналу програми, процедури введення та редагування даних, а також кроки для формування звітів та виконання інших операцій.

ВИСНОВКИ

У ході виконання даної роботи було детально проаналізовано поточний стан ресторанного бізнесу, зокрема бізнес-процеси кав'ярні Neroaroma. Було виконане порівняння з існуючими програмними рішеннями, такими як Toast, Square for Restaurants, Lightspeed Restaurant, що допомогло сформулювати чітку постановку задачі для розробки нового програмного забезпечення.

У процесі вибору технологій проектування та розробки особливу увагу було приділено аналізу вимог до системи, побудові use-case діаграм, визначенню основних прецедентів. Це дозволило зробити обґрунтований вибір мови програмування (C#), середовища розробки (Visual Studio), та СУБД (MS Access). Враховуючи специфіку задачі, було прийнято рішення про розробку програмного забезпечення на основі трьохрівневої архітектури, що має свої особливості на кожному з рівнів (DAL, BLL, UI).

Робота над розробкою програмного забезпечення включала проєктування бази даних на основі MS Access, розробку основних алгоритмів та програмних модулів. Впровадження функціонального та модульного тестування (Unit test) дозволило перевірити коректність роботи програмних модулів та їх взаємодію. Робота над інструкцією користувача забезпечила зрозумілість та простоту використання розробленого програмного забезпечення для кінцевих користувачів.

В результаті розробки програмного забезпечення для підтримки бізнес-процесів кав'ярні Neroaroma було створено функціональний інструмент, який відповідає всім вимогам та потребам бізнесу. Це рішення дозволяє автоматизувати значну частину бізнес-процесів, забезпечує швидкість та надійність обробки даних, а також покращує якість обслуговування клієнтів.

Розробка програмного забезпечення для підтримки бізнес-процесів кав'ярні Neroaroma має великі перспективи для подальшого розвитку та вдосконалення, а саме:

– розширення функціоналу. В перспективі можна додати нові функції, що відповідають змінним потребам бізнесу, такі як інтеграція з мобільними додатками для замовлення, автоматизація процесу замовлення продуктів та інвентарю, аналітика продажів та взаємодії з клієнтами;

– інтеграція з іншими системами. Є можливість інтеграції з іншими системами, що використовуються в галузі ресторанного бізнесу, такими як системи бухгалтерії, системи управління персоналом, системи онлайн-бронювання та ін;

– масштабування. Розроблене програмне забезпечення може бути адаптовано для використання в інших кав'ярнях чи ресторанах. Воно також може бути оптимізовано для великих мереж ресторанів, що вимагає додаткового масштабування та оптимізації;

– міграція на потужнішу СУБД. У разі зростання обсягів даних та потреби в більш продуктивних рішеннях можлива міграція на потужніші СУБД, такі як MS SQL Server або Oracle.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Король А. Бізнес-процеси ресторанного бізнесу: сучасні підходи до аналізу та оптимізації. Київ : Видавництво "Київський університет", 2020. 210с.

2. Дмитренко Ю. Бізнес-планування в гастрономічній сфері: від теорії до практики. Одеса : Видавництво "Астропринт", 2018. 178 с.

3. Smith, J. Modern Cafe Business Processes: Analysis and Improvement. New York: Coffee Books Publishing, 2021. 290 c.

4. Davis, L. & Johnson, R. The Impact of Technology on Coffee Shop Operations: A Case Study. International Journal of Business and Management, 2019. 189 c.

5. Richardson, P. Sustainable practices in coffee business: An examination of the benefits and challenges. Journal of Hospitality and Tourism Management, 2020. 765 c.

6. Johnson, M. Understanding Toast POS: A Comprehensive Guide. Business Software Digest, 2020. 245 c.

7. Harris, J. The impact of Toast POS on the restaurant industry. Journal of Business Technology, 2019. 358 c.

8. Peterson, L. The effectiveness of Toast POS in the restaurant business. Business Innovations, 2021. 287 c.

9. Davis, G. Square for Restaurants: A New Approach to POS Systems. Tech Review, 2020. 250 c.

10. White, J. The Role of Square POS in Modern Restaurant Business. International Journal of Business and Management, 2019. 754 c.

11. Young, P. The Comparative Study of Square POS in Restaurant Industry. Hospitality Tech Review, 2021. 452 c.

12. Wilson, H. Lightspeed Restaurant POS: A detailed analysis. Business Tech Journal, 2020. 352 с.

13. Anderson, L. The Role of Lightspeed in the Modern Restaurant Industry. Journal of Hospitality and Technology, 2019. 451 с.

14. Thompson, R. Evaluating Lightspeed Restaurant POS for your business. Tech Review, 2021. 874 с.

15. Ларман К. Agile і ітеративна розробка: Практичне керівництво. Харків: Фактор, 2018. 752 с.

16. Cockburn A. Writing Effective Use Cases. Boston, MA: Addison-Wesley Professional, 2018. 345с.

17. Albahari J., Albahari B. C# 7.0 in a Nutshell: The Definitive Reference. Sebastopol, CA: O'Reilly Media, 2018. 576 с.

18. Страуструп Б. Програмування. Принципи та практика за допомогою C++. Київ: Видавництво "Dialectika", 2019. 760 с.

19. Хорстманн К. Java. Бібліотека професіонала. Том 1. Основи. Київ: Видавництво "Dialectika", 2020. 864 с.

20. Johnson M. Mastering Visual Studio 2019: Become proficient in .NET Framework and .NET Core by using Visual Studio 2019. Birmingham, UK: Packt Publishing, 2019. 766 с.

21. Vogel L., Heinrichs D., Wayne B. The Java Developer's Guide to Eclipse, 2nd Edition. Boston, MA: Addison-Wesley Professional, 2020. 877 с.

22. Lindqvist D. Mastering PyCharm. Birmingham, UK: Packt Publishing, 2020. 345 с.

23. Petkovic D. Microsoft SQL Server 2019: A Beginner's Guide, Seventh Edition. New York, NY: McGraw-Hill Education, 2020. 767 с.

24. Bryla B., Loney K. Oracle Database 12c DBA Handbook. New York, NY: McGraw-Hill Education, 2018. 876 с.

25. Hernandez M. J. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design, 4th Edition. Boston, MA: Addison-Wesley Professional, 2020. 456 с.

26. Фаулер М., Рісби Р. Рефакторинг. Удосконалення існуючого коду. Київ: Видавництво "Dialectika", 2019. 456 с.

27. Старчиков Є. Сучасні підходи до проектування програмних систем. Львів: ЛНУ ім. Івана Франка, 2018. 320 с.

28. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, MA: Addison-Wesley Professional, 2020. 233 с.

Додаток А. Лістинги програми

Лістинг 1. Код класу «OrdersProvider»

```
using RestaurantApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantApp.Providers {
  class OrdersProvider {
    ClientsProvider _ClientProvider = new ClientsProvider();
    List<Clients> _ClientList = new List<Clients>();
    RestaurantTablesProvider _RestaurantTablesProvider = new RestaurantTablesProvider();
    List<RestaurantTables> _RestaurantTablesList = new List<RestaurantTables>();
    WorkersProvider _WorkerProvider = new WorkersProvider();
    List<Workers> _WorkerList = new List<Workers>();


    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertOrders(DateTime StartDate, int ClientsId, int WorkersId, int
      RestaurantTablesId, string Description) {
      string SqlString = "INSERT INTO Orders (StartDate, ClientsId, WorkersId, " +
        "RestaurantTablesId, Description" +
        ") Values(?, ?, ?, ?, ?)";
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("StartDate", StartDate.ToString());
          cmd.Parameters.AddWithValue("ClientsId", ClientsId);
          cmd.Parameters.AddWithValue("WorkersId", WorkersId);
          cmd.Parameters.AddWithValue("RestaurantTablesId", RestaurantTablesId);
          cmd.Parameters.AddWithValue("Description", Description);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Orders> GetAllOrders() {
      _ClientList = _ClientProvider.GetAllClients();
      _WorkerList = _WorkerProvider.GetAllWorkers();
      _RestaurantTablesList = _RestaurantTablesProvider.GetAllRestaurantTables();

      int i = 0;
      string SqlString = "SELECT * FROM Orders WHERE Status=false ORDER BY StartDate";
```

```
    List<Orders> listOrders = new List<Orders>();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            Orders oneOrders = new Orders();
            oneOrders.Number = ++i;
            oneOrders.OrdersId = Convert.ToInt32(reader["OrdersId"]);
            oneOrders.StartDate = Convert.ToDateTime(reader["StartDate"]);
            oneOrders.ClientsId = Convert.ToInt32(reader["ClientsId"]);
            oneOrders.WorkersId = Convert.ToInt32(reader["WorkersId"]);
            oneOrders.RestaurantTablesId = Convert.ToInt32(reader["RestaurantTablesId"]);
            oneOrders.Description = reader["Description"].ToString();
            listOrders.Add(oneOrders);
          }
        }
        conn.Close();
      }
    }

    if (listOrders.Count == 0) {
      Orders noOrders = new Orders();
      noOrders.OrdersId = 0;
      noOrders.Message = NamesMy.NoDataNames.NoDataInOrders;
      listOrders.Add(noOrders);
    } else {
      for (int j = 0; j < listOrders.Count; j++) {
        listOrders[j].ClientsFIO = GetClientFIO(listOrders[j].ClientsId, _ClientList);
        listOrders[j].WorkersFIO = GetWorkerFIO(listOrders[j].WorkersId, _WorkerList);
        listOrders[j].RestaurantTablesName =
GetRestaurantTablesName(listOrders[j].RestaurantTablesId, _RestaurantTablesList);
      }
    }
    return listOrders;
  }

  public List<Orders> GetAllFinishOrders() {
    _ClientList = _ClientProvider.GetAllClients();
    _WorkerList = _WorkerProvider.GetAllWorkers();
    _RestaurantTablesList = _RestaurantTablesProvider.GetAllRestaurantTables();

    int i = 0;
    string SqlString = "SELECT * FROM Orders WHERE Status=true ORDER BY StartDate";

    List<Orders> listOrders = new List<Orders>();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
```

```
            Orders oneOrders = new Orders();
            oneOrders.Number = ++i;
            oneOrders.OrdersId = Convert.ToInt32(reader["OrdersId"]);
            oneOrders.StartDate = Convert.ToDateTime(reader["StartDate"]);
            oneOrders.ClientsId = Convert.ToInt32(reader["ClientsId"]);
            oneOrders.WorkersId = Convert.ToInt32(reader["WorkersId"]);
            oneOrders.RestaurantTablesId = Convert.ToInt32(reader["RestaurantTablesId"]);
            oneOrders.Description = reader["Description"].ToString();
            listOrders.Add(oneOrders);
          }
        }
        conn.Close();
      }
    }

    if (listOrders.Count == 0) {
      Orders noOrders = new Orders();
      noOrders.OrdersId = 0;
      noOrders.Message = NamesMy.NoDataNames.NoDataInOrders;
      listOrders.Add(noOrders);
    } else {
      for (int j = 0; j < listOrders.Count; j++) {
        listOrders[j].ClientsFIO = GetClientFIO(listOrders[j].ClientsId, _ClientList);
        listOrders[j].WorkersFIO = GetWorkerFIO(listOrders[j].WorkersId, _WorkerList);
        listOrders[j].RestaurantTablesName =
GetRestaurantTablesName(listOrders[j].RestaurantTablesId, _RestaurantTablesList);
      }
    }
    return listOrders;
  }

  public Orders SelectedOrdersByOrdersId(int OrdersId) {
    string SqlString = "SELECT *  FROM Orders Where OrdersId=" + OrdersId.ToString();

    Orders oneOrders = new Orders();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneOrders.OrdersId = Convert.ToInt32(reader["OrdersId"]);
            oneOrders.StartDate = Convert.ToDateTime(reader["StartDate"]);
            oneOrders.ClientsId = Convert.ToInt32(reader["ClientsId"]);
            oneOrders.WorkersId = Convert.ToInt32(reader["WorkersId"]);
            oneOrders.RestaurantTablesId = Convert.ToInt32(reader["RestaurantTablesId"]);
            oneOrders.Description = reader["Description"].ToString();
          }
        }
      }
      conn.Close();
    }
    return oneOrders;
```

```
    }

    private string GetRestaurantTablesName(int RestaurantTablesId, List<RestaurantTables>
RestaurantTablesList) {
      for (int i = 0; i < RestaurantTablesList.Count; i++) {
        if (RestaurantTablesId == RestaurantTablesList[i].RestaurantTablesId) {
          return RestaurantTablesList[i].RestaurantTablesName.ToString();
        }
      }
      return "";
    }

    private string GetClientFIO(int ClientsId, List<Clients> ClientsList) {
      for (int i = 0; i < ClientsList.Count; i++) {
        if (ClientsId == ClientsList[i].ClientsId) {
          return ClientsList[i].FIO;
        }
      }
      return "";
    }

    private string GetWorkerFIO(int WorkersId, List<Workers> WorkersList) {
      for (int i = 0; i < WorkersList.Count; i++) {
        if (WorkersId == WorkersList[i].WorkersId) {
          return WorkersList[i].FIO;
        }
      }
      return "";
    }

    public void UpdateOrders(DateTime StartDate, int ClientsId, int WorkersId, int
RestaurantTablesId, string Description, int OrdersId) {
      string SqlString = "UPDATE Orders SET StartDate=?, ClientsId=?, WorkersId=?,
RestaurantTablesId=?, Description=? " +
  "WHERE OrdersId=?";
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("StartDate", StartDate.ToString());
          cmd.Parameters.AddWithValue("ClientsId", ClientsId);
          cmd.Parameters.AddWithValue("WorkersId", WorkersId);
          cmd.Parameters.AddWithValue("RestaurantTablesId", RestaurantTablesId);
          cmd.Parameters.AddWithValue("Description", Description);
          cmd.Parameters.AddWithValue("OrdersId", OrdersId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public void DeleteOrdersByOrdersId(int OrdersId) {
```

```
    string SqlString = "DELETE FROM Orders WHERE OrdersId=" + OrdersId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
     using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
     }
    }
   }

   public int GetLastRecords() {
    int lastRecordNumber = 0;
    string SqlString = "Select LAST (OrdersId) From Orders ";
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
     using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
       while (reader.Read()) {
        lastRecordNumber = Convert.ToInt32(reader.GetValue(0));
       }
      }
     }
     conn.Close();
    }
    return lastRecordNumber;
   }

   public void UpdateStatus(int OrdersId, bool Status) {
    string SqlString = "UPDATE Orders SET Status=? WHERE OrdersId=?";
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
     using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("Status", Status);
      cmd.Parameters.AddWithValue("OrdersId", OrdersId);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
     }
    }
   }

  }
}

public class Orders {
  private int _Number;
  private int _OrdersId;
  private DateTime _StartDate;
  private int _ClientsId;
  private string _ClientsFIO;
  private int _WorkersId;
  private string _WorkersFIO;
```

```
private int _RestaurantTablesId;
private string _RestaurantTablesName;
private string _Description;
private double _Price;
private bool _Status;
private string _Message;

public Orders() {
  _Number = 0;
  _OrdersId = 0;
  _Description = String.Empty;
  _ClientsId = 0;
  _WorkersId = 0;
  _WorkersFIO = String.Empty;
  _StartDate = new DateTime();
  _RestaurantTablesId = 0;
  _RestaurantTablesName = String.Empty;
  _ClientsFIO = String.Empty;
  _Price = 0.0;
  _Status = false;
  _Message = String.Empty;
}

public int Number {
  set { _Number = value; }
  get { return _Number; }
}
public int OrdersId {
  set { _OrdersId = value; }
  get { return _OrdersId; }
}
public DateTime StartDate {
  set { _StartDate = value; }
  get { return _StartDate; }
}
public int ClientsId {
  set { _ClientsId = value; }
  get { return _ClientsId; }
}
public string ClientsFIO {
  set { _ClientsFIO = value; }
  get { return _ClientsFIO; }
}
public int WorkersId {
  set { _WorkersId = value; }
  get { return _WorkersId; }
}
public string WorkersFIO {
  set { _WorkersFIO = value; }
  get { return _WorkersFIO; }
}
```

```
  public int RestaurantTablesId {
    set { _RestaurantTablesId = value; }
    get { return _RestaurantTablesId; }
  }
  public string RestaurantTablesName {
    set { _RestaurantTablesName = value; }
    get { return _RestaurantTablesName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public bool Status {
    set { _Status = value; }
    get { return _Status; }
  }
  public double Price {
    set { _Price = value; }
    get { return _Price; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 2. Код класу «RestaurantExpensesProvider»

```
using RestaurantApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantApp.Providers {
  class RestaurantExpensesProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertRestaurantExpenses(double Electricitys, double Waters, double Gas, double
InternetTelephone, double Garbage) {
      string SqlString = "INSERT INTO RestaurantExpenses (Electricitys, Waters, Gas,
InternetTelephone, Garbage" +
        ") Values(?, ?, ?, ?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("Electricitys", Electricitys);
          cmd.Parameters.AddWithValue("Waters", Waters);
          cmd.Parameters.AddWithValue("Gas", Gas);
```

```
        cmd.Parameters.AddWithValue("InternetTelephone", InternetTelephone);
        cmd.Parameters.AddWithValue("Garbage", Garbage);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

    public RestaurantExpenses SelectedRestaurantExpensesByRestaurantExpensesId(int
RestaurantExpensesId) {
     string SqlString = "SELECT * FROM RestaurantExpenses Where RestaurantExpensesId=" +
RestaurantExpensesId.ToString();

     RestaurantExpenses oneRestaurantExpenses = new RestaurantExpenses();
     using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneRestaurantExpenses.RestaurantExpensesId =
Convert.ToInt32(reader["RestaurantExpensesId"]);
            oneRestaurantExpenses.RestaurantExpensesId =
Convert.ToInt32(reader["RestaurantExpensesId"]);
            oneRestaurantExpenses.Electricitys = Convert.ToDouble(reader["Electricitys"]);
            oneRestaurantExpenses.Waters = Convert.ToDouble(reader["Waters"]);
            oneRestaurantExpenses.Gas = Convert.ToDouble(reader["Gas"]);
            oneRestaurantExpenses.InternetTelephone =
Convert.ToDouble(reader["InternetTelephone"]);
            oneRestaurantExpenses.Garbage = Convert.ToDouble(reader["Garbage"]);
          }
        }
      }
      conn.Close();
    }
    return oneRestaurantExpenses;
  }

   public void UpdateRestaurantExpenses(double Electricitys, double Waters, double Gas,
double InternetTelephone, double Garbage, int RestaurantExpensesId) {
     string SqlString = "UPDATE RestaurantExpenses SET Electricitys=?, Waters=?, Gas=?,
InternetTelephone=?, Garbage=?  " +
  "WHERE RestaurantExpensesId=?";

     using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("Electricitys", Electricitys);
        cmd.Parameters.AddWithValue("Waters", Waters);
        cmd.Parameters.AddWithValue("Gas", Gas);
        cmd.Parameters.AddWithValue("InternetTelephone", InternetTelephone);
        cmd.Parameters.AddWithValue("Garbage", Garbage);
```

```
        cmd.Parameters.AddWithValue("RestaurantExpensesId", RestaurantExpensesId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public void DeleteRestaurantExpensesByRestaurantExpensesId(int RestaurantExpensesId) {
    string SqlString = "DELETE FROM RestaurantExpenses WHERE RestaurantExpensesId="
+ RestaurantExpensesId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

 }
}


public class RestaurantExpenses {
 private int _Number;
 private int _RestaurantExpensesId;
 private double _Electricitys;
 private double _Waters;
 private double _Gas;
 private double _InternetTelephone;
 private double _Garbage;
 private string _Message;

 public RestaurantExpenses() {
  _Number = 0;
  _RestaurantExpensesId = 0;
  _Electricitys = 0.0;
  _Waters = 0.0;
  _Gas = 0.0;
  _InternetTelephone = 0.0;
  _Garbage = 0.0;
  _Message = String.Empty;
 }

 public int Number {
  set { _Number = value; }
  get { return _Number; }
 }
 public int RestaurantExpensesId {
  set { _RestaurantExpensesId = value; }
  get { return _RestaurantExpensesId; }
```

```
  }
  public double Electricitys {
    set { _Electricitys = value; }
    get { return _Electricitys; }
  }
  public double Waters {
    set { _Waters = value; }
    get { return _Waters; }
  }
  public double Gas {
    set { _Gas = value; }
    get { return _Gas; }
  }
  public double InternetTelephone {
    set { _InternetTelephone = value; }
    get { return _InternetTelephone; }
  }
  public double Garbage {
    set { _Garbage = value; }
    get { return _Garbage; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 3. Код класу «WorkersProvider»

```
using RestaurantApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantApp.Providers {
  class WorkersProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    public void InsertWorkers(string LastName, string FirstName, string Phone, string Address,
string Email, double Salaries, int PostsId) {
      string SqlString = "INSERT INTO Workers (LastName, FirstName, Phone, Address, " +
        "Email, Salaries, PostsId) Values(?, ?, ?, ?, ?, ?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("LastName", LastName);
          cmd.Parameters.AddWithValue("FirstName", FirstName);
```

```
      cmd.Parameters.AddWithValue("Phone", Phone);
      cmd.Parameters.AddWithValue("Address", Address);
      cmd.Parameters.AddWithValue("Email", Email);
      cmd.Parameters.AddWithValue("Salaries", Salaries);
      cmd.Parameters.AddWithValue("PostsId", PostsId);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}

public List<Workers> GetAllWorkers() {
  int i = 0;
  string SqlString = "SELECT * FROM Workers";

  List<Workers> listAllWorkers = new List<Workers>();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Workers oneWorkers = new Workers();
          oneWorkers.Number = ++i;
          oneWorkers.WorkersId = Convert.ToInt32(reader["WorkersId"]);
          oneWorkers.FirstName = reader["FirstName"].ToString();
          oneWorkers.LastName = reader["LastName"].ToString();
          oneWorkers.FIO = oneWorkers.LastName + " " + oneWorkers.FirstName;
          oneWorkers.Phone = reader["Phone"].ToString();
          oneWorkers.Address = reader["Address"].ToString();
          oneWorkers.Email = reader["Email"].ToString();
          oneWorkers.Salaries = Convert.ToDouble(reader["Salaries"]);
          oneWorkers.PostsId = Convert.ToInt32(reader["PostsId"]);
          listAllWorkers.Add(oneWorkers);
        }
      }
      conn.Close();
    }
  }

  if (listAllWorkers.Count == 0) {
    Workers noWorkers = new Workers();
    noWorkers.WorkersId = 0;
    noWorkers.Message = NamesMy.NoDataNames.NoDataInWorkers;
    listAllWorkers.Add(noWorkers);
  }
  return listAllWorkers;
}

public Workers SelectedWorkersByWorkersId(int WorkersId) {
  string SqlString = "SELECT * FROM Workers Where WorkersId=" + WorkersId.ToString();
```

```
    Workers oneWorkers = new Workers();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneWorkers.WorkersId = Convert.ToInt32(reader["WorkersId"]);
            oneWorkers.FirstName = reader["FirstName"].ToString();
            oneWorkers.LastName = reader["LastName"].ToString();
            oneWorkers.FIO = oneWorkers.LastName + " " + oneWorkers.FirstName;
            oneWorkers.Phone = reader["Phone"].ToString();
            oneWorkers.Address = reader["Address"].ToString();
            oneWorkers.Email = reader["Email"].ToString();
            oneWorkers.Salaries = Convert.ToDouble(reader["Salaries"]);
            oneWorkers.PostsId = Convert.ToInt32(reader["PostsId"]);
          }
        }
      }
      conn.Close();
    }
    return oneWorkers;
  }

  public void UpdateWorkers(string LastName, string FirstName, string Phone, string Address,
string Email, double Salaries, int PostsId, int WorkersId) {
    string SqlString = "UPDATE Workers SET FirstName=?, LastName=?, Phone=?, " +
 "Address=?, Email=?, Salaries=?, PostsId=? WHERE WorkersId=?";

    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("FirstName", FirstName);
        cmd.Parameters.AddWithValue("LastName", LastName);
        cmd.Parameters.AddWithValue("Phone", Phone);
        cmd.Parameters.AddWithValue("Address", Address);
        cmd.Parameters.AddWithValue("Email", Email);
        cmd.Parameters.AddWithValue("Salaries", Salaries);
        cmd.Parameters.AddWithValue("PostsId", PostsId);
        cmd.Parameters.AddWithValue("WorkersId", WorkersId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public void DeleteWorkersByWorkersId(int WorkersId) {
    string SqlString = "DELETE FROM Workers WHERE WorkersId=" + WorkersId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
```

```
            conn.Close();
          }
        }
      }

    }
  }


public class Workers {
  private int _Number;
  private int _WorkersId;
  private string _FirstName;
  private string _LastName;
  private string _Phone;
  private string _Address;
  private string _Email;
  private double _Salaries;
  private int _PostsId;
  private string _FIO;
  private string _Message;

  public Workers() {
    _Number = 0;
    _WorkersId = 0;
    _FirstName = String.Empty;
    _LastName = String.Empty;
    _Phone = String.Empty;
    _Address = String.Empty;
    _Email = String.Empty;
    _Salaries = 0.0;
    _PostsId = 0;
    _FIO = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int WorkersId {
    set { _WorkersId = value; }
    get { return _WorkersId; }
  }
  public string FirstName {
    set { _FirstName = value; }
    get { return _FirstName; }
  }
  public string LastName {
    set { _LastName = value; }
    get { return _LastName; }
  }
```

```
  public string Phone {
    set { _Phone = value; }
    get { return _Phone; }
  }
  public string Address {
    set { _Address = value; }
    get { return _Address; }
  }
  public string Email {
    set { _Email = value; }
    get { return _Email; }
  }
  public double Salaries {
    set { _Salaries = value; }
    get { return _Salaries; }
  }
  public int PostsId {
    set { _PostsId = value; }
    get { return _PostsId; }
  }
  public string FIO {
    set { _FIO = value; }
    get { return _FIO; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 4. Код класу «RestaurantExpensesForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms.Dictionary {
  public partial class RestaurantExpensesForm : Form {
    private RestaurantExpenses _selectedRestaurantExpenses = new RestaurantExpenses();
    private RestaurantExpensesProvider _RestaurantExpensesProvider = new
RestaurantExpensesProvider();
    private ValidationMy _Validation = new ValidationMy();
    public RestaurantExpensesForm() {
      InitializeComponent();
```

```
    LoadAllDate();
  }

  private void SaveBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {

_RestaurantExpensesProvider.UpdateRestaurantExpenses(Convert.ToDouble(ElectricitysTBox.T
ext),
        Convert.ToDouble(WatersTBox.Text), Convert.ToDouble(GasTBox.Text),
Convert.ToDouble(InternetTelephoneTBox.Text), Convert.ToDouble(GarbageTBox.Text), 1);
      MessageBox.Show("Дані успішно збережено!");
      this.Close();
    }
  }

  private void ExitBtn_Click(object sender, EventArgs e) {
    this.Close();
  }

  private void LoadAllDate() {
    _selectedRestaurantExpenses =
_RestaurantExpensesProvider.SelectedRestaurantExpensesByRestaurantExpensesId(1);
    ElectricitysTBox.Text = _selectedRestaurantExpenses.Electricitys.ToString();
    WatersTBox.Text = _selectedRestaurantExpenses.Waters.ToString();
    GasTBox.Text = _selectedRestaurantExpenses.Gas.ToString();
    InternetTelephoneTBox.Text = _selectedRestaurantExpenses.InternetTelephone.ToString();
    GarbageTBox.Text = _selectedRestaurantExpenses.Garbage.ToString();
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_Validation.IsDataConvertToDouble(ElectricitysTBox.Text)) {
      ElectricitysValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      ElectricitysValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(WatersTBox.Text)) {
      WatersValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      WatersValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(GasTBox.Text)) {
      GasValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      GasValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(InternetTelephoneTBox.Text)) {
      InternetTelephoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
```

```
      InternetTelephoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_Validation.IsDataConvertToDouble(GarbageTBox.Text)) {
      GarbageValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      GarbageValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     return isCorrect;
    }

  }
}
```

Лістинг 5. Код класу «RestaurantMenusForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms.Dictionary {
  public partial class RestaurantMenusForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private RestaurantMenusProvider _RestaurantMenusPrivider = new
RestaurantMenusProvider();
    private List<RestaurantMenus> _RestaurantMenusList = new List<RestaurantMenus>();
    private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
    private List<Categories> _CategoriesList = new List<Categories>();

    public RestaurantMenusForm() {
      InitializeComponent();
      LoadAllDate();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _RestaurantMenusPrivider.InsertRestaurantMenus(Convert.ToDouble(PriceTBox.Text),
MenusNameTBox.Text, IngredientsTBox.Text,
          Convert.ToInt32(NumberOfCaloriesTBox.Text), DescriptionTBox.Text,
Convert.ToInt32(WeightsTBox.Text), Convert.ToInt32(CategoriesCBox.SelectedValue));
        DataLoad();
        ClearAllControls();
```

```
    }
  }

  private void ClearBtn_Click(object sender, EventArgs e) {
    ClearAllControls();
  }

  private void ExitBtn_Click(object sender, EventArgs e) {
    this.Close();
  }

  private void LoadAllDate() {
    _CategoriesList = _CategoriesProvider.GetAllCategories();
    CategoriesCBox.DataSource = _CategoriesList;
    CategoriesCBox.ValueMember = "CategoriesId";
    CategoriesCBox.DisplayMember = "CategoriesName";
  }

  private void DataLoad() {
    int firstRowIndex = 0;
    if (RestaurantMenusGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = RestaurantMenusGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _RestaurantMenusList = _RestaurantMenusPrivider.GetAllRestaurantMenus();
      LoadDataInRestaurantMenusGridView(_RestaurantMenusList);
      if (_selectedRowIndex == RestaurantMenusGridView.Rows.Count) {
        _selectedRowIndex = RestaurantMenusGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        RestaurantMenusGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        RestaurantMenusGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInRestaurantMenusGridView(List<RestaurantMenus>
RestaurantMenusList) {
    RestaurantMenusGridView.DataSource = null;
    RestaurantMenusGridView.Columns.Clear();
    RestaurantMenusGridView.AutoGenerateColumns = false;
    RestaurantMenusGridView.RowHeadersVisible = false;

    RestaurantMenusGridView.DataSource = RestaurantMenusList;

    if (RestaurantMenusList.Count > 0) {
      if (RestaurantMenusList[0].Message ==
NamesMy.NoDataNames.NoDataInRestaurantMenus) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = RestaurantMenusGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
```

```
            RestaurantMenusGridView.Columns.Add(messageColumn);
        } else {
            DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
            DetailIdColumn.DataPropertyName = "RestaurantMenusId";
            RestaurantMenusGridView.Columns.Add(DetailIdColumn);
            RestaurantMenusGridView.Columns[0].Visible = false;

            DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
            numberColumn.HeaderText = "№ ";
            numberColumn.DataPropertyName = "Number";
            numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
            numberColumn.Width = NamesMy.SizeOptins.NumberSize;
            RestaurantMenusGridView.Columns.Add(numberColumn);

            DataGridViewColumn MenusNameColumn = new DataGridViewTextBoxColumn();
            MenusNameColumn.HeaderText = "Назва  блюда/напитку";
            MenusNameColumn.DataPropertyName = "MenusName";
            MenusNameColumn.Width = NamesMy.SizeOptins.NameSize;
            RestaurantMenusGridView.Columns.Add(MenusNameColumn);

            DataGridViewColumn PriceColumn = new DataGridViewTextBoxColumn();
            PriceColumn.HeaderText = "Ціна";
            PriceColumn.DataPropertyName = "Price";
            PriceColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
            PriceColumn.Width = 100;
            RestaurantMenusGridView.Columns.Add(PriceColumn);


            DataGridViewColumn WeightsColumn = new DataGridViewTextBoxColumn();
            WeightsColumn.HeaderText = "Вага гр.";
            WeightsColumn.DataPropertyName = "Weights";
            WeightsColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
            WeightsColumn.Width = 50;
            RestaurantMenusGridView.Columns.Add(WeightsColumn);

            DataGridViewColumn NumberOfCaloriesColumn = new
DataGridViewTextBoxColumn();
            NumberOfCaloriesColumn.HeaderText = "Калорій";
            NumberOfCaloriesColumn.DataPropertyName = "NumberOfCalories";
            NumberOfCaloriesColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
            NumberOfCaloriesColumn.Width = 50;
            RestaurantMenusGridView.Columns.Add(NumberOfCaloriesColumn);
        }
        for (int i = 0; i < RestaurantMenusGridView.Columns.Count; i++) {
            RestaurantMenusGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
        }
    }
}
```

```
private void ClearAllControls() {
  MenusNameTBox.Text = String.Empty;
  PriceTBox.Text = "0";
  NumberOfCaloriesTBox.Text = "0";
  IngredientsTBox.Text = String.Empty;
  DescriptionTBox.Text = String.Empty;
}

private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataEntering(MenusNameTBox.Text)) {
    MenusNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    MenusNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataConvertToDouble(PriceTBox.Text)) {
    PriceValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PriceValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataConvertToInt(NumberOfCaloriesTBox.Text)) {
    NumberOfCaloriesValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    NumberOfCaloriesValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataConvertToInt(WeightsTBox.Text)) {
    WeightsValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    WeightsValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(IngredientsTBox.Text)) {
    IngredientsValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    IngredientsValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (Convert.ToInt32(CategoriesCBox.SelectedValue) > 0) {
    CategoriesValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    CategoriesValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  return isCorrect;
}
```

```
    private void RestaurantMenusGridView_CellClick(object sender,
DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && RestaurantMenusGridView[0, e.RowIndex].Value.ToString() !=
_RestaurantMenusList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdateRestaurantMenusForm updateRestaurantMenusForm = new
UpdateRestaurantMenusForm(Convert.ToInt32(RestaurantMenusGridView[0,
e.RowIndex].Value.ToString()));
        updateRestaurantMenusForm.ShowDialog();
        DataLoad();
      }
    }

  }
}
```

Лістинг 6. Код класу «RestaurantTablesForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms.Dictionary {
  public partial class RestaurantTablesForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    RestaurantTablesProvider _RestaurantTablesProvider = new RestaurantTablesProvider();
    List<RestaurantTables> _RestaurantTablesList = new List<RestaurantTables>();

    public RestaurantTablesForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {

_RestaurantTablesProvider.InsertRestaurantTables(Convert.ToInt32(NumberSeatsTBox.Text),
RestaurantTablesNameTBox.Text, TablesLocationTBox.Text, DescriptionTBox.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
```

```
    ClearAllControls();
  }

  private void ExitBtn_Click(object sender, EventArgs e) {
    this.Close();
  }

  private void DataLoad() {
    int firstRowIndex = 0;
    if (RestaurantTablesGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = RestaurantTablesGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _RestaurantTablesList = _RestaurantTablesProvider.GetAllRestaurantTables();
      LoadDataInRestaurantTablesGridView(_RestaurantTablesList);
      if (_selectedRowIndex == RestaurantTablesGridView.Rows.Count) {
        _selectedRowIndex = RestaurantTablesGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        RestaurantTablesGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        RestaurantTablesGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInRestaurantTablesGridView(List<RestaurantTables>
RestaurantTablesList) {
    RestaurantTablesGridView.DataSource = null;
    RestaurantTablesGridView.Columns.Clear();
    RestaurantTablesGridView.AutoGenerateColumns = false;
    RestaurantTablesGridView.RowHeadersVisible = false;

    RestaurantTablesGridView.DataSource = RestaurantTablesList;

    if (RestaurantTablesList.Count > 0) {
      if (RestaurantTablesList[0].Message ==
NamesMy.NoDataNames.NoDataInRestaurantTables) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = RestaurantTablesGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        RestaurantTablesGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "RestaurantTablesId";
        RestaurantTablesGridView.Columns.Add(DetailIdColumn);
        RestaurantTablesGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
```

```
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        RestaurantTablesGridView.Columns.Add(numberColumn);

        DataGridViewColumn RestaurantTablesNameColumn = new
DataGridViewTextBoxColumn();
        RestaurantTablesNameColumn.HeaderText = "Назва столика";
        RestaurantTablesNameColumn.DataPropertyName = "RestaurantTablesName";
        RestaurantTablesNameColumn.Width = 200;
        RestaurantTablesGridView.Columns.Add(RestaurantTablesNameColumn);

        DataGridViewColumn NumberSeatsColumn = new DataGridViewTextBoxColumn();
        NumberSeatsColumn.HeaderText = "К-сть місць";
        NumberSeatsColumn.DataPropertyName = "NumberSeats";
        NumberSeatsColumn.Width = 80;
        NumberSeatsColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        RestaurantTablesGridView.Columns.Add(NumberSeatsColumn);

        DataGridViewColumn TablesLocationColumn = new DataGridViewTextBoxColumn();
        TablesLocationColumn.HeaderText = "Розташування";
        TablesLocationColumn.DataPropertyName = "TablesLocation";
        TablesLocationColumn.Width = 150;
        RestaurantTablesGridView.Columns.Add(TablesLocationColumn);

        DataGridViewColumn DescriptionColumn = new DataGridViewTextBoxColumn();
        DescriptionColumn.HeaderText = "Опис";
        DescriptionColumn.DataPropertyName = "Description";
        DescriptionColumn.Width = 230;
        RestaurantTablesGridView.Columns.Add(DescriptionColumn);

      }
      for (int i = 0; i < RestaurantTablesGridView.Columns.Count; i++) {
        RestaurantTablesGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void ClearAllControls() {
    RestaurantTablesNameTBox.Text = String.Empty;
    NumberSeatsTBox.Text = "2";
    TablesLocationTBox.Text = String.Empty;
    DescriptionTBox.Text = String.Empty;
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(RestaurantTablesNameTBox.Text)) {
      RestaurantTablesNameValiadtionLbl.Text =
NamesMy.ProgramButtons.RequiredValidation;
    } else {
```

```
      RestaurantTablesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_validation.IsDataConvertToInt(NumberSeatsTBox.Text)) {
      NumberSeatsValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      NumberSeatsValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_validation.IsDataEntering(TablesLocationTBox.Text)) {
      TablesLocationValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      TablesLocationValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     return isCorrect;
    }

    private void RestaurantTablesGridView_CellClick(object sender,
DataGridViewCellEventArgs e) {
     if (e.RowIndex >= 0 && RestaurantTablesGridView[0, e.RowIndex].Value.ToString() !=
_RestaurantTablesList[0].Message) {
      _selectedRowIndex = e.RowIndex;
      UpdateRestaurantTablesForm updateRestaurantTablesForm = new
UpdateRestaurantTablesForm(Convert.ToInt32(RestaurantTablesGridView[0,
e.RowIndex].Value.ToString()));
      updateRestaurantTablesForm.ShowDialog();
      DataLoad();
     }
    }
   }
}
```

Лістинг 7. Код класу «UpdateCategoriesForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms {
 public partial class UpdateCategoriesForm : Form {
  private int _CategoriesId = 0;
  private Categories _selectedCategories = new Categories();
  private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
  private ValidationMy _validation = new ValidationMy();
```

```
    public UpdateCategoriesForm(int CategoriesId) {
      InitializeComponent();
      _CategoriesId = CategoriesId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _CategoriesProvider.UpdateCategories(CategoriesNameTBox.Text, DescriptionTBox.Text,
_CategoriesId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _CategoriesProvider.DeleteCategoriesByCategoriesId(_CategoriesId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _selectedCategories =
_CategoriesProvider.SelectedCategoriesByCategoriesId(_CategoriesId);
      CategoriesNameTBox.Text = _selectedCategories.CategoriesName;
      DescriptionTBox.Text = _selectedCategories.Description;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(CategoriesNameTBox.Text)) {
        CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

Лістинг 8. Код класу «UpdateClientsForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms.Dictionary {
  public partial class UpdateClientsForm : Form {
    private int _ClientsId = 0;
    private Clients _selectedClients = new Clients();
    private ClientsProvider _ClientsProvider = new ClientsProvider();
    private ValidationMy _validation = new ValidationMy();

    public UpdateClientsForm(int ClientsId) {
      InitializeComponent();
      _ClientsId = ClientsId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _ClientsProvider.UpdateClients(LastNameTBox.Text, FirstNameTBox.Text,
PhoneTBox.Text, AddressTBox.Text, EmailTBox.Text, _ClientsId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _ClientsProvider.DeleteClientsByClientsId(_ClientsId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _selectedClients = _ClientsProvider.SelectedClientsByClientsId(_ClientsId);
      LastNameTBox.Text = _selectedClients.LastName;
      FirstNameTBox.Text = _selectedClients.FirstName;
      PhoneTBox.Text = _selectedClients.Phone;
      AddressTBox.Text = _selectedClients.Address;
      EmailTBox.Text = _selectedClients.Email;
    }

    private bool IsDataEnteringCorrect() {
```

```
      bool isCorrect = true;
      if (_validation.IsDataEntering(LastNameTBox.Text)) {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(FirstNameTBox.Text)) {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(PhoneTBox.Text)) {
        PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

Лістинг 9. Код класу «OrdersForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms.Controls {
  public partial class OrdersForm : Form {
    private ValidationMy _Validation = new ValidationMy();
    private RestaurantTablesProvider _RestaurantTablesProvider = new
RestaurantTablesProvider();
    private List<RestaurantTables> _RestaurantTablesList = new List<RestaurantTables>();
    private ClientsProvider _ClientsProvider = new ClientsProvider();
    private List<Clients> _ClientsList = new List<Clients>();
    private OrdersProvider _OrdersProvider = new OrdersProvider();
    private WorkersProvider _WorkersProvider = new WorkersProvider();
    private List<Workers> _WorkersList = new List<Workers>();
    private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
    private List<Categories> _CategoriesList = new List<Categories>();
```

```
    private OrdersMenuProvider _OrdersMenuProvider = new OrdersMenuProvider();
    private List<OrdersMenu> _OrdersMenuList = new List<OrdersMenu>();
    private RestaurantMenusProvider _RestaurantMenusProvider = new
RestaurantMenusProvider();
    private List<RestaurantMenus> _RestaurantMenusList = new List<RestaurantMenus>();
    private bool _IsCategoriesLoad = false;
    public OrdersForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void AddMenuBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect2()) {
        OrdersMenu oneOrdersMenu = new OrdersMenu();
        RestaurantMenus selectedRestaurantMenus = new RestaurantMenus();
        selectedRestaurantMenus =
_RestaurantMenusProvider.SelectedRestaurantMenusByRestaurantMenusId(Convert.ToInt32(Re
staurantMenusCBox.SelectedValue));
        oneOrdersMenu.MenusName = RestaurantMenusCBox.Text;
        oneOrdersMenu.RestaurantMenusId =
Convert.ToInt32(RestaurantMenusCBox.SelectedValue);
        oneOrdersMenu.Amount = Convert.ToInt32(AmountTBox.Text);
        oneOrdersMenu.Price = selectedRestaurantMenus.Price;
        _OrdersMenuList.Add(oneOrdersMenu);

_RestaurantTablesProvider.UpdateBusyStatus(Convert.ToInt32(RestaurantTablesCBox.Selected
Value), true);
        LoadDataOrdersMenu(_OrdersMenuList);
        AllSumLbl.Text = "Загальна сума: " + GetAllSumm(_OrdersMenuList).ToString();
      }
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _OrdersProvider.InsertOrders(StartDateDTP.Value,
Convert.ToInt32(ClientsCBox.SelectedValue), Convert.ToInt32(WorkersCBox.SelectedValue),
          Convert.ToInt32(RestaurantTablesCBox.SelectedValue), DescriptionTBox.Text);
        int OrdersId = _OrdersProvider.GetLastRecords();
        for (int i = 0; i < _OrdersMenuList.Count; i++) {
          _OrdersMenuList[i].OrdersId = OrdersId;
        }

_RestaurantTablesProvider.UpdateBusyStatus(Convert.ToInt32(RestaurantTablesCBox.Selected
Value), true);
        _OrdersMenuProvider.InsertBatchOrdersMenu(_OrdersMenuList);
        LoadAllDate();
        MessageBox.Show("Замовлення прийняте", "");
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
```

```
  ClearAllControls();
}

private void ExitBtn_Click(object sender, EventArgs e) {
  this.Close();
}

private void LoadAllDate() {
  _ClientsList = _ClientsProvider.GetAllClients();
  ClientsCBox.DataSource = _ClientsList;
  ClientsCBox.ValueMember = "ClientsId";
  ClientsCBox.DisplayMember = "FIO";

  _RestaurantTablesList = _RestaurantTablesProvider.GetAllNotBusyRestaurantTables();
  RestaurantTablesCBox.DataSource = _RestaurantTablesList;
  RestaurantTablesCBox.ValueMember = "RestaurantTablesId";
  RestaurantTablesCBox.DisplayMember = "RestaurantTablesName";

  _WorkersList = _WorkersProvider.GetAllWorkers();
  WorkersCBox.DataSource = _WorkersList;
  WorkersCBox.ValueMember = "WorkersId";
  WorkersCBox.DisplayMember = "FIO";

  _CategoriesList = _CategoriesProvider.GetAllCategories();
  CategoriesCBox.DataSource = _CategoriesList;
  CategoriesCBox.ValueMember = "CategoriesId";
  CategoriesCBox.DisplayMember = "CategoriesName";
  _IsCategoriesLoad = true;
  CategoriesCBox_SelectedIndexChanged(CategoriesCBox, EventArgs.Empty);

  StartDateDTP.Value = DateTime.Now;
}

private void LoadDataOrdersMenu(List<OrdersMenu> OrdersMenuList) {
  OrdersMenuView.DataSource = null;
  OrdersMenuView.Columns.Clear();
  OrdersMenuView.AutoGenerateColumns = false;
  OrdersMenuView.RowHeadersVisible = false;

  OrdersMenuView.DataSource = OrdersMenuList;
  DataGridViewColumn RestaurantMenusIdColumn = new DataGridViewTextBoxColumn();
  RestaurantMenusIdColumn.DataPropertyName = "RestaurantMenusId";
  RestaurantMenusIdColumn.Name = "RestaurantMenusId";
  OrdersMenuView.Columns.Add(RestaurantMenusIdColumn);
  OrdersMenuView.Columns[0].Visible = false;

  DataGridViewColumn NameColumn = new DataGridViewTextBoxColumn();
  NameColumn.HeaderText = "Блюдо/напиток";
  NameColumn.DataPropertyName = "MenusName";
  NameColumn.Width = NamesMy.SizeOptins.Names;
  OrdersMenuView.Columns.Add(NameColumn);
```

```
    DataGridViewColumn AmountColumn = new DataGridViewTextBoxColumn();
    AmountColumn.HeaderText = "Кількість";
    AmountColumn.DataPropertyName = "Amount";
    AmountColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
    AmountColumn.Width = NamesMy.SizeOptins.Price;
    OrdersMenuView.Columns.Add(AmountColumn);

    DataGridViewColumn PriceColumn = new DataGridViewTextBoxColumn();
    PriceColumn.HeaderText = "Ціна";
    PriceColumn.DataPropertyName = "Price";
    PriceColumn.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
    PriceColumn.Width = NamesMy.SizeOptins.Price;
    OrdersMenuView.Columns.Add(PriceColumn);

    DataGridViewButtonColumn IsResidesBtn = new DataGridViewButtonColumn();
    IsResidesBtn.Text = "Видалити";
    IsResidesBtn.UseColumnTextForButtonValue = true;
    IsResidesBtn.ToolTipText = "Видалити";
    IsResidesBtn.Width = NamesMy.SizeOptins.DeleteBtnSize;
    OrdersMenuView.Columns.Add(IsResidesBtn);

    for (int i = 0; i < OrdersMenuView.Columns.Count; i++) {
      OrdersMenuView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
    }
  }

  private void ClearAllControls() {
    ClientsCBox.SelectedValue = 1;
    WorkersCBox.SelectedValue = 1;
    RestaurantTablesCBox.SelectedItem = 1;
    DescriptionTBox.Text = "";
    StartDateDTP.Value = DateTime.Now;
    _OrdersMenuList.Clear();
    LoadDataOrdersMenu(_OrdersMenuList);
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (Convert.ToInt32(ClientsCBox.SelectedValue) > 0) {
      ClientsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      ClientsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (Convert.ToInt32(WorkersCBox.SelectedValue) > 0) {
      WorkersValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      WorkersValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (Convert.ToInt32(RestaurantTablesCBox.SelectedValue) > 0) {
```

```
      RestaurantTablesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      RestaurantTablesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     return isCorrect;
    }

    private bool IsRestaurantMenusIdExistInList(int RestaurantMenusId, List<OrdersMenu>
OrdersMenuList) {
      for (int i = 0; i < OrdersMenuList.Count; i++) {
       if (RestaurantMenusId == OrdersMenuList[i].RestaurantMenusId) {
        return false;
       }
      }
     return true;
    }
    private bool IsDataEnteringCorrect2() {
     bool isCorrect = true;
     if (Convert.ToInt32(CategoriesCBox.SelectedValue) > 0) {
      CategoriesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      CategoriesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (Convert.ToInt32(RestaurantMenusCBox.SelectedValue) > 0) {
      RestaurantMenusValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      RestaurantMenusValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_Validation.IsDataConvertToInt(AmountTBox.Text)) {
      AmountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      AmountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if
(!IsRestaurantMenusIdExistInList(Convert.ToInt32(RestaurantMenusCBox.SelectedValue),
_OrdersMenuList)) {
      MessageBox.Show("Ви не може додати дане блюдо/напиток, тая як вже його додали!",
"Увага!");
      isCorrect = false;
     }
     return isCorrect;
    }

    private void OrdersMenuView_CellClick(object sender, DataGridViewCellEventArgs e) {
     if (e.ColumnIndex == 4 && _OrdersMenuList.Count > 0) {
      int _RestaurantMenusId = Convert.ToInt32(OrdersMenuView["RestaurantMenusId",
e.RowIndex].Value.ToString());
      for (int i = 0; i < _OrdersMenuList.Count; i++) {
```

```
        if (_RestaurantMenusId == _OrdersMenuList[i].RestaurantMenusId) {
          _OrdersMenuList.RemoveAt(i);
          LoadDataOrdersMenu(_OrdersMenuList);
          AllSumLbl.Text = "Загальна сума: " + GetAllSumm(_OrdersMenuList).ToString();
          break;
        }
      }
    }
  }

  private double GetAllSumm(List<OrdersMenu> OrdersMenuList) {
    double allSum = 0;
    for (int i = 0; i < OrdersMenuList.Count; i++) {
      allSum += OrdersMenuList[i].Amount * OrdersMenuList[i].Price;
    }
    return allSum;
  }

  private void CategoriesCBox_SelectedIndexChanged(object sender, EventArgs e) {
    if (_IsCategoriesLoad) {
      _RestaurantMenusList =
_RestaurantMenusProvider.GetAllRestaurantMenusByCategoriesId(Convert.ToInt32(Categories
CBox.SelectedValue));
      RestaurantMenusCBox.DataSource = _RestaurantMenusList;
      RestaurantMenusCBox.ValueMember = "RestaurantMenusId";
      RestaurantMenusCBox.DisplayMember = "MenusName";
    }
  }
 }
}
```

Лістинг 10. Код класу «FinishOrdersForm»

```
using RestaurantApp.AppCode;
using RestaurantApp.Providers;
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 RestaurantApp.Forms.Controls {
 public partial class FinishOrdersForm : Form {
  private int _selectedRowIndex = 0;
  private OrdersProvider _OrdersProvider = new OrdersProvider();
  private List<Orders> _OrdersList = new List<Orders>();

  public FinishOrdersForm() {
    InitializeComponent();
    DataLoad();
```

```
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (OrdersGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = OrdersGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _OrdersList = _OrdersProvider.GetAllFinishOrders();
        LoadDataInGridView(_OrdersList);
        if (_selectedRowIndex == OrdersGridView.Rows.Count) {
          _selectedRowIndex = OrdersGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          OrdersGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          OrdersGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInGridView(List<Orders> OrdersList) {
      OrdersGridView.DataSource = null;
      OrdersGridView.Columns.Clear();
      OrdersGridView.AutoGenerateColumns = false;
      OrdersGridView.RowHeadersVisible = false;

      OrdersGridView.DataSource = OrdersList;

      if (OrdersList.Count > 0) {
        if (OrdersList[0].Message == NamesMy.NoDataNames.NoDataInOrders) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = OrdersGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
          OrdersGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn AppointmentSheetIdColumn = new
DataGridViewTextBoxColumn();
          AppointmentSheetIdColumn.DataPropertyName = "RestaurantTablesId";
          OrdersGridView.Columns.Add(AppointmentSheetIdColumn);
          OrdersGridView.Columns[0].Visible = false;

          DataGridViewColumn OrdersIdColumn = new DataGridViewTextBoxColumn();
          OrdersIdColumn.Name = "OrdersId";
          OrdersIdColumn.DataPropertyName = "OrdersId";
          OrdersIdColumn.Visible = false;
          OrdersGridView.Columns.Add(OrdersIdColumn);

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
```

```
            numberColumn.Width = NamesMy.SizeOptins.NumberSize;
            OrdersGridView.Columns.Add(numberColumn);

            DataGridViewColumn StartDateColumn = new DataGridViewTextBoxColumn();
            StartDateColumn.HeaderText = "Дата/час";
            StartDateColumn.DataPropertyName = "StartDate";
            StartDateColumn.Width = NamesMy.SizeOptins.Date;
            OrdersGridView.Columns.Add(StartDateColumn);

            DataGridViewColumn ClientsFIOColumn = new DataGridViewTextBoxColumn();
            ClientsFIOColumn.HeaderText = "Клієнт";
            ClientsFIOColumn.DataPropertyName = "ClientsFIO";
            ClientsFIOColumn.Width = 180;
            OrdersGridView.Columns.Add(ClientsFIOColumn);

            DataGridViewColumn WorkersFIOColumn = new DataGridViewTextBoxColumn();
            WorkersFIOColumn.HeaderText = "Офіціант";
            WorkersFIOColumn.DataPropertyName = "WorkersFIO";
            WorkersFIOColumn.Width = 180;
            OrdersGridView.Columns.Add(WorkersFIOColumn);

            DataGridViewColumn RestaurantTablesColumn = new DataGridViewTextBoxColumn();
            RestaurantTablesColumn.HeaderText = "Столик";
            RestaurantTablesColumn.DataPropertyName = "RestaurantTablesName";
            RestaurantTablesColumn.Width = 150;
            OrdersGridView.Columns.Add(RestaurantTablesColumn);
          }
          for (int i = 0; i < OrdersGridView.Columns.Count; i++) {
            OrdersGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
          }
        }
      }
    }
  }
```